Merge-and-Shrink Abstractions for Classical Planning

Theory, Strategies, and Implementation

Inauguraldissertation
zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Silvan Sievers

aus Freiburg, Deutschland

Basel, 2018

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Malte Helmert,
Universität Basel, Dissertationsleiter, Fakultätsverantwortlicher

Prof. Dr. Jörg Hoffmann
Universität des Saarlandes, Korreferent

Basel, den 17.10.2017

Prof. Dr. Martin Spiess,
Universität Basel, Dekan

To Judica

Abstract

Classical planning is the problem of finding a sequence of deterministic actions in a state space that lead from an initial state to a state satisfying some goal condition. The dominant approach to optimally solve planning tasks is heuristic search, in particular A^* search combined with an admissible heuristic. While there exist many different admissible heuristics, we focus on abstraction heuristics in this thesis, and in particular, on the well-established merge-and-shrink heuristics.

Our main theoretical contribution is to provide a comprehensive description of the merge-and-shrink framework in terms of transformations of transition systems. Unlike previous accounts, our description is fully compositional, i.e. can be understood by understanding each transformation in isolation. In particular, in addition to the name-giving merge and shrink transformations, we also describe pruning and label reduction as such transformations. The latter is based on generalized label reduction, a new theory that removes all of the restrictions of the previous definition of label reduction. We study the four types of transformations in terms of desirable formal properties and explain how these properties transfer to heuristics being admissible and consistent or even perfect. We also describe an optimized implementation of the merge-and-shrink framework that substantially improves the efficiency compared to previous implementations.

Furthermore, we investigate the expressive power of merge-and-shrink abstractions by analyzing factored mappings, the data structure they use for representing functions. In particular, we show that there exist certain families of functions that can be compactly represented by so-called non-linear factored mappings but not by linear ones.

On the practical side, we contribute several non-linear merge strategies to the merge-and-shrink toolbox. In particular, we adapt a merge strategy from model checking to planning, provide a framework to enhance existing merge strategies based on symmetries, devise a simple score-based merge strategy that minimizes the maximum size of transition systems of the merge-and-shrink computation, and describe another framework to enhance merge strategies based on an analysis of causal dependencies of the planning task.

In a large experimental study, we show the evolution of the performance of merge-and-shrink heuristics on planning benchmarks. Starting with the state of the art before the contributions of this thesis, we subsequently evaluate all of our techniques and show that state-of-the-art non-linear merge-and-shrink heuristics improve significantly over the previous state of the art.

Zusammenfassung

Klassisches Planen ist das Problem, in einem Zustandsraum eine Abfolge von deterministischen Aktionen zu finden, welche von einem Anfangszustand zu einem Zustand führt, der eine Zielbedingung erfüllt. Der häufigste Ansatz, Plannungsprobleme optimal zu lösen, ist heuristische Suche, und insbesondere A*-Suche in Kombination mit einer zulässigen Heuristik. Während es viele verschiedene zulässige Heuristiken gibt, konzentrieren wir uns in dieser Dissertation auf Abstraktionsheuristiken und insbesondere die bewährten Merge-and-shrink-Heuristiken.

Unser theoretischer Hauptbeitrag ist die umfassende Beschreibung des Merge-and-shrink-Frameworks in Form von Transformationen von Transitionssystemen. Im Gegensatz zu früheren Darstellungen ist unsere Beschreibung modular, d.h. kann durch Verstehen aller isolierten Transformationen verstanden werden. Insbesondere beschreiben wir nicht nur die namensgebenden Merge- und Shrink-Transformationen, sondern auch Pruning und Label Reduction als solche Transformationen. Letztere basiert auf einer verallgemeinerten Theorie von Label Reduction, welche alle Einschränkungen der vorherigen Definition von Label Reduction aufhebt. Wir untersuchen die vier Typen von Transformationen im Hinblick auf wünschenswerte formale Eigenschaften und erläutern, wie sich diese Eigenschaften auf Heuristiken auswirken, sodass diese zulässig und konsistent oder sogar perfekt sind. Des Weiteren beschreiben wir eine optimierte Implementierung des Merge-and-shrink-Frameworks, welche substanziell effizienter als vorherige Implementierungen ist.

Weiterhin untersuchen wir die Ausdrucksstärke von Merge-and-shrink-Abstraktionen, indem wir faktorisierte Abbildungen, die von Merge-and-shrink zur Darstellung von Funktionen verwendeten Datenstrukturen, analysieren. Insbesondere zeigen wir, dass es bestimmte Familien von Funktionen gibt, die sich mit sogenannten nicht-linearen faktorisierte Abbildungen kompakt darstellen lassen können, nicht aber mit linearen.

Auf der praktischen Seite tragen wir mehrere nicht-lineare Mergestrategien zum Merge-and-shrink-Portfolio bei. Insbesondere übertragen wir eine Mergestrategie aus dem Bereich der Modellprüfung zum Planen, beschreiben ein Framework zur Verbesserung existierender Mergestrategien durch Verwendung von Symmetrien, entwickeln eine einfache wertbasierte Mergestrategie zur Minimierung der maximalen Größe von Transitionssystemen während der Merge-and-shrink-Berechnung und beschreiben ein weiteres Framework zur Verbesserung existierender Mergestrategien, welches auf einer Analyse von Kausalabhängigkeiten des Planungsproblems beruht.

In einer groß angelegten experimentellen Studie zeigen wir die Entwicklung der Performanz von Merge-and-shrink-Heuristiken auf Planungs-Benchmarks auf. Beginnend mit dem Stand der Technik vor den Beiträgen dieser Dissertation evaluieren wir anschließend alle unsere Techniken und zeigen, dass nicht-lineare Merge-and-shrink-Heuristiken des neuesten Stands der Technik signifikant besser sind als vorherige Merge-and-shrink-Heuristiken.

Acknowledgments

First of all, I would like to thank my advisor Malte Helmert. Without the planning lecture he gave in the winter term 2008 at the university of Freiburg, I would not likely be where I am now: about to graduate. This lecture was my entry to the world of planning and academia, which I have been faithful to since then mainly because Malte always was (and still is of course) a great teacher and supervisor, which I quickly realized when working on several projects with him. His ability to quickly answer questions which I did not even fully understand myself in the first place is amazing and extremely helpful for everyone working with him.

I also want to thank my other great colleagues Salomé Eriksson, Cedric Geissmann, Manuel Heusner, Thomas Keller, Florian Pommerening, Gabi Röger, Jendrik Seipp, and Martin Wehrle. It it always a pleasure to work with them, and I appreciate a lot the familiar atmosphere we have in the research group in Basel, which forms the basis of many interesting discussions related to research, but also related to any other topic.

I am also very happy to have been able to attend many interesting conferences, starting with ICAPS 2011 in Freiburg, where I met numerous friendly people from all over the world. It is great to be part of such a friendly community which allows uncomplicated scientific exchange and collaborations. In particular, I would like to thank Michael Katz for our collaborations and many fruitful discussions, be it via emails, telephone conference, or in person at different conferences.

I also want to thank Jörg Hoffmann for co-reviewing my thesis, and in particular for agreeing on the "extremely sportive timeline" for reviewing, as he phrased it himself. I am further very thankful to those who proofread my thesis on a very short note, helping me to adhere to the aforementioned timeline: Cedric Geissmann, Manuel Heusner, Thomas Keller, Florian Pommerening, Jendrik Seipp, Agazi Tesfai, Lam Tran, and Daniel Wendler. Their comments were very helpful in finding mistakes and helping improve the text.

Finally, I owe a huge thanks to everyone in my social environment. In the first place, I want to thank my girlfriend Judica, for her continued support and bearing numerous deadlines where I had to work on weekends and during nights, and also for pushing and stopping me at the right times. I am also grateful to my family Rainer, Renate, Ruben, and Rudi, for their continued support and to render possible my studies in the first place. Finally, I am very thankful for all past and current, close and not-so-close friends with whom I spent so much great time during the five years of my PhD studies (sorry if I forgot anyone): Agazi, Amadeus, Andrea, Chris, Daniel, Domi, Hannes, Jelena, Jojo, Julia, Julian, Kessi, Krisi, Lam, Leon, Leonie, Lucas, Lydia, Marian, Mateo, Milena, Natalie, Philip, Robin, Salome, Sebastian, Sophie, and Tobi, and last but not least, in memoriam Patrick. Thanks also to the co-singers in my choir for all the great concerts and tours we had.

Contents

1. Introduction

Automated planning (Ghallab, Nau, & Traverso, 2004) is one of the oldest disciplines of artificial intelligence. Planning deals with finding a sequence of actions that leads from an initial situation of the world to some desirable goal situation, or to show that no such solution exists. For example, a logistics planning task could be to transport several packages, which initially are in different cities, to their destinations, using trucks to drive between cities or airplanes to cover larger distances. Other so-called planning domains include, but are not limited to, single-player games such as FreeCell or Sokoban, solving combinatorial puzzles like the 15-puzzle, green-house logistics and other logistics-like problems, space applications such as navigation and scheduling of a mars rover and scheduling of satellites, other scheduling tasks such as airport security or printing in large printer networks, controlling an elevator, and many more. The key of *domain-independent* planing, however, is to solve such planning tasks of different domains in a domain- and problem-agnostic way, i.e. only using the structural description of the planning task at hand.

In this thesis, we deal with *classical* planning, which is the subset of planning that works with planning tasks which are fully observable, deterministic, discrete, and single-agent problems, which means that there is no adversary or other sources of non-determinism or uncertainty. Furthermore, we are interested in finding *optimal* solutions, i.e. solutions that minimize cost. Classical planning tasks can be compactly described and are often specified using the planning description language PDDL (e.g. McDermott et al., 1998; Fox & Long, 2003; Edelkamp & Hoffmann, 2004). Most modern planning systems ground the lifted PDDL representation of planning tasks into STRIPS or SAS^+ representations.

One of the most prominent approaches of domain-independent planners of the last decade is (explicit) *heuristic search* (Bonet & Geffner, 2001). With this approach, planners cast the problem of solving a planning task as finding a path in the *state space* induced by the planning task. To this end, they use a search algorithm in conjunction with a *heuristic function* (Pearl, 1984) that estimates costs-to-go in the state space for search guidance. While the induced state spaces are typically too large to be explicitly represented in memory and exhaustively searched, they can be searched using the black box model of the state space provided by the compact description of the planning task. The most common approach is to use *explicit (forward) search*, which represents states explicitly and uses data structures like queues and hash sets to store individual states encountered during search. Many enhancements for and variations of standard forward search algorithms have been suggested, e.g. symmetry-breaking (e.g. Fox & Long, 1999, 2002; Pochter, Zohar, & Rosenschein, 2011; Domshlak, Katz, & Shleyfman, 2012, 2015), partial order reduction (e.g. Valmari, 1989; Wehrle & Helmert, 2012; Wehrle, Helmert, Alkhazraji, & Mattmüller, 2013), decoupled search (Gnad & Hoffmann, 2015), and many others.

Furthermore, there are alternatives to explicit heuristic search: symbolic (heuristic) search (McMillan, 1993), in contrast to explicit search, expands sets of states simultaneously, using

Binary Decision Diagrams (Bryant, 1986) to represent such sets of states, which is much more efficient than explicitly representing sets of states. While blind symbolic search (often regression or bidirectional search) is already powerful, it has also been combined with heuristics (e.g. Kissmann & Edelkamp, 2011; Edelkamp, Kissmann, & Torralba, 2012; Torralba, Linares López, & Borrajo, 2013, 2016; Torralba, Alcázar, Kissmann, & Edelkamp, 2017). Another alternative that does not use heuristics at all is SAT-based planning (e.g. Rintanen, Heljanko, & Niemelä, 2006; Rintanen, 2012), which casts planning tasks as satisfiability problems and uses off-the-shelf SAT solvers to solve the encoded planning tasks.

In this thesis, we focus on explicit heuristic search, and since we want to solve planning tasks optimally, we use the A* algorithm (Hart, Nilsson, & Raphael, 1968). An A* search finds optimal paths in directed graphs if the heuristic used is *admissible*, i.e. underestimates the real cost-to-go. A natural question to ask is how to come up with domain-independent (admissible) heuristics. From research in classical planning of the recent decades, four major classes of heuristics have emerged (Helmert & Domshlak, 2009). Delete relaxation heuristics are based on a simplification of planning tasks that ignores all "negative" effects (e.g. Bonet & Geffner, 2001; Hoffmann & Nebel, 2001; Hoffmann, 2005; Keyder, Hoffmann, & Haslum, 2014; Domshlak, Hoffmann, & Katz, 2015). Critical path heuristics estimate the cost of reaching a subgoal of a task by the cost of the most expensive subsets of the subgoal being reached (e.g. Haslum & Geffner, 2000; Haslum, 2009). Landmark heuristics compute sets of actions of which at least one action must be taken in any plan and estimate the cost-to-go by the cost of the actions (e.g. Richter & Westphal, 2010; Helmert & Domshlak, 2009; Keyder, Richter, & Helmert, 2010; Bonet & Helmert, 2010; Bonet & Castillo, 2011). Finally, *abstraction heuristics* (see below) compute exact plan costs in an abstracted variant of the planning task which only represents some aspects of the planning task, but not all of it.

Besides these four classes of heuristics, techniques such as *operator cost partitioning* can be used to combine heuristics (Katz & Domshlak, 2008; Yang, Culberson, Holte, Zahavi, & Felner, 2008; Katz & Domshlak, 2010b, 2007; Edelkamp, 2006; Pommerening, Röger, & Helmert, 2013). More recently, there has been an increasing interest in general operator cost partitioning (Pommerening, Helmert, Röger, & Seipp, 2015) and heuristics that can be defined in the *operator counting framework* (Pommerening, Röger, Helmert, & Bonet, 2014; Seipp, Pommerening, & Helmert, 2015; Pommerening, Helmert, & Bonet, 2017), where constraints of different types of heuristics can be combined. Another line of work investigates different types of cost partitionings for Cartesian abstractions (Seipp, Keller, & Helmert, 2017b, 2017a; Seipp, 2017).

In this thesis, we focus on using (single) abstraction heuristics to solve planning tasks optimally. Lots of research has been devoted to devising abstraction heuristics such as *pattern databases* (Culberson & Schaeffer, 1998; Edelkamp, 2001; Felner, Korf, & Hanan, 2004; Holte, Felner, Newton, Meshulam, & Furcy, 2006; Felner, Korf, Meshulam, & Holte, 2007; Haslum, Botea, Helmert, Bonet, & Koenig, 2007), implicit abstractions (Katz & Domshlak, 2010a), heuristics based on abstraction refinement (e.g. Knoblock, 1994; Seipp & Helmert, 2013; Bäckström & Jonsson, 2013), or, more generally, to finding admissible ways of combining abstraction heuristics (e.g. Felner et al., 2004; Holte et al., 2006; Yang et al., 2008; Katz & Domshlak, 2010b). The particular abstraction heuristic we investigate in this thesis is the class of *merge-and-shrink* heuristics, originally introduced for directed model checking (Dräger, Finkbeiner, & Podelski, 2006, 2009), and later adapted to planning (Helmert, Haslum, & Hoffmann, 2007;

Helmert, Haslum, Hoffmann, & Nissim, 2014). Merge-and-shrink abstractions are a flexible class of abstractions which dominate many other classes of classical planning heuristics in terms of expressive power (Helmert & Domshlak, 2009), including the abstractions underlying pattern databases (e.g. Helmert et al., 2014). Furthermore, they are among the few heuristic approaches that can derive perfect heuristics in polynomial time in nontrivial cases (Nissim, Hoffmann, & Helmert, 2011; Helmert et al., 2014), thus offering polynomial-time optimal planning algorithms for these cases.

The merge-and-shrink algorithm operates on a *factored transition system*, i.e. a set of explicitly represented transition systems, also called factors, which together represent the *product transition system*, such as the transition system induced by a planning task. For example, the state space of a SAS^+ planning task (Bäckström & Klein, 1991) can be described by a factored transition system where each factor is an *atomic transition system*, describing how the operators of the planning task affect a single state variable. The process of computing a merge-and-shrink abstraction begins with the factored transition system consisting of all atomic transition systems and then iteratively applies *transformation steps* to the current factored transition system until only a single factor remains. In parallel, merge-and-shrink algorithms also keep track of the abstraction mapping from the given factored transition system to the current one, using so-called *factored mappings*, which are tree-like data structures representing arbitrary functions defined on variable assignments. In the end, the final factored mapping represents an abstraction from the product transition system of the given factored transition system to the final factor.

One example of transformations used by the computation are *merge* steps, which combine two factors into a single factor representing their joint behavior. Another example are *shrink* steps, which replace a single factor by a local abstraction, i.e. a smaller transition system that overapproximates the behavior of this factor. Merge steps and shrink steps are the most prominent transformations underlying merge-and-shrink abstractions, as can be seen from the fact that the approach is named after them. However, they are not the only important transformations in the framework.

Starting with the first work on merge-and-shrink abstractions in classical planning (Helmert et al., 2007), all practical merge-and-shrink implementations have additionally included *pruning* steps that eliminate states that can be shown not to be part of any solution, and *label reduction* steps that modify the labels of state transitions, for example by combining multiple labels into one. Pruning was only treated as a side node in previous work and label reduction was originally implemented (but not described) as a crucial efficiency improvement for the computation of merge-and-shrink heuristics in the Fast Downward planning system (Helmert, 2006). Later, Nissim et al. (2011) showed that label reduction in conjunction with bisimulation-based shrinking allows the computation of perfect heuristics in polynomial time for some planning domains.

In this thesis, we investigate the merge-and-shrink framework, making the contributions described in the following section.

1.1. Contributions

The first and major contribution of this thesis is the introduction of *generalized label reduction* to the merge-and-shrink framework, replacing the previous theory of label reduction (Helmert et

al., 2014), which had several severe shortcomings. Firstly, the "old" label reduction is very complicated, spanning 7 pages of definitions and theorems in the main paper plus more than 2 pages of proofs in an appendix in order to establish the conditions under which label reduction can be safely applied. Secondly, these conditions restrict safe application of label reductions to only one branch of the "merge tree" that represents the order of merge transformations, which led all previous merge-and-shrink algorithms to only use so-called *linear* merge strategies. Thirdly, the old label reduction breaks the clean view of merge-and-shrink abstractions as composable transformations of arbitrary factored transition systems because it requires keeping track of structured information for each label based on the representation of the underlying planning operators. This requires "looking into" the planning task and that the allowed label reductions also depend on shrinking. In contrast to this old label reduction, generalized label reduction is easy to understand, always safe to apply, and fully conforms to the view of merge-and-shrink transformations as local transformations of arbitrary factored transition systems with no restrictions regarding the "internal structure" of transitions.

Furthermore, we make the following theoretical, algorithmic, and practical contributions to the merge-and-shrink framework, which are often based on generalized label reduction or *non-linear* merge-and-shrink:

- We provide a comprehensive description of the merge-and-shrink framework in terms of transformations of factored transition systems. In addition to the *merge* and *shrink* transformations, we show how *label reduction* and *pruning* (usually of unreachable states or states from which the goal cannot be reached) fit naturally as generic transformations of factored transition systems. The main contribution here is the new theory of label reduction, but we also provide the first detailed and formal account of the prune transformation. We study these four types of transformations in terms of formal properties, such as exactly preserving the behavior of the joint transition system represented by the factored transition system, overapproximating this behavior, or preserving/overapproximating the behavior on the set of reachable states only.

- We discuss the efficient implementation of merge-and-shrink abstractions. Specifically, we describe improvements over the previous implementation of merge-and-shrink in the Fast Downward planning system (Helmert, 2006; Helmert et al., 2014) in terms of memory usage and runtime by collapsing locally equivalent labels of transition systems and further simplifications made possible by the fact that the new theory of label reduction permits treating labels as opaque objects rather than operators of a planning task.

- We analyze the expressive power of factored mappings, the data structures used by merge-and-shrink to represent arbitrary functions defined on assignments over finite-domain variables, and prove that non-linear factored mappings are strictly more powerful than linear ones by showing that there exist problem families that can be represented compactly with general factored mappings but not with linear ones. We also give a precise bound that quantifies the necessary blowup incurred by conversions from general factored mappings to linear ones.

- We provide a literature overview of merge and shrink strategies that existed before our addition of generalized label reduction to the merge-and-shrink framework. Furthermore,

we review related work which is based on the theory of generalized label reduction or more generally, which makes use of non-linear merge-and-shrink.

- We devise several new non-linear merge strategies that improve the state of the art of merge-and-shrink heuristics:
  - We transfer the concept of structural symmetries (Shleyfman, Katz, Helmert, Sievers, & Wehrle, 2015) to factored transitions systems, defining *factored symmetries*. We investigate under which conditions factored symmetry reduction (i.e. shrinking based on factored symmetries) yields perfect heuristics and discuss the relationship to the state-of-art shrink strategy based on bisimulation (Nissim et al., 2011). We devise a framework to enhance existing merge strategies by altering them to preferably merge transition systems that are symmetric under factored symmetries, increasing the amount of opportunities for perfect shrinking based on bisimulation.
  - We adapt the original merge strategy from model checking to planning, naming it DFP after their authors (Dräger et al., 2006), which is a simple "score-based" merge strategy that assigns a score to each merge candidate. As an alternative to the rather complex precomputed MIASM strategy, we devise a score-based alternative to minimize the maximum size of transition systems of the merge-and-shrink computation. We further describe tie-breaking strategies that can be used with such score-based merge strategies.
  - We describe how to combine a-priori information from the strongly connected components (SCCs) of the *causal graph* with simple score-based merge strategies in the SCC framework: in a first step, the factors of the factored transition system are partitioned according to the SCCs and then the score-based merge strategy is used to first merge all factors within partitions before combining the resulting products.
- We perform a large experimental study:
  - We evaluate the impact of label reduction, showing that it has a large positive influence on performance and that generalized label reduction increases performance even further. We also show that the first non-linear merge strategy DFP outperforms all previous linear ones.
  - Using the framework to enhance merge strategies by using factored symmetries, we show that the computation of symmetry-enhanced merge strategies is more efficient and that they often produce heuristics of higher quality. We also evaluate alternative ways of using symmetries, such as pruning the A^* search or using symmetric lookups, and combine the techniques as well.
  - We show that an optimized implementation that takes advantage of generalized label reduction by treating labels as completely opaque tokens increases the efficiency of the merge-and-shrink computation significantly.
  - In an analysis of merge strategies, we show that there is an untapped potential of merge strategies by evaluating all possible merge strategies on small planning tasks and large sets of randomly sampled merge strategies on all planning benchmarks.

Furthermore, we investigate the effects of tie-breaking on score-based merge strategies, showing large variations of performance. We also evaluate the SCC framework with different secondary merge strategies.

– Finally, we provide an overview of different state-of-the-art merge-and-shrink configurations and also compare them to other abstraction heuristics and state-of-the-art planners.

1.2. Publications

Most results presented in this thesis have been previously published, mostly in major conferences on artificial intelligence or automated planning. This section lists these and other publications in chronological order, however grouped by their relevance for this thesis. For each paper, we briefly describe its contents and provide pointers to the sections where it is discussed in this thesis.

The following papers are of central importance for this thesis, as they all directly deal with merge-and-shrink abstractions:

- The paper *Generalized Label Reduction for Merge-and-Shrink Heuristics* (Sievers, Wehrle, Helmert; AAAI 2014) introduces the theory of generalized label reduction to the merge-and-shrink framework and shows that it is more powerful than the previous theory both conceptually and experimentally. In the paper, we also adapt the original merge strategy from model checking to planning and evaluate it.

  The paper won an **Honorable Mention for the Outstanding Paper Award** at AAAI 2014 (shared with three other candidates for the outstanding paper award).

  We discuss generalized label reduction in Sections 3.6 and 3.8, describe DFP in Section 5.4, and evaluate the impact of (generalized) label reduction and DFP in the experiments in Section 6.2.

- The paper *Factored Symmetries for Merge-and-Shrink Abstractions* (Sievers, Wehrle, Helmert, Shleyfman, Katz; AAAI 2015) defines factored symmetries of factored transition systems based on the notion of symmetries as used previously in planning for symmetry-based pruning in a forward state-space search. In the paper, we show that shrinking based on so-called atomic factored symmetries is captured by shrinking based on bisimulation and as such is information-preserving. Based on this insight, we devise a framework to enhance existing merge strategies through using information about factored symmetries so that the merge process is guided towards maximizing the amount of exact shrinking.

  Section 5.3 is based on this paper in large parts, excluding the experimental evaluation which we discuss in Section 6.3.

- The paper *On the Expressive Power of Non-Linear Merge-and-Shrink Representations* (Helmert, Röger, Sievers; ICAPS 2015) investigates the representational power of merge-and-shrink. It shows that non-linear merge-and-shrink representations (called factored

mappings in this thesis) are strictly more powerful than linear ones in that they can compactly represent problem families which linear merge-and-shrink cannot compactly represent. It further gives a precise bound that quantifies the necessary blowup incurred by conversions from general factored mappings to linear ones.

Chapter 4 is based on this paper in large parts.

- The paper *An Analysis of Merge Strategies for Merge-and-Shrink Heuristics* (Sievers, Wehrle, Helmert; ICAPS 2016) analyses all possible merge strategies on small planning tasks as well as large sets of randomly sampled merge strategies on the entire benchmark set and shows an untapped potential of existing merge strategies. The paper further investigates the influence of tie-breaking on the merge strategy DFP, evaluates a simple score-based variant of the MIASM strategy, and finally describes a hybrid merge strategy that uses precomputed information from the causal graph and the score-based merge strategy DFP to achieve state-of-the-art performance among merge-and-shrink heuristics.

  We discuss the new merge strategies and the tie-breaking criteria of this paper in more detail in Section 5.4. Most of the paper is of experimental nature, and we present its results in our experimental study in Section 6.6.

- The paper *Merge-and-Shrink Abstractions for Factored Transition Systems* (Sievers, Wehrle, Helmert; in preparation for submission to JAIR) presents the merge-and-shrink algorithm as a framework operating on factored transition systems and factored mappings. We define transformations of (factored) transition systems and show which desirable properties they have and how these transfer to heuristics induced by the transformation being admissible and consistent, or even perfect. We also provide an algorithmic view of the transformations and discuss their implementation. The paper is based on the first paper of this list, but it also includes a discussion of related work that appeared after the introduction of generalized label reduction and contains a larger experimental evaluation.

  Chapter 3 of this thesis is based on this work-in-progress paper in large parts.

The following papers are more tangential to the central topic of this thesis, but still mentioned or discussed:

- In the paper *Efficient Implementation of Pattern Database Heuristics for Classical Planning* (Sievers, Ortlieb, Helmert; SoCS 2012), we describe the efficient implementation of PDBs in Fast Downward.

  While this thesis does not investigate PDB heuristics, we briefly discuss the differences of factored mappings to PDBs in Section 3.2.3, in particular with respect to their expressive power. We also compare merge-and-shrink heuristics against abstraction heuristics based on PDBs in Section 6.9.

- The paper *Heuristics and Symmetries in Classical Planning* (Shleyfman, Katz, Sievers, Wehrle, Helmert; AAAI2015) introduces the notion of *structural symmetries*, subsuming several previous definitions of symmetries for planning. We show that many heuristics based on delete relaxation, landmarks, and critical paths are invariant under symmetry

in the sense that, given two symmetric states, they are guaranteed to compute the same estimate.

We introduce structural symmetries in Section 2.4 and evaluate their application for symmetry-based pruning and symmetric lookups, also in combination with factored symmetries, in Section 6.5.

- The paper *An Empirical Case Study on Symmetry Handling in Cost-Optimal Planning as Heuristic Search* (Sievers, Wehrle, Katz, Helmert; KI 2015), among other abstraction heuristics, evaluates the three alternatives of using symmetries for merge-and-shrink and the combinations thereof: for pruning an A^* search, for using symmetric lookups over several merge-and-shrink heuristics, and for using factored symmetries to enhance the merge-and-shrink heuristic.

  We briefly compare these alternatives experimentally in Section 6.5.

Finally, for completeness, we also include a list of unrelated publications that were written in parallel to the publications relevant to this thesis:

- In the paper *Bounded Intention Planning Revisited* (Sievers, Wehrle, Helmert; ECAI 2014), we show that the operator partitioning of bounded intention planning (Wolfe & Russell, 2011) corresponds to strong stubborn sets (Valmari, 1989; Wehrle & Helmert, 2012, 2014) under certain conditions.

- We took part in the International Planning Competition (IPC) 2014 with three entries:
  - *Fast Downward Cedalion* (Seipp, Sievers, Hutter; IPC Deterministic Part 2014) is an automatically learned sequential portfolio of Fast Downward configurations for both the satisficing and optimal setting. To let the learning algorithm also possibly choose merge-and-shrink configurations, we added rudimentary support of conditional effects in merge-and-shrink, since this was a requirement of IPC 2014. The portfolio we learned for the optimal deterministic competition indeed uses a merge-and-shrink configuration, however it is still based on the previous theory of label reduction with linear merge strategies, and hence we do not discuss this portfolio in more detail in this thesis.
  - *Fast Downward Cedalion* (Seipp, Sievers, Hutter; IPC Planning and Learning Part 2014) uses the same technique as above for the learning part.
  - *Fast Downward SMAC* (Seipp, Sievers, Hutter; IPC Planning and Learning Part 2014) is a variant of Fast Downward Cedalion that learns a single best configuration rather than a sequential portfolio.

  All three planners are based on the same learning technique which we describe in the paper *Automatic Configuration of Sequential Planning Portfolios* (Seipp, Sievers, Helmert, & Hutter, 2015), see below.

  Results:

  - In the deterministic setting, Fast Downward Cedalion achieved the 7th place out of 20 competitors in the satisficing track, and the 8th place out of 17 competitors in the optimal track.

  - Out of 11 competitors in the learning part, Fast Downward Cedalion won the **second place of the best quality award** and the **first place of the best learner award**.
  - Out of 11 competitors in the learning part, Fast Downward SMAC won the **third place of the best quality award** and the **third place of the best learner award**. It also won the **first place of the best basic solver award**, out of 5 competitors, granted to non-portfolio planners.

- In the paper *Automatic Configuration of Sequential Planning Portfolios* (Seipp, Sievers, Hutter; AAAI 2015), we describe Cedalion, a conceptually simple approach for the idea of combining portfolios and algorithm configuration, that greedily searches for the $\langle$parameter configuration, runtime$\rangle$ pair which, when appended to the current portfolio, maximizes the portfolio improvement per additional runtime spent. We provide theoretical guarantees that Cedalion yields portfolios within a constant factor of optimal for the training set distribution. In an experimental evaluation, we show that Cedalion produces portfolios of Fast Downward configurations that compete with and even outperform competitors in the satisficing, agile, and learning setting.

- The paper *A Doppelkopf Player Based on UCT* (Sievers, Helmert; KI 2015) introduces the German card game doppelkopf as a benchmark for research on AI games and provides a baseline player based on the UCT algorithm (Kocsis & Szepesvári, 2006).

- The planner abstract *Fast Downward Aidos* (Seipp, Pommerening, Sievers, Wehrle, Fawcett, Alkhazraji; UIPC 2016) describes our submission to the First Unsolvability International Planning Competition. The three planners we submitted are portfolios, and the automatically configured variant AIDOS 3 contains several configurations that use merge-and-shrink with generalized label reduction. Since the two manually configured variants AIDOS 1 & 2 do not contain merge-and-shrink configurations and perform better, and since it is hard to quantify the contribution of merge-and-shrink heuristics in AIDOS 3, we do not discuss this paper further in this thesis.

  Aidos is the **winner of the UIPC 2016**, outperforming 10 competitors.

- In the paper *Graph-Based Factorization of Classical Planning Problems* (Wehrle, Sievers, Helmert; IJCAI 2016), we propose a generic approach for factorizing a classical planning problem into an equivalent problem with fewer operator and variable dependencies. Our approach is based on variable factorization, which can be reduced to the well-studied problem of graph factorization. While the state spaces of the original and the factorized problems are isomorphic, the factorization offers the potential to exponentially reduce the complexity of planning techniques like factored planning and partial order reduction.

  Presumably, variable factorization can be formalized as a transformation of our transformation framework: it can be understood as "inverse merging", however using the Cartesian product instead of the synchronized product used by the merge transformation.

- The paper *Strengthening Canonical Pattern Databases with Structural Symmetries* (Sievers, Wehrle, Helmert, Katz; SoCS 2017) can be understood as a continuation of our work on factored symmetries for merge-and-shrink in that it describes a way of using structural

symmetries to enhance PDB heuristics. More precisely, we show how to strengthen the canonical pattern databases heuristic by adding symmetric pattern databases, making the resulting heuristic invariant under structural symmetry, thus making it especially attractive for symmetry-based pruning search methods. Further, we prove that this heuristic is at least as informative as using symmetric lookups over the original heuristic. An experimental evaluation confirms these theoretical results.

- In the paper *Structural Symmetries of the Lifted Representation of Classical Planning Tasks* (Sievers, Röger, Wehrle, Katz; ICAPS 2017 HSDIP Workshop), we transfer the notion of structural symmetries to lifted planning task representations, based on a generalizing concept of abstract structures we use to model planning tasks. We show that symmetries are preserved by common grounding methods and shed some light on the relation to previous symmetry concepts. We discuss promising applications for which this paper lays the foundation.

1.3. Outline

This thesis is structured as follows. In Chapter 2, we describe the necessary background. In particular, we define classical planning tasks and describe how we can use heuristic search to solve them (optimally). Furthermore, we introduce the concepts of abstractions and symmetries.

Chapter 3 then describes the merge-and-shrink framework. We begin with defining transformations of transition systems as a general concept and show that transformations can have desirable properties. These properties in turn transfer to properties of heuristics induced by the transformations. We then turn our attention to factored representations as used by the merge-and-shrink framework: factored transition systems describe transition systems such as those induced by planning tasks and factored mappings are data structures that are suited to represent functions defined on assignments over finite-domain variables, such as mappings from states of factored transition systems to states of single transition systems. Furthermore, we show that our transformation framework can also operate directly on factored transition systems. We then define the four merge-and-shrink transformations, showing for each which properties it has and how it can efficiently be composed with other transformations. Armed with all ingredients of the merge-and-shrink framework, we describe the merge-and-shrink algorithm in full detail. Finally, we discuss generalized label reduction in the light of the previous theory of label reduction, describe an algorithm to compute label reductions, and provide the means for an optimized implementation of merge-and-shrink based on generalized label reduction.

In Chapter 4, we show that non-linear factored mappings are strictly more powerful than linear ones in the sense that they can compactly represent problem families which no linear factored mapping can compactly represent. We also provide precise bounds on the necessary blowup incurred by converting a non-linear factored mapping to a linear one.

Chapter 5 gives a complete overview of transformation strategies used in practical merge-and-shrink algorithms. In particular, we discuss related work that was presented before and after the addition of generalized label reduction. Furthermore, we describe our own contributions to the merge-and-shrink toolbox in detail.

In Chapter 6, we provide an extensive experimental study of merge-and-shrink heuristics, showing the evolution of their performance starting with linear merge-and-shrink using the previous theory of label reduction and ending with state-of-the-art configurations that use the newest non-linear merge strategies. The study includes an evaluation of the impact of label reduction, the framework based on factored symmetries, the optimized implementation based on generalized label reduction, and all non-linear merge strategies we devised. We also compare against other state-of-the-art planning techniques.

Finally, Chapter 7 concludes this thesis with a summary of our contributions and a discussion of future work.

2. Background

In this section, we describe the necessary background. In particular, we define classical planning tasks and describe how we can use heuristic search to solve them (optimally). Furthermore, we introduce the concepts of abstractions and heuristics induced by abstractions. Finally, we briefly touch on the topic of symmetries and their use for heuristics and search algorithms.

2.1. Classical Planning and Transition Systems

The formalism we consider in this thesis for the presentation of planning tasks and heuristics is the SAS$^+$ formalism (Bäckström & Nebel, 1995), augmented with action costs. In our experimental study reported in Chapter 6, we use the STRIPS (Fikes & Nilsson, 1971) subset of PDDL planning tasks, which can be efficiently translated into SAS$^+$ tasks (e.g. Helmert, 2009). For both formalizations of planning tasks, the problem of deciding whether a plan exists for a task of a given but unspecified domain is PSPACE-complete (Bäckström & Nebel, 1995).

SAS$^+$ planning tasks are based on finite-domain variables, which we define next.

Definition 2.1 (Variable, Fact)**.** *A* finite-domain variable v *is a variable which can take values from its associated finite domain* $dom(v)$*. A* fact $\langle v, x\rangle$ *is a pair consisting of a variable* v *and one its values* $x \in dom(v)$*.*

Given a set $\mathcal{V}$ of finite-domain variables, we define states over $\mathcal{V}$ as assignments to the variables in $\mathcal{V}$.

Definition 2.2 ((Partial) State)**.** *Let* $\mathcal{V}$ *be a set of finite-domain variables. A* partial state s *is a partial function over* $\mathcal{V}$*, mapping each variable* $v \in vars(s)$ *to a value from* $dom(v)$*. By* $vars(s)$*, we denote the domain of the partial function, i.e. the variables for which the partial state is defined. If* $vars(s) = \mathcal{V}$*, then* s *is called a state.*

We write (partial) states s as sets of assignments to values for all variables in $vars(s)$, e.g. $s = \{v \mapsto x, w \mapsto y\}$ for variables $\{v, w\} = vars(s)$ and values $x \in dom(v), y \in dom(w)$. By $s[v]$ we denote the value of v in s. We say that two partial states s and s' *comply* if $s[v] = s'[v]$ for all variables $v \in vars(s) \cap vars(s')$.

Planning tasks also have operators to modify states.

Definition 2.3 (Operator)**.** *Let* $\mathcal{V}$ *be a set of finite-domain variables. An* operator o *has an associated partial state* $pre(o)$ *over* $\mathcal{V}$ *called the* precondition *of* o*, an associated partial state* $\mathit{eff}(o)$ *over* $\mathcal{V}$ *called the* effect *of* o*, and an associated non-negative value* $cost(o) \in \mathbb{R}_0^+$*, called the* cost *of* o*.*

We can now put the pieces together to define a planning task.

Definition 2.4 (Planning Task)**.** *A* planning task *(in the* SAS$^+$ *formalism) is a tuple* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ *with the following components:*

- $\mathcal{V}$ *is a finite set of finite-domain* variables*.*
- $\mathcal{O}$ *is a finite set of operators.*
- s_0 *is a state, called the* initial state*.*
- $s_\star$ *is a partial state, called the* goal*.*

As we will see below, planning tasks induce *transition systems*, which we define next.

Definition 2.5 (Labeled Transition System)**.** *A* labeled transition system *(or* transition system *for short) is a tuple* $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ *where* S *is a finite set of* states*,* L *is a finite set of transition* labels*,* $c : L \mapsto \mathbb{R}_0^+$ *is a* label cost function*,* $T \subseteq S \times L \times S$ *is a set of* labeled transitions*,* $s_0 \in S$ *is the* initial state *and* $S_\star \subseteq S$ *is the set of* goal states*.*

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

Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system. We write $s \xrightarrow{\ell} s'$ to denote a transition $\langle s, \ell, s' \rangle$ from s to s' with label ℓ, and we may write $s \xrightarrow{\ell} s' \in \Theta$ for $s \xrightarrow{\ell} s' \in T$, and similarly, $s \in \Theta$ for $s \in S$, whenever Θ is not specified further. A *path* from s to s' is a sequence $\pi = \langle t_1, \ldots, t_n \rangle$ of transitions such that there exist states $s = s_0, \ldots, s_n = s'$ with $t_i = \langle s_{i-1}, \ell_i, s_i \rangle \in T$ for all $i \in \{1, \ldots, n\}$. The *cost* of such a path is the accumulated cost of the labels of the transitions, i.e. $\sum_{i=1}^{n} c(\ell_i)$.

We can now define the semantics of a planning task in terms of its induced transition system.

Definition 2.6 (Induced Transition System)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ *be a planning task. The* induced transition system $\Theta(\Pi)$ *is defined as* $\langle S, L, c, T, s_0, S_\star \rangle$ *with the following components:*

- S *is the set of states over* $\mathcal{V}$*.*
- $L = O$*, i.e.* L *contains a label* o *for each operator* $o \in \mathcal{O}$*.*
- c *is a cost function with* $c(o) = cost(o)$ *for all* $o \in \mathcal{O}$*.*
- $s \xrightarrow{\ell} t \in T$ *iff* $pre(\ell)$ *complies with* s *and* t *is the state complying with* $eff(\ell)$ *and* $t[v] = s[v]$ *for all* $v \notin vars(eff(\ell))$*.*
- s_0 *is the initial state of the planning task.*
- $S_\star = \{s \in S \mid s \text{ complies with } s_\star\}$*.*

$\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ with
$\mathcal{V} = \{v_T, v_P\}$ with $dom(v_T) = \{\text{A}, \text{B}, \text{C}\}$ and $dom_(v_P) = \{\text{A}, \text{B}, \text{C}, \text{T}\}$
$\mathcal{O} = \{$DRIVE-A-B : $pre($DRIVE-A-B$) = \{v_T \mapsto \text{A}\}$ $\quad$ $\mathit{eff}($DRIVE-A-B$) = \{v_T \mapsto \text{B}\}$
$cost($DRIVE-A-B$) = 1$,
DRIVE-A-C : $pre($DRIVE-A-C$) = \{v_T \mapsto \text{A}\}$ $\quad$ $\mathit{eff}($DRIVE-A-C$) = \{v_T \mapsto \text{C}\}$
$cost($DRIVE-A-C$) = 1$,
DRIVE-B-A : $pre($DRIVE-B-A$) = \{v_T \mapsto \text{B}\}$ $\quad$ $\mathit{eff}($DRIVE-B-A$) = \{v_T \mapsto \text{A}\}$
$cost($DRIVE-B-A$) = 1$,
DRIVE-B-C : $pre($DRIVE-B-C$) = \{v_T \mapsto \text{B}\}$ $\quad$ $\mathit{eff}($DRIVE-B-C$) = \{v_T \mapsto \text{C}\}$
$cost($DRIVE-B-C$) = 1$,
DRIVE-C-A : $pre($DRIVE-C-A$) = \{v_T \mapsto \text{C}\}$ $\quad$ $\mathit{eff}($DRIVE-C-A$) = \{v_T \mapsto \text{A}\}$
$cost($DRIVE-C-A$) = 1$,
DRIVE-C-B : $pre($DRIVE-C-B$) = \{v_T \mapsto \text{C}\}$ $\quad$ $\mathit{eff}($DRIVE-C-B$) = \{v_T \mapsto \text{B}\}$
$cost($DRIVE-C-B$) = 1$,
LOAD-A : $pre($LOAD-A$) = \{v_T \mapsto \text{A}, v_P \mapsto \text{A}\}$ $\quad$ $\mathit{eff}($LOAD-A$) = \{v_P \mapsto \text{T}\}$
$cost($LOAD-A$) = 1$,
LOAD-B : $pre($LOAD-B$) = \{v_T \mapsto \text{B}, v_P \mapsto \text{B}\}$ $\quad$ $\mathit{eff}($LOAD-B$) = \{v_P \mapsto \text{T}\}$
$cost($LOAD-B$) = 1$,
LOAD-C : $pre($LOAD-C$) = \{v_T \mapsto \text{C}, v_P \mapsto \text{C}\}$ $\quad$ $\mathit{eff}($LOAD-C$) = \{v_P \mapsto \text{T}\}$
$cost($LOAD-C$) = 1$,
UNLOAD-A : $pre($UNLOAD-A$) = \{v_T \mapsto \text{A}, v_P \mapsto \text{T}\}$ $\quad$ $\mathit{eff}($UNLOAD-A$) = \{v_P \mapsto \text{A}\}$
$cost($UNLOAD-A$) = 1$,
UNLOAD-B : $pre($UNLOAD-B$) = \{v_T \mapsto \text{B}, v_P \mapsto \text{T}\}$ $\quad$ $\mathit{eff}($UNLOAD-B$) = \{v_P \mapsto \text{B}\}$
$cost($UNLOAD-B$) = 1$,
UNLOAD-C : $pre($UNLOAD-C$) = \{v_T \mapsto \text{C}, v_P \mapsto \text{T}\}$ $\quad$ $\mathit{eff}($UNLOAD-C$) = \{v_P \mapsto \text{C}\}$
$cost($UNLOAD-C$) = 1\}$
$s_0 = \{v_T \mapsto \text{A}, v_P \mapsto \text{B}\}$
$s_\star = \{v_P \mapsto \text{C}\}$

Figure 2.1.: Simple logistics planning task with one truck, one package and three locations.

Intuitively, the states of the induced transition system are the states over the variables of the planning task, and the labels identify the operators of the planning task. There is a transition from s to t with label ℓ if the operator o identified by ℓ is *applicable* in s, i.e.. if the precondition of o complies with s. The *successor state* t is defined as the state that complies with the effect of o for all variables for which this effect is defined, and for all other variables, it complies with the original state s. The initial state is the same as in the planning task and the set of goal states contains all states complying with the goal condition of the planning task.

Solutions for planning tasks are called *plans*. A plan π for Π is a path from the initial state to some goal state in $\Theta(\Pi)$, and its cost is the cost of the path. *Optimal planning* is the problem of finding a cost-minimal plan for a given planning task or showing that no plan exists.

We now illustrate the above definitions of planning tasks and (induced) transition systems with the help of our running example, which is a simple logistics planning task with one truck T and one package P. There are three locations A, B, and C, and the truck T can move from and to all locations. The package P can be loaded into and unloaded from truck T if both are at the same location.

Figure 2.1 shows a typical SAS$^+$ representation of the planning task using two variables v_T

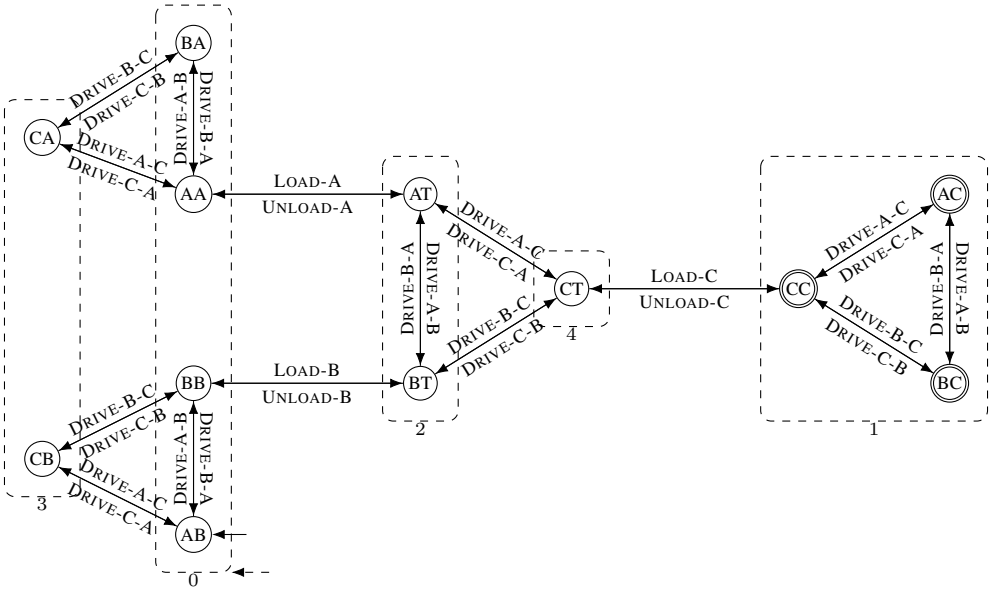


Figure 2.2.: Induced transition system $\Theta(\Pi)$ of the example planning task Π of Figure 2.1.

and v_P that encode the position of the truck and the package, respectively. The initial state is the situation where the truck is at A and the package at B. The goal condition is a partial state which states that the package is at C. It is undefined for v_T, i.e. the truck can be at any of the locations in a goal state. The truck can move with DRIVE operators with the obvious precondition and effect on the position variable v_T of the truck. To load the package into the truck, both have to be at the same location, and the effect is that the package is in the truck, i.e. $v_P \mapsto$ T. Analogously for unloading, the package has to be in the truck, and the effect is that the package is at the same location as the truck afterwards, i.e. $v_P \mapsto$ X if $v_T \mapsto$ X.

We illustrate transition systems as follows. We draw states as circles with labels identifying the state. Transitions are shown as arrows, labeled with the label of the transitions. As our example is invertible, i.e. for every operator, there is an operator inverting the effect, we draw double-headed arrows to denote that the there are two transitions, one in each direction between the neighboring states, each labeled with one of the two labels next to the arrow. We follow no rule which of the labels correspond to which direction, hoping that it is clear from the context. Goal states are marked with a double circle, and the initial state has an unlabeled arrow ingoing from no source node.

Consider now Figure 2.2 which shows the induced transition system $\Theta(\Pi)$ of the planning task. Ignore the dashed boxes for the moment. In this example, the states are labeled with two letters, the first denoting the position of the truck and the second denoting the position of the package. For example, the initial state AB of the figure corresponds to the state $\{v_T \mapsto A, v_P \mapsto \text{B}\}$ where the truck is at A and the package at B. All states whose second component reads C are goal states because the package is at C in such states. Looking at the transition AB $\xrightarrow{\text{DRIVE-A-B}}$ BB, it is clear that the corresponding operator is applicable in the state AB because the truck is at A, and that the state BB is the successor state because the truck is at B

and the position of the package did not change.

Before closing this section, we also define the concept of the causal graph (Knoblock, 1994), which most of the merge strategies we consider in Chapter 5 are based on.

Definition 2.7 (Causal Graph)**.** *Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, the* causal graph (CG) *for Π is the directed graph $G = \langle \mathcal{V}, E \rangle$, where E contains the following edges: for all variables $v \neq w \in \mathcal{V}$ and operators $o = \langle pre(o), eff(o) \rangle \in \mathcal{O}$, if $v \in pre(o)$ and $w \in eff(o)$, there is an edge $\langle v, w \rangle \in E$, and if $v, w \in eff(o)$, there are edges $\langle v, w \rangle \in E$ and $\langle w, v \rangle \in E$.*

The causal graph captures causal dependencies of a planning task: an edge between two variables indicates that changing the value of one of the variables depends on the value of the other one

In our example, the causal graph contains two nodes for the two variables v_T and v_P. There is a single edge from v_T to v_P induced by the LOAD and UNLOAD operators because they have a precondition on v_T and an effect on v_P.

2.2. Heuristic Search

As we address optimal classical planning through the means of (explicit) heuristic search, we are not only interested in deriving merge-and-shrink abstractions, but also in defining heuristics based on these abstractions. Hence, we briefly recall the concept of heuristics (e.g. Pearl, 1984).

Definition 2.8 (Heuristic)**.** *Let Θ be a transition system with states S. A* heuristic *for Θ is a function $h_\Theta : S \to \mathbb{R}_0^+ \cup \{\infty\}$ that maps each state $s \in S$ to a value, called the* estimate *or* heuristic value *of s. It estimates the cost of reaching a goal state from s.*

Heuristics can have desirable properties which we define next.

Definition 2.9 (Properties of Heuristics)**.** *Let Θ be a transition system with states S and label cost function c. Then a heuristic h_Θ for Θ is called*

- perfect for state $s \in \Theta$ *if it maps s to the* true *cost of a cheapest path from s to any goal state of Θ or ∞ if no such path exists. We write $h^*_\Theta(s)$ for the perfect heuristic value of s.*
- goal-aware for state $s \in \Theta$ *if $h_\Theta(s) = 0$ or s is no goal state.*
- consistent for transition $s \xrightarrow{\ell} t \in \Theta$ *if $h_\Theta(s) \leq c(\ell) + h_\Theta(s')$.*
- admissible for state $s \in \Theta$ *if $h_\Theta(s) \leq h^*_\Theta(s)$.*

If the heuristic is perfect for all states of Θ, we write h^*_Θ and call it the perfect heuristic. Analogously, h_Θ is called goal-aware or admissible if it is goal-aware or admissible for all states of Θ, and it is called consistent if it is consistent for all transitions of Θ. We also write h instead of h_Θ if the transition system is clear from context.

To solve planning tasks optimally, we use the A* algorithm (Hart et al., 1968). The A* algorithm is a best-first search that uses a priority queue which orders search nodes according to the f-value of their state. The f-value of a state s is defined as the sum of the cost to reach s

from the initial state (g-value) and the estimated cost to reach a goal state from s (h-value). Hart et al. (1968) prove that A* produces optimal solutions if used with admissible heuristics and that it never needs to reopen a node if the heuristic is also consistent. We argue that these claims still hold for an even less restricted class of heuristics, namely heuristics which are admissible only for the subset of *reachable* states or consistent only for transitions between reachable states, because only such states can be encountered and hence evaluated by the heuristic in any forward search, including A*. As a consequence, the heuristic values of unreachable states do not matter, and we can allow heuristics to estimate the cost-to-go of unreachable states with arbitrary values without affecting the properties of the heuristic for all other states.

The following definition formally introduces the concepts of dead and alive states.

Definition 2.10 (Reachable, Relevant, Dead, and Alive States)**.** *Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system. A state $s \in S$ is* unreachable *iff there exists no path from s_0 to s, otherwise it is* reachable*. It is* irrelevant *iff there exists no path from s to some state $s' \in S_\star$, otherwise it is* relevant*. A state s that is unreachable or irrelevant is called* dead*; otherwise it is reachable and relevant and called* alive*.*

Let Θ be a transition system with reachable states R. We call a heuristic h_Θ *forward-goal-aware* if it is goal-aware for all states $s \in R$; *forward-consistent* if it is consistent for all transitions between states $s, t \in R$; *forward-admissible* if it is admissible for all states $s \in R$; and *forward-perfect* if it is perfect for all states $s \in R$.

The following proposition formulates our above argumentation concerning A* used with heuristics that are admissible or consistent only for the subset of the reachable states, i.e. forward-admissible heuristics.

Proposition 2.1. *The* A* *algorithm produces optimal solutions if used with forward-admissible heuristics. It never needs to reopen a node if the heuristic is also forward-consistent.*

2.3. Abstractions

Since we deal with abstraction heuristics in this thesis, we define abstractions and heuristics induced by abstractions in this section. In particular, we present two prominent abstraction heuristics, namely *pattern databases* (e.g. Culberson & Schaeffer, 1998; Edelkamp, 2001) and Cartesian abstractions (Seipp & Helmert, 2013). We describe merge-and-shrink abstractions in Chapter 3, where we will also define abstractions in more detail when defining them as a transformation with specific properties within the transformation framework used by the merge-and-shrink algorithm.

Definition 2.11 (Abstraction Mapping, Induced Abstract Transition System)**.** *Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system. An* abstraction mapping *(or* abstraction*) of Θ is a function $\alpha : S \mapsto S^\alpha$, and the* induced abstract transition system Θ^α *is defined as $\langle S^\alpha, L, c, T^\alpha, s_0^\alpha, S_\star^\alpha \rangle$ where $T^\alpha = \{\langle \alpha(s), \ell, \alpha(t) \rangle \mid \langle s, \ell, t \rangle \in T\}$, $s_0^\alpha = \alpha(s_0)$, and $S_\star^\alpha = \{\alpha(s) \mid s \in S_\star^\alpha\}$.*

An abstraction is a *state mapping* α that maps from the states of the given transition system to states of the abstract transition system. The abstract transition system in turn is induced by

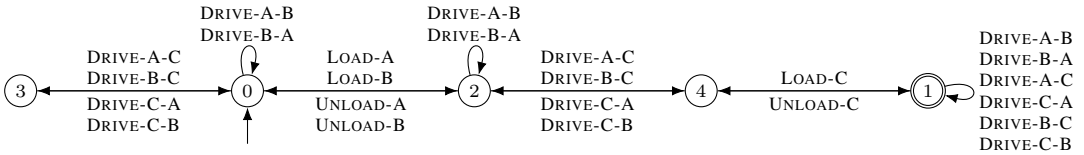


Figure 2.3.: Induced abstract transition system $\Theta(\Pi)^\alpha$ of the induced transition system of the example planning task Π of Figure 2.1.

α, i.e. the initial state is mapped to the *abstract initial state*, goal states are mapped to *abstract goal states*, and for each transition between two states in the given transition system, there is an *abstract transition* with the same label between the two abstract states. We call α an abstraction because it usually maps to a set of abstract states that is smaller than the original set of states.

As example, consider again the logistics planning task of Figure 2.1. Figure 2.2, showing the induced transition system $\Theta(\Pi)$, includes an example abstraction mapping α, illustrated with the dashed boxes: all states within a single dashed box are mapped to the same abstract state, labeled with the number shown next to the box. For example, $\alpha(\text{AC}) = \alpha(\text{BC}) = \alpha(\text{CC}) = 1$, and $\alpha(\text{CA}) = \alpha(\text{CB}) = 3$.

Figure 2.3 shows the induced abstract transition system $\Theta(\Pi)^\alpha$. We observe that, for example, all transitions between the states AC, BC, and CC in the original transition system induce self-looping transitions at state 1, onto which the three original states are mapped. Furthermore, state 1 is a goal state because at least one of the original states mapped to it is a goal state. (In this case, all of the original states are goal states.) The state 0 is the initial state because the original initial state AB is mapped to 0.

Definition 2.12 (Abstraction Heuristic)**.** *Let Θ be a transition system and let α be an abstraction of Θ. Then the* abstraction heuristic *for Θ induced by α is defined as $h^\alpha_\Theta(s) = h^*_{\Theta^\alpha}(s)$ for all states s of Θ.*

Informally speaking, an abstraction heuristic for a transition system and an abstraction α computes perfect heuristic values in the abstract transition system induced by the abstraction α. Abstraction heuristics as defined above are admissible and consistent, a well-known result (Helmert, Haslum, & Hoffmann, 2008) that we repeat in Chapter 3, adapted to our notion of transformations.

In our example, the value of the abstraction heuristic $h^\alpha_{\Theta(\Pi)}$ for the initial state is 3, because in $\Theta(\Pi)^\alpha$, any optimal plan starting in state $\alpha(\text{AB}) = 0$, e.g. $\langle$LOAD-A, DRIVE-A-C, UNLOAD-C$\rangle$, has cost 3.

In the following, we restrict our attention to a particular class of abstractions, called projections, which form the basis for pattern database heuristics.

Definition 2.13 (Projection and PDB Heuristic)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task and let S be the set of states over $\mathcal{V}$. Let $P \subseteq \mathcal{V}$, called a* pattern*, and let S' be the set of states over P. A* projection *is a function $\pi_P : S \mapsto S'$ where $\pi_P(s) = s|_P$. An abstraction heuristic induced by a projection π_P is called a* PDB heuristic*, written h^{π_P} or h^P for short.*

Intuitively, a projection is an abstraction that perfectly preserves all aspects of a planning task described with variables in the pattern and disregards all other aspects. Typically, a PDB

heuristic is precomputed by computing all heuristic values for all states S' over the pattern P and storing them in a one-dimensional table (called the pattern database, from which stems the name of the heuristic). With a perfect hash function that can be efficiently computed (e.g. Sievers, Ortlieb, & Helmert, 2012), this PDB then allows simple lookups of heuristic values during search.

We remark that for SAS$^+$ planning tasks Π as we consider them here, we can compute the abstract transition system induced by the projection by applying the *syntactic projection* onto Π, denoted by $\Pi|_P$, which means to "ignore" all information not pertaining to variables from P, and then computing the transition system of $\Pi|_P$. Further note that the example abstraction shown above and illustrated in Figures 2.2 and 2.3 is *not* a projection, because, for example, values A and B of variable v_T are not distinguished in the abstraction, but value C is perfectly distinguishable. For the abstraction to be a projection, either all or no values of v_T have to be distinguishable.

This example also shows the main limitation of PDBs: they are limited to either entirely or not at all taking a variable into consideration. Merge-and-shrink heuristics are more flexible than PDB heuristics in that they allow arbitrary combinations of both variables and values, and they subsume PDB heuristics, although possibly incurring a polynomial overhead (e.g. Helmert et al., 2014). We compare the representations underlying PDB heuristics and merge-and-shrink heuristics in Section 3.2.3.

Despite this limitation, PDB heuristics have a long success story both for heuristic search and planning and there is a large amount of research addressing questions such as how to select patterns, how to combine several PDB heuristics, or how to efficiently compute and store PDB heuristics (e.g. Felner et al., 2004; Holte et al., 2006; Edelkamp, 2006; Felner et al., 2007; Haslum et al., 2007; Sievers et al., 2012). For example, Haslum et al. (2007) define a simple criterion for PDBs to be additive, which means that their heuristic values can be summed up without violating admissibility. Based on this type of additivity, for a set of patterns (pattern collections) C, they define the canonical PDB (CPDB) heuristic as follows. Let A be the maximal (w.r.t. set inclusion) additive subsets of C. Then the CPDB heuristic for C is defined as

$$h^C(s) = \max_{B \in A} \sum_{P \in A} h^P(s)$$

Intuitively, the heuristic computes the sum whenever this is admissible and the maximum otherwise. Haslum et al. also present a hill climbing procedure that performs a search in the space of pattern collections, aiming at obtaining pattern collections which yield the best results with the CPDB heuristic. The combination of hill climbing with the CPDB heuristic is commonly called *iPDB*.

A different type of abstraction heuristics are the so-called *Cartesian abstraction heuristics* (Seipp & Helmert, 2013). A set of states is called Cartesian if it can be written as the Cartesian product of subsets of the domain of each variable, and an abstraction is called Cartesian if all abstract states induced by the abstraction are Cartesian sets. This means that in a simple example with two binary variables v, w that can take values in $\{0, 1\}$, an abstraction that maps $\{v \mapsto 0, w \mapsto 0\}$ and $\{v \mapsto 1, w \mapsto 1\}$ into an abstract state s and $\{v \mapsto 0, w \mapsto 1\}$ and $\{v \mapsto 1, w \mapsto 0\}$ into another abstract state t is *not* Cartesian because neither s nor t are Cartesian sets of states: s uses all values of the two variable domains (hence $\{0, 1\} \times \{0, 1\}$ is a candidate for

a Cartesian set describing s), however the states $\{v \mapsto 0, w \mapsto 1\}$ and $\{v \mapsto 1, w \mapsto 0\}$ are not mapped to s. In contrast, the abstraction example described above (cf. Figures 2.2 and 2.3) is Cartesian because all abstract states are Cartesian sets. For example, the abstract state 3 is the Cartesian set $\{\mathrm{C}\} \times \{\mathrm{A}, \mathrm{B}\}$, and the abstract state 0 is the Cartesian set $\{\mathrm{A}, \mathrm{B}\} \times \{\mathrm{A}, \mathrm{B}\}$.

The advantage of Cartesian abstractions is their increased granularity compared to PDBs and that they allow efficient refinement steps of the *counter-example guided abstraction refinement (CEGAR)* algorithm (e.g. Clarke, Grumberg, & Peled, 1999), which, starting from the coarsest abstraction that maps all states into a single state, iteratively refines the abstraction by splitting abstract states (Seipp & Helmert, 2013). While Cartesian abstractions subsume PDBs, merge-and-shrink abstractions are more general than Cartesian abstractions because they can represent every abstraction function. However, it is unknown whether merge-and-shrink abstractions can compactly represent every Cartesian abstraction, i.e. with at most a polynomial overhead, similarly to how they subsume PDBs (Seipp & Helmert, 2013).

Like PDBs, Cartesian abstraction heuristics are also more powerful if used in an ensemble: Seipp and Helmert (2014) suggest computing a set of diverse Cartesian abstractions by guiding the abstraction refinement process with different strategies, and to use *saturated cost partitioning* on the abstractions to ensure that their combination remains admissible. In our experimental study in Chapter 6, we compare against both iPDB and a state-of-the-art heuristic that is a combination of several cost partitionings over a diverse set of abstractions, including PDBs and Cartesian abstractions (Seipp, 2017).

2.4. Symmetries

While the application of symmetries outside of merge-and-shrink is not the focus of this thesis, factored symmetries as we define them for the merge-and-shrink framework in Section 5.3 are inspired by the notion of symmetries as they have been successfully applied in classical planning. Examples include symmetry-based pruning (e.g. Fox & Long, 1999, 2002; Pochter et al., 2011; Domshlak et al., 2012, 2015), state space transformations (e.g. Riddle, Douglas, Barley, & Franco, 2016), performing symmetric lookups (e.g. Felner, Zahavi, Schaeffer, & Holte, 2005; Felner et al., 2011; Sievers, Wehrle, Helmert, & Katz, 2015), and enhancing heuristics through the incorporation of information on symmetries (Sievers, Wehrle, Helmert, Shleyfman, & Katz, 2015; Sievers, Wehrle, Helmert, & Katz, 2017). Most of these techniques are based on ground representations of planning tasks, such as STRIPS or SAS$^+$, but recently, there have also been efforts to investigate symmetries of lifted representations such as PDDL (Riddle et al., 2016; Sievers, Röger, Wehrle, & Katz, 2017).

To lay the foundation of factored symmetries as we discuss them later, we formally introduce *structural symmetries* for SAS$^+$ planning tasks in this section. Shleyfman et al. (2015) showed that structural symmetries subsume several of previous concepts of symmetries. We further describe two applications of structural symmetries in classical planning which is also evaluate in our experimental study in Section 6.5.

We borrow the definition of structural symmetries for SAS$^+$ planning tasks and some of the following text from Sievers, Wehrle, Helmert, and Katz (2017).

Definition 2.14 (Structural Symmetry)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be a planning task, and let F be*

the set of facts *over* $\mathcal{V}$. *A* structural symmetry *for* Π *is a permutation* $\sigma : \mathcal{V} \cup F \cup \mathcal{O} \to \mathcal{V} \cup F \cup \mathcal{O}$, *where*

1. $\sigma(\mathcal{V}) = \mathcal{V}$ *and* $\sigma(F) = F$ *such that* $\sigma(\langle v, d \rangle) = \langle v', d' \rangle$ *implies* $v' = \sigma(v)$;
2. $\sigma(\mathcal{O}) = \mathcal{O}$ *such that for* $o \in \mathcal{O}$, $\sigma(pre(o)) = pre(\sigma(o))$, $\sigma(\mathit{eff}(o)) = \mathit{eff}(\sigma(o))$, $cost(\sigma(o)) = cost(o)$;
3. $\sigma(s_\star) = s_\star$;

where $\sigma(\{x_1, \ldots, x_n\}) := \{\sigma(x_1), \ldots, \sigma(x_n)\}$, *and for a partial state* s, $s' := \sigma(s)$ *is the partial state obtained from* s *such that for all* $v \in vars(s)$, $\sigma(\langle v, s[v] \rangle) = \langle v', d' \rangle$ *implies* $s'[v'] = d'$.

Structural symmetries are also called goal-stable automorphisms because they preserve the structure of the planning task with respect to goal states and paths to goal states.

We remark that given a structural symmetry σ, its application to sets or tuples X is naturally defined as the set or tuple of element-wise applications of σ. The set of all structural symmetries Γ_Π of a planning task Π forms a group under the composition operation. In practice, a set of structural symmetries that generates (a subgroup of) the symmetry group Γ_Π can be efficiently computed using off-the-shelf tools for discovery of automorphisms in explicit graphs (Pochter et al., 2011; Shleyfman et al., 2015). For simplicity, in what follows, by a symmetry group Γ we refer to a subgroup of the symmetry group Γ_Π of the planning task Π.

As an example, consider again the simple logistics task of Figure 2.1, this time extended to include a second truck U. This extended task contains an additional variable v_U with $dom(v_U) = dom(v_T)$. Furthermore, there is a copy of each operator for truck U, e.g. DRIVE$_U$-A-B, where the occurrences of v_T are replaced by v_U. Finally, the domain of variable v_P has an additional value U. Intuitively, the two trucks T and U are interchangeable in every aspect of the planning task. Formally, this modified task exhibits a structural symmetry σ that permutes the two trucks. For example, $\sigma(v_T) = v_U$ and $\sigma(\langle v_T, \mathrm{X} \rangle) = \langle v_U, \mathrm{X} \rangle$ for all $\mathrm{X} \in dom(v_T)$. σ also permutes the operators, e.g. $\sigma(\text{DRIVE-A-B}) = \text{DRIVE}_U\text{-A-B}$, with their preconditions and effects.

The most prominent application of symmetries in planning in recent years is symmetry-based pruning of a forward search such as A*. In a nutshell, symmetry-based pruning search algorithms use symmetries to prune some of the symmetric states encountered during search if previously seen symmetric states have been reached with the same or lower cost. Due to the structure-preserving property of goal-stable automorphisms, a plan from state s exists iff the plan under symmetry (hence of the same cost) exists from the symmetric state s'. This kind of pruning preserves optimality.

One algorithm that implements such symmetry-based pruning and that we consider in our experimental study is *orbit space search (OSS)* (Domshlak et al., 2015). It performs an A* search, however replaces all encountered states by their so-called canonical symmetric representatives (e.g. the lexicographically smallest state of the set of all symmetric states), thus implicitly performing duplicate detection against all symmetric states. To extract a solution after successful termination of the search, the states and operators of the computed plan need to be mapped back to the states and operators that were originally generated during search. For details, we refer to the original paper.

The second alternative we consider for using symmetries during search is to perform *symmetric lookups*, which means to maximize heuristic values over a set of symmetric states for a given state s that should be evaluated. More precisely, for a given heuristic h, a state s and a symmetry group Γ, the *symmetric lookup heuristic over* h is defined as $h_{SL}(s) := \max_{s \in S} h(s)$, where $S := \{s, s^1, \ldots, s^m\}$ is a set of states symmetric to s under structural symmetries from Γ, including s itself. S can be chosen arbitrarily to trade off computation time against informativeness of the symmetric lookups, i.e. $m = 0$ is possible as well as computing the set of all states symmetric to s under Γ.

3. The Merge-and-Shrink Framework

In this central chapter of the thesis, we present the merge-and-shrink framework. The first section, Section 3.1, describes the transformation framework for transition systems, including desirable properties of transformations and how they transfer to heuristics being admissible or perfect. While this section is rather independent of the merge-and-shrink framework, we still include it here because it is a central building block of the merge-and-shrink framework in our formalization. Section 3.2 then describes factored representations of transition systems and (state) mappings, which are the second central concept of the merge-and-shrink framework. It further links planning tasks to factored transition systems and explains the merge-and-shrink algorithm using factored representations. Armed with the transformation framework and factored representations, the forthcoming four sections describe the individual transformations of the merge-and-shrink framework, i.e. shrinking, merging, pruning and label reduction. Then, in Section 3.7, we describe the merge-and-shrink algorithm in full detail. Finally, Section 3.8 discusses generalized label reduction: we compare the new theory with the previous theory of label reduction, describe an algorithm to compute label reductions, and explain how merge-and-shrink based on generalized label reduction can be efficiently implemented.

This chapter in large parts stems from our work-in-progress paper planned to be submitted to JAIR.

3.1. Transformations of Transition Systems

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

Definition 3.1 (Transformation)**.** *Let Θ be a transition system with states S and labels L, and let Θ' be a transition system with states S' and labels L'. A* transformation *of Θ into Θ' is a tuple $\langle \Theta', \sigma, \lambda \rangle$, where Θ' is called the* transformed transition system*, $\sigma : S \to S'$ is called the* state mapping*, and $\lambda : L \to L'$ is called the* label mapping*.*

In words, a transformation of a transition system specifies the transformed transition system, provides a state mapping relating the states of the given transition system to the states of the transformed one, and analogously defines a label mapping relating the label sets.

Transformations can be used to compute mappings between transition systems. In particular, we can use them to compute abstractions of transition systems. For instance, the merge-and-shrink framework repeatedly applies transformations to a given transition system to obtain an

abstract transition system, which can in turn be used to derive a heuristic function for the original transition system (among other uses).

Definition 3.2 (Heuristic Induced by a Transformation)**.** *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system Θ into transition system Θ'. The* heuristic for Θ induced by τ, h^τ_Θ *(or h^τ for short), is defined as $h^\tau_\Theta(s) = h^*_{\Theta'}(\sigma(s))$.*

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

3.1.1. Properties of Transformations

We now define desirable properties of transformations of transition systems. In the following, for a function $f : A \to B$, by f^{-1} we denote the *inverse function* from elements of the image B of f to subsets of the preimage A of f, defined as $f^{-1}(b) = \{a \in A \mid f(a) = b\}$. The inverse function may also be applied to sets $B' \subseteq B$ using the definition $f^{-1}(B') = \{a \in A \mid f(a) \in B'\}$.

Definition 3.3 (Properties of Transformations)**.** *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ into a transition system $\Theta' = \langle S', L', c', T', s'_0, S'_\star \rangle$. The following list defines properties that τ may have, along with a short-hand name for each property. (For example, we say that τ satisfies* $\mathbf{IND_S}$ *if τ is state-induced, as defined in the first list entry.)*

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

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

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

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

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

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

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

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

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

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

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

Based on these basic properties, we define the following derived properties, where $\mathbf{A} = \mathbf{B}+\mathbf{C}$ means that τ has property $\mathbf{A}$ if it has properties $\mathbf{B}$ and $\mathbf{C}$:

- conservative: $\mathbf{CONS} = \mathbf{CONS_T} + \mathbf{CONS_G} + \mathbf{CONS_C}$
- induced: $\mathbf{IND} = \mathbf{IND_S} + \mathbf{IND_L} + \mathbf{IND_T} + \mathbf{IND_G} + \mathbf{IND_C}$
- refinable: $\mathbf{REF} = \mathbf{REF_T} + \mathbf{REF_G} + \mathbf{REF_C}$

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

Finally, we introduce variants of $\mathbf{CONS_T}$ *and* $\mathbf{CONS_G}$ *for a subset* $Q \subseteq S$ *of the states:*

$\mathbf{CONS_T^Q}$ τ *is* transition-conservative for Q *if* $\forall s \xrightarrow{\ell} t \in T$ *with* $s, t \in Q$: $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$.

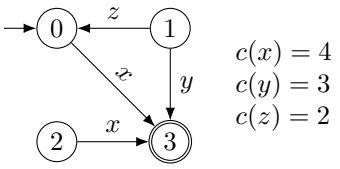
$\mathbf{CONS_G^Q}$ τ *is* goal-conservative for Q *if:* $\forall s \in (S_\star \cap Q)$: $\sigma(s) \in S'_\star$.

All derived properties can be analogously applied with respect to a subset $Q \subseteq S$ *of the states. For example,* τ *is* conservative for Q *(*$\mathbf{CONS}^Q$*) if it satisfies* $\mathbf{CONS_T^Q} + \mathbf{CONS_G^Q} + \mathbf{CONS_C}$*, and it is a* strict homomorphism for Q *if it is* $\mathbf{CONS}^Q + \mathbf{IND}$.

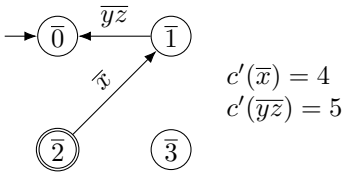
Informally speaking, a transformation τ of a transition system Θ into transition system Θ' is an abstraction (homomorphism) if all behaviors possible in Θ are preserved by τ: every transition of Θ has a corresponding abstract transition in Θ' ($\mathbf{CONS_T}$) of the same or lower cost ($\mathbf{CONS_C}$),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

Among these homomorphisms, strict homomorphism are in some sense the most accurate ones: while they include all transitions and goal states that a homomorphism must include, they do not include any additional transitions ($\mathbf{IND_T}$) or goal states ($\mathbf{IND_G}$) beyond those required by the homomorphism property. They must not contain any abstract states ($\mathbf{IND_S}$) or labels ($\mathbf{IND_L}$) beyond those that the state and label mapping map to. Finally, transformed label costs must correspond to the cost of some original label ($\mathbf{IND_C}$), which together with cost-conservativeness implies that the cost of a transformed label must be the minimum cost of its preimage labels. It is not difficult to show that for every state mapping σ and label mapping λ, there exists a unique transition system Θ' such that $\tau = \langle \Theta', \sigma, \lambda \rangle$ is a strict homomorphism, namely the induced abstract transition system from Definition 2.11. Hence, strict homomorphisms are uniquely described by their state and label mappings, and we say that Θ' is the transition system *induced* by σ and λ. Induced abstractions are practically desirable because they provide the strongest possible heuristics among all abstractions with the same state and label mappings. They are also theoretically desirable because they can be fully understood and analyzed in terms of the state and label mapping.

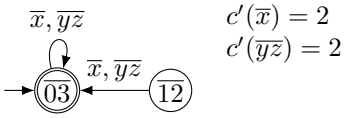
Exact transformations are conservative "in both directions": intuitively, refinability means that all behaviors possible in Θ' are also possible in Θ. All transformed transitions $s' \xrightarrow{\ell'} t'$ can be mapped back to original transitions $s \xrightarrow{\ell} t$ for *all* preimages s of s'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



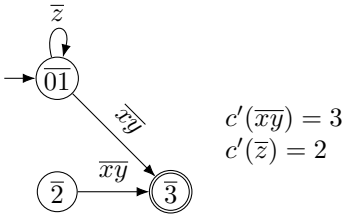
(a) Original transition system.



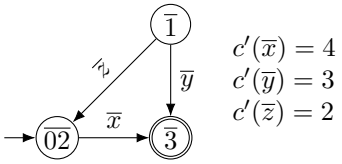
(b) Arbitrary transformation (not an abstraction).



(c) Abstraction (not a strict homomorphism).



(d) Strict homomorphism (not exact).



(e) Exact transformation.

Figure 3.1.: Four different transformations of the transition system in part (a). The captions of part (b)–(e) indicate the properties of the transformation.

the same way. To make this formal, we will prove that heuristics based on exact transformations are perfect.

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

Figure 3.1 illustrates four example transformations with different properties. The original transition system is shown in part (a) of the figure. All other parts of the figures show a transformation of this transition system. We use undecorated numbers and letters to denote states and labels of the original transition system and overlined symbols to denote states and labels of the transformed transition systems. If a state s of the original transition systems is mapped to some state of the transformed transition system, then the name of the state includes $\overline{s}$. For example, a transformed state whose preimage consists of states 0 and 3 is denoted by $\overline{03}$. We proceed analogously for labels. All transformations are state-induced and label-induced, so there are no transformed states and labels other than those that the original state and labels map to.

Figure 3.1b shows the first transformation, an example of a non-abstraction transformation. Indeed, it satisfies none of the properties of a conservative transformation. It is not goal-

conservative because $\overline{3}$ is not a goal state even though 3 is. It is not transition-conservative because the transformed transition system has no transition corresponding to $0 \xrightarrow{x} 3$. Finally, it is not cost-conservative because the cost of $\overline{yz}$ is higher (5) than the costs of y and z (4 and 3).

Figure 3.1c shows an abstraction: all transitions induce abstract transitions, all goal states induce abstract goal states, and all labels are mapped to labels of the same or lower cost. If we removed the transitions from $\overline{12}$ to $\overline{03}$, it would no longer be an abstraction (violating $\mathbf{CONS_T}$), but it would still be an abstraction for the subset of reachable states. The transformation is no strict homomorphism because the transition $\overline{03} \xrightarrow{\overline{yz}} \overline{03}$ is not induced by any original transition (violating $\mathbf{IND_T}$) and also because the label cost 2 of $\overline{x}$ differs from the cost 4 of its only preimage label x (violating $\mathbf{IND_C}$).

Figure 3.1d shows a strict homomorphism: all states, labels, transitions, goal states and label costs are induced by the original transition system. The transformation is not exact for several reasons: for example, it is not transition-refinable because the transition $\overline{01} \xrightarrow{\overline{z}} \overline{01}$ has no matching original transition for the preimage 0 of $\overline{01}$, as neither $0 \xrightarrow{z} 0$ nor $0 \xrightarrow{z} 1$ are transitions of the original transition system. The transformation is also not cost-refinable because the cost of label $\overline{xy}$ (3) is lower than the cost of x (4).

Finally, Figure 3.1e shows an exact induced transformation: it is conservative, induced and refinable.

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

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

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

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

on goal states and label costs, which are not present in their formalism.

Bäckström and Jonsson further define the converse properties $\mathbf{R}_\downarrow$ and $\mathbf{C}_\downarrow$ in the obvious way and call a transformation satisfying $\mathbf{M}_\uparrow\mathbf{R}_\updownarrow\mathbf{C}_\updownarrow$ a strong homomorphism. This corresponds to adding $\mathbf{IND_T}$ to $\mathbf{CONS_T}$. Again, for a transformation to be a strong (strict) homomorphism in our sense, we require additional conditions on goal states and label costs.

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

3.1.2. Composition of Transformations

We now consider *composing* transformations.

Definition 3.4 (Composition of Transformations)**.** *Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of a transition system Θ into a transition system Θ', and let $\tau' = \langle \Theta'', \sigma', \lambda' \rangle$ be a transformation of Θ' into a transition system Θ''. The* composition *of τ' and τ is the transformation of Θ into Θ'' defined as $\tau' \circ \tau := \langle \Theta'', \sigma' \circ \sigma, \lambda' \circ \lambda \rangle$.*

It is easy to verify that the composition of two transformations is indeed a transformation. The following theorem shows that a composed transformation inherits the common properties of its component transformations.

Theorem 3.1. *Let X be any of the properties of transformations from Definition 3.3. Let τ be a transformation of transition system Θ into transition system Θ' with property X, and let τ' be a transformation of Θ' into transition system Θ'' with property X. Then the composed transformation $\tau'' = \tau' \circ \tau$ also satisfies X.*

For the properties that are restricted to a subset of states ($\mathbf{CONS}_\mathbf{T}^Q$ and $\mathbf{CONS}_\mathbf{G}^Q$), if τ satisfies the property for the subset Q of states of Θ and τ' satisfies it for the subset Q' of the states of Θ', then τ'' satisfies the property for the subset $Q'' = Q \cap \sigma^{-1}(Q')$ of states of Θ.

Proof. Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$, $\Theta' = \langle S', L', c', T', s_0', S_\star' \rangle$, and $\Theta'' = \langle S'', L'', c'', T'', s_0'', S_\star'' \rangle$. Let $\tau = \langle \Theta', \sigma, \lambda \rangle$ and $\tau' = \langle \Theta'', \sigma', \lambda' \rangle$. Then $\tau'' = \langle \Theta'', \sigma'', \lambda'' \rangle$ with $\sigma'' = \sigma' \circ \sigma$ and $\lambda'' = \lambda' \circ \lambda$.

$\mathbf{IND_S}$ The composition of surjective functions is surjective.

$\mathbf{IND_L}$ The composition of surjective functions is surjective.

$\mathbf{CONS}_\mathbf{T}^Q$ Let $Q'' = Q \cap \sigma^{-1}(Q')$. Consider $s, t \in Q''$ and $\ell \in L$ with $s \xrightarrow{\ell} t \in T$. Because τ has property $\mathbf{CONS}_\mathbf{T}^Q$ for $Q'' \subseteq Q$, we have that $\sigma(s) \xrightarrow{\lambda(\ell)} \sigma(t) \in T'$. Because τ' has property $\mathbf{CONS}_\mathbf{T}^Q$ for $\sigma(Q'') \subseteq Q'$, we have that $\sigma'(\sigma(s)) \xrightarrow{\lambda'(\lambda(\ell))} \sigma'(\sigma(t)) \in T''$, which is $\sigma''(s) \xrightarrow{\lambda''(\ell)} \sigma''(t) \in T''$. Hence we get that $\forall s, t \in Q'' \, \forall \ell \in L : s \xrightarrow{\ell} t \in T \Rightarrow \sigma''(s) \xrightarrow{\lambda''(\ell)} \sigma''(t) \in T''$, which shows that τ'' has property $\mathbf{CONS}_\mathbf{T}^Q$ for Q''.

$\mathbf{CONS_T}$ Follows from $\mathbf{CONS}_\mathbf{T}^Q$ for $Q = S$ and $Q' = S'$. We obtain $Q'' = Q \cap \sigma^{-1}(Q') = S \cap \sigma^{-1}(S') = S \cap S = S$.

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

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

$\mathbf{CONS_G^Q}$ Let $Q'' = Q \cap \sigma^{-1}(Q')$. Consider $s \in Q'' \cap S_\star$. Because τ has property $\mathbf{CONS_G^Q}$ for $Q'' \subseteq Q$, we have that $\sigma(s) \in S'_\star$. Because τ' has property $\mathbf{CONS_G^Q}$ for $\sigma(Q'') \subseteq Q'$, we have that $\sigma''(s) = \sigma'(\sigma(s)) \in S''_\star$, and hence τ'' has property $\mathbf{CONS_G^Q}$ for $Q \cap \sigma^{-1}(Q')$.

$\mathbf{CONS_G}$ Follows from $\mathbf{CONS_G^Q}$ for $Q = S$ and $Q' = S'$. We obtain $Q'' = Q \cap \sigma^{-1}(Q') = S \cap \sigma^{-1}(S') = S \cap S = S$.

$\mathbf{IND_G}$ Consider $s'' \in S''_\star$. Then, because τ' has property $\mathbf{IND_G}$, there exists $s' \in S'_\star$ with $s' \in \sigma'^{-1}(s'')$. Because $s' \in S'_\star$ and τ has property $\mathbf{IND_G}$, there exists $s \in S_\star$ with $s \in \sigma^{-1}(s')$. Hence, put together, we obtain that given $s'' \in S''_\star$, there exists $s \in S_\star$ with $s \in \sigma^{-1}(\sigma'^{-1}(s'')) = \sigma''^{-1}(s'')$, which shows that τ'' has property $\mathbf{IND_G}$.

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

$\mathbf{CONS_C}$ Consider $\ell \in L$. Because τ has property $\mathbf{CONS_C}$, $c'(\lambda(\ell)) \leq c(\ell)$. Because τ' has property $\mathbf{CONS_C}$, $c''(\lambda'(\lambda(\ell)) \leq c'(\lambda(\ell))$. Hence $c''(\lambda'') = c''(\lambda'(\lambda(\ell)) \leq c(\ell)$, which shows that τ'' has property $\mathbf{CONS_C}$.

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

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

□

3.1.3. Effect of Properties of Transformations on Heuristics

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

In the following, for a path $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \ldots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle$ in a transition system Θ, we say that π is *within* $Q \subseteq S$ if $s_i \in Q$ for all $0 \leq i \leq n$. The empty path from $s \in Q$ to s is considered to be within Q. If $\tau = \langle \Theta', \sigma, \lambda \rangle$ is a transformation of Θ, we write $\tau(\pi)$ for the *transformed path* $\langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \ldots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n) \rangle$. Note that in general $\tau(\pi)$ may include transitions that are not transitions of Θ'. We call $\tau(\pi)$ a *legal path* if it is actually a path in Θ'.

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

The first theorem establishes how conservative transformations affect the existence and cost of paths and plans in the transformed transition system.

Theorem 3.2. *Let Θ be a transition system with states S and label costs c, and let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let $Q \subseteq S$, and let π be a path within Q in Θ. Then:*

1. *If τ is transition-conservative for Q (**CONS**$_\mathbf{T}^Q$), then $\tau(\pi)$ is a legal path within $\sigma(Q)$ in Θ'.*
2. *If τ is transition- and goal-conservative for Q (**CONS**$_\mathbf{T}^Q$ + **CONS**$_\mathbf{G}^Q$) and π is an s-plan, then $\tau(\pi)$ is a $\sigma(s)$-plan within $\sigma(Q)$ for Θ'.*
3. *If τ is cost-conservative (**CONS**$_\mathbf{C}$), then $c'(\tau(\pi)) \leq c(\pi)$.*

Proof. 1. Let $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \ldots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle$ be a path within Q in Θ. We have $\tau(\pi) = \langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \ldots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n) \rangle$. Because τ is transition-conservative for Q, all elements of $\tau(\pi)$ are transitions of Θ', and hence $\tau(\pi)$ is a legal path in Θ'. Moreover, because all states of π are in Q, all states of $\tau(\pi)$ are in $\sigma(Q)$.

2. Let π be an s-plan within Q for Θ, ending in goal state $s' \in Q$. From the previous part, we get that $\tau(\pi)$ is a legal path within $\sigma(Q)$ in Θ', beginning in state $\sigma(s)$ and ending in state $\sigma(s')$. Because τ is goal-conservative for Q, $\sigma(s')$ is a goal state, and hence $\tau(\pi)$ is a $\sigma(s)$-plan within $\sigma(Q)$ for Θ'.

3. Let $\pi = \langle s_0 \xrightarrow{\ell_1} s_1, \ldots, s_{n-1} \xrightarrow{\ell_n} s_n \rangle$. Then $\tau(\pi) = \langle \sigma(s_0) \xrightarrow{\lambda(\ell_1)} \sigma(s_1), \ldots, \sigma(s_{n-1}) \xrightarrow{\lambda(\ell_n)} \sigma(s_n) \rangle$, and we get $c'(\tau(\pi)) = \sum_{i=1}^{n} c'(\lambda(\ell_i)) \leq \sum_{i=1}^{n} c(\ell_i) = c(\pi)$, where the inequality holds because τ is cost-conservative.

□

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

Theorem 3.3. *Let Θ be a transition system with reachable states R, and let τ be a transformation of Θ. The heuristic h^τ for Θ induced by τ is*

1. *forward-goal-aware if τ is goal-conservative for R (**CONS**$^R_{\mathbf{G}}$),*
2. *forward-consistent if τ is transition-conservative for R and cost-conservative (**CONS**$^R_{\mathbf{T}}$ + **CONS**$_{\mathbf{C}}$),*
3. *forward-admissible if τ is conservative for R (**CONS**^{R}),*
4. *goal-aware if τ is goal-conservative (**CONS**$_{\mathbf{G}}$),*
5. *consistent if τ is transition-conservative and cost-conservative (**CONS**$_{\mathbf{T}}$ + **CONS**$_{\mathbf{C}}$), and*
6. *admissible if τ is conservative (**CONS**).*

Proof. Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ and $\Theta' = \langle S', L', c', T', s'_0, S'_\star \rangle$. We prove the claims 1.–3. and observe that the same proofs can be used for the claims 4.–6. by replacing the set of reachable states R with the set of all states S.

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

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

3. Follows from the previous two claims because (forward-) goal-awareness and (forward-) consistency implies (forward-) admissibility.

□

In summary, conservative transformations (abstractions) give rise to admissible and consistent heuristics. If a transformation is only conservative on the set of reachable states, we obtain forward-admissibility and forward-consistency (i.e. admissibility and consistency for the set of reachable states). We remark that the theorem can be generalized to other subsets Q of states than the set of reachable states as long as we are only interested in paths or plans within Q.

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

Theorem 3.4. *Let Θ be a transition system with label costs c, and let $\tau = \langle \Theta', \sigma, \lambda \rangle$ be a transformation of Θ into transition system Θ' with label costs c'. Let π' be a path from state s' to state t' in Θ', and let $s \in \sigma^{-1}(s')$. Then:*

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

Proof. 1. The proof is by induction over the length of π', denoted by $|\pi'|$. Base case: $|\pi'| = 0$, which implies $\pi' = \langle\rangle$ and $s' = t'$. Set $t = s$. Then $\pi = \langle\rangle \in \tau^{-1}(\pi')$ is a legal path from $s \in \sigma^{-1}(s')$ to $t = s \in \sigma^{-1}(s') = \sigma^{-1}(t')$. Inductive step: assume that the property holds for $|\pi'| = n$ and consider π' with $|\pi'| = n + 1$. Let $|\pi'| = \pi'_{u'} \circ \langle u' \xrightarrow{\ell'} t' \rangle$, i.e. π' consists of a length-n path from s' to some state u', followed by a transition from u' to t' with label ℓ'.[2] By the induction hypothesis, there exists a legal path $\pi_u \in \tau^{-1}(\pi'_{u'})$ in Θ from s to some state $u \in \sigma^{-1}(u')$. Because of **REF**$_\mathbf{T}$, Θ has a transition $u \xrightarrow{\ell} t$ with $\ell \in \lambda^{-1}(\ell')$ and $t \in \sigma^{-1}(t')$. Let $\pi = \pi_u \circ \langle u \xrightarrow{\ell} t \rangle$. We observe that $\pi \in \tau^{-1}(\pi')$ and π is a path from s to $t \in \sigma^{-1}(t')$ in Θ, concluding the proof.

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

3. Let $\langle \ell'_1, \ldots, \ell'_n \rangle$ be the sequence of labels in π'. From the definition of refined paths, the sequence of labels in π is of the form $\langle \ell_1, \ldots, \ell_n \rangle$ with $\ell_i \in \lambda^{-1}(\ell'_i)$ for all $1 \leq i \leq n$. From **REF**$_\mathbf{C}$, we get $c(\ell_i) = c'(\ell'_i)$ for all $1 \leq i \leq n$. We obtain $c(\pi) = \sum_{i=1}^{n} c(\ell_i) = \sum_{i=1}^{n} c'(\ell'_i) = c'(\pi')$, concluding the proof.

□

The theorem shows that with refinable transformations, plans in the transformed transition system can be "transformed back" into plans of the original transition system, at the same cost. This result is roughly converse to Theorem 3.2, and it implies that the shortest path costs in the transformed transition system under a refinable transformation can never be lower than the corresponding shortest path costs in the original transition system. This leads to the following result.

[2] We use the symbol "$\circ$" to denote concatenation of sequences.

Theorem 3.5. *Let Θ be a transition system with reachable states R, and let τ be a transformation of Θ. The heuristic h^τ for Θ induced by τ is*

1. *lower-bounded by h^* ($h^*(s) \leq h^\tau(s)$ for all states s) if τ is refinable (**REF**),*
2. *forward-perfect if τ is exact for R (**CONS**^{R} + **REF**), and*
3. *perfect if τ is exact (**CONS** + **REF**).*

Proof. Let $\tau = \langle \Theta', \sigma, \lambda \rangle$. Part 1. follows directly from Theorem 3.4: for states s with $h^\tau(s) = \infty$, there is nothing to show. If $h^\tau(s) = k < \infty$, then Θ' has a $\sigma(s)$-plan π' of cost k. Because $s \in \sigma^{-1}(\sigma(s))$, Theorem 3.4.2 shows that Θ has an s-plan $\pi \in \tau^{-1}(\pi')$, and Theorem 3.4.3 shows that the cost of this plan is k. Therefore $h^*(s)$, the cost of an *optimal* s-plan, is at most k.

Parts 2. and 3. follow from part 1. and the forward-admissibility/admissibility results of Theorem 3.3 because $h^*(s) \leq h^\tau(s)$ (part 1.) and $h^\tau(s) \leq h^*(s)$ (admissibility) imply $h^\tau(s) = h^*(s)$. □

The main message of this theorem is that exact transformations give rise to perfect heuristics. Of course, this is no coincidence: we chose the name "exact" for conservative and refinable transformations precisely for this reason.

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

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

Bäckström and Jonsson do not include goal states in their formalization, so they study notions of admissibility and consistency for distances between arbitrary pairs of states rather than from a given state to the nearest goal state. Their main result on admissibility and consistency is that their notion of homomorphisms (transformations with property $\mathbf{M}_\uparrow\mathbf{R}_\uparrow\mathbf{C}_\uparrow$ in their notation, corresponding to our property $\mathbf{CONS_T}$ as discussed in Section 3.1.1) gives rise to admissible and consistent heuristics if we additionally require the equivalent of property $\mathbf{CONS_C}$ for their way of modeling transition costs. This result (Theorem 15 in their paper) is analogous to our Theorem 3.3.

Of course, the observation that homomorphisms induce admissible and consistent heuristics is a very well known basic property of abstractions, so at a high level, this is neither a new contribution by Bäckström and Jonsson nor is it a new contribution by us. Rather, in both works the value of the results comes from the way that these well known observations are connected

to a general model of transformations. In this sense, proving that abstractions induce admissible and consistent heuristics in both works acts as a sanity test that (in both cases) the notions of "abstractions" and "induced heuristics" were defined in a reasonable way.

Theorem 3.3 also notes that at least forward-admissibility and forward-consistency are guaranteed if the homomorphism property is only guaranteed for the set of reachable states. In the work of Bäckström and Jonsson, there are no distinguished initial and goal states and hence no concept of "reachable states". However, they do discuss and prove some results for a variant of admissibility that only applies to finite heuristic values (roughly speaking, if the heuristic value is finite, it must be admissible, but inadmissible infinite estimates are permitted), which is similar in spirit.

Bäckström and Jonsson do not discuss conditions for perfect heuristics, so their work does not contain a result equivalent to Theorem 3.5, which establishes that conservative and refinable transformations induce perfect heuristics. However, they do cover several notions of refinability, leading to results with some similarity to our Theorem 3.4.[3] Specifically, they consider properties of transformations under which every path in the transformed transition system is *downward state refinable* (in the following *refinable* for short), studying several variants of refinability.

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

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

Bäckström and Jonsson (2012) show that strong refinability implies weak refinability, which in turn implies trivial refinability, while the converse statements do not hold. One major difference between these notions and our notion of refinability employed in Theorem 3.4 is that Bäckström and Jonsson treat transition systems as unlabeled digraphs for the purposes of refinement; transition labels are ignored. Also, their notions are centered on paths, not transitions, which in particular means that a single transition may be refined into an arbitrary long path.

Both of these choices mean that their notion of refinement cannot be used for quantitative properties of heuristics like the relationship $h^\tau(s) \geq h^*(s)$ that holds for our notion of refinement. However, refinement in the style of Bäckström and Jonsson *can* in principle be used for a qualitative variation of this property based on reachability, namely that $h^\tau(s) = \infty$ whenever $h^*(s) = \infty$, i.e. *perfect dead end detection*. Of the above three notions of refinability, only strong refinability guarantees perfect dead end detection: weak refinability (and by implication trivial refinability) is too weak because of the existential quantification over $s_0 \in \sigma^{-1}(s'_0)$. Strong refinability does guarantee perfect dead end detection, but is more restrictive than necessary for

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

this purpose: it suffices to require that for every path from s' to t' in Θ' and all $s \in \sigma^{-1}(s')$, there exists a path from s to *some* $t \in \sigma^{-1}(t')$ in Θ. This condition is equivalent to a fourth (unnamed) variant of refinability in the work of Bäckström and Jonsson (2012, 2013), which they denote by $\mathbf{P}_{\downarrow}$.

We conclude our discussion of transformations with a brief summary of the main results. We have seen that conservative transformations (abstractions) give rise to admissible and consistent heuristics, while exact (conservative and refinable) transformations give rise to perfect heuristics. In the weaker case where the abstraction property only holds for the reachable states, we still obtain the "forward" versions of these heuristic properties. These relationships between transformation properties and heuristic properties, together with the earlier observation that induced abstractions are in a formal sense the "most informative" abstractions, are the reason why we consider conservativeness, inducedness and refinability desirable properties of transformations. In Sections 3.3–3.6, we will come back to these properties of transformations in the context of the merge-and-shrink framework. In particular, this will allow us to conclude under which conditions the merge-and-shrink transformations give rise to admissible or perfect heuristics.

3.2. Factored Representations

In this section, we define the factored representations used by the merge-and-shrink framework. We first define factored transition systems and explain how classical planning is an example where factored transition systems occur. Then we also describe the so-called factored mappings that can be used to represent functions defined on states of a factored transition system, such as abstraction mappings or heuristic functions. Factored transition systems and factored mappings can be used together for factored transformations, which in turn represent transformations as we have discussed them in the previous section. Finally, we sketch the general merge-and-shrink algorithm, making use of the factored representations defined before.

3.2.1. Factored Transition Systems

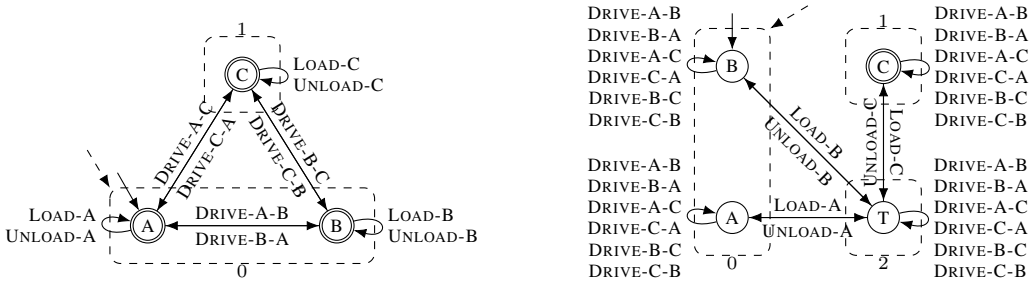
We first define factored transition systems.

Definition 3.5 (Factored Transition System)**.** *A* factored transition system *is a finite set* $F = \{\Theta^1, \ldots, \Theta^n\}$ *of transition systems where each transition system* $\Theta^i \in F$ *has the same set of labels* L *and the same label cost function* c*, i.e.* $\Theta^i = \langle S^i, L, c, T^i, s_0^i, S_\star^i \rangle$ *for all* $1 \leq i \leq n$*.*

Informally speaking, a factored transition system consists of transition systems, also called *factors*, sharing the same labels with the same cost. Figure 3.2 shows an example factored transition system with two factors. We will give some intuition for the two factors in the next section where we describe how planning tasks induce factored transition systems.

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

Definition 3.6 (Product System)**.** *Let* $F = \{\Theta^1, \ldots, \Theta^n\}$ *be a factored transition system with* $\Theta^i = \langle S^i, L, c, T^i, s_0^i, S_\star^i \rangle$ *for all* $1 \leq i \leq n$*. The* product system *(also called* synchronized product*) of* F *is defined as* $\bigotimes F := \langle S^\otimes, L, c, T^\otimes, s_0^\otimes, S_\star^\otimes \rangle$*, where* $S^\otimes = \prod_{i=1}^n S^i$*,* $T^\otimes =$



(a) Atomic factor for v_T.

(b) Atomic factor for v_P.

Figure 3.2.: Example of a factored transition system. Also: induced factored transition system of the example planning task of Figure 2.1.

$\{\langle s^1, \ldots, s^n\rangle \xrightarrow{\ell} \langle t^1, \ldots, t^n\rangle \mid s^i \xrightarrow{\ell} t^i \in T^i \text{ for all } 1 \leq i \leq n\}$, $s_0^{\otimes} = \langle s_0^1, \ldots, s_0^n\rangle$, *and* $S_\star^{\otimes} = \prod_{i=1}^n S_\star^i$.[4]

In words, the product system is the transition system which is implicitly represented through the synchronized behavior of the factored transition system. States in the product system correspond to combinations of states of its factors. Labels are used to synchronize the factors of the factored transition system via the labeled transitions: there is a transition between two states in the product system iff all factors have a transition between the corresponding component states labeled with the same label. A state is an initial/goal state in the product if all its components are initial/goal states in the respective factors.

Let $F = \{\Theta^1, \ldots, \Theta^n\}$ be a factored transition system. Similarly to the notations for transition systems, we write $\langle s^1, \ldots, s^n\rangle \in \bigotimes F$ to denote states of $\bigotimes F$, and also $\langle s^1, \ldots, s^n\rangle \xrightarrow{\ell} \langle t^1, \ldots, t^n\rangle \in \bigotimes F$ to denote transitions of $\bigotimes F$. Sometimes, we also interpret states of $\bigotimes F$ as assignments of the factors to their states, i.e. we may write a state s of $\bigotimes F$ as $\{\Theta \mapsto t \mid \Theta \in F, t \in S \text{ with } S \text{ states of } \Theta\}$.[5]

Consider the factored transition system with the two factors shown in Figure 3.2. Ignore the dashed boxes for the moment. Figure 3.3 shows the product system of this factored transition system (note that the transition system is identical to the induced transition system of the example planning task shown in Figure 2.2). However, the "names" of states are not formally correct: for example, state AB should formally be written as $\langle \text{A}, \text{B}\rangle$ to properly refer to the product state of the product system. In the following, we will consider transition systems as equivalent if they only differ in the names of states. With this convention, the product operation is associative and commutative. To exemplify the definition of the product, we see that in the first factor there is a transition A $\xrightarrow{\text{DRIVE-A-B}}$ B, and in the second factor there is a self-looping transition B $\xrightarrow{\text{DRIVE-A-B}}$ B, and hence in the product there must be a transition AB $\xrightarrow{\text{DRIVE-A-B}}$ BB. Figure 3.3 indeed shows such a transition. The goal states of the product system are all those where the second component reads C because only state C is a goal state in the second factor, and all

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

[5] The notation of states of product systems is not quite clean because we define F as a set and not a tuple, and thus factors of F are not indexed. However, we think that it is clear what we mean and that the notation is sometimes more readable than that using sets of assignments of factors (which will be useful when we reason about FMs).

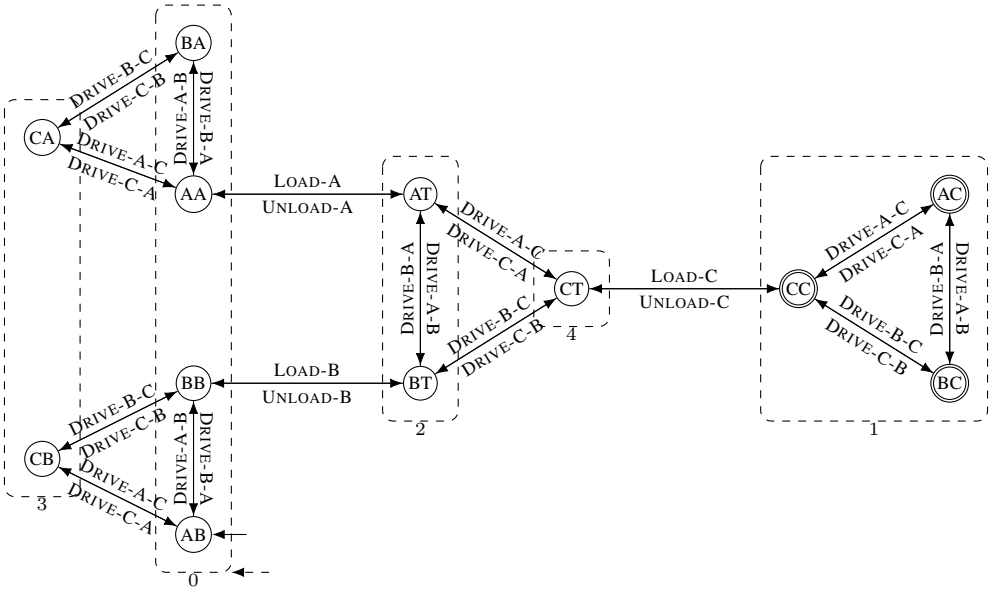


Figure 3.3.: Product system of the factored transition system shown in Figure 3.2.

states are goal states in the first factor.

3.2.2. Factored Representations of Planning Tasks

While the concepts we present in this chapter apply to arbitrary factored transition systems, we illustrate them through their application to classical planning. We already defined planning tasks in Section 2.1, cf. Definition 2.4, and showed that they induce transition systems which define their semantics.

Planning tasks also induce a factored representation of the induced transition system in a natural way. Before defining the induced factored transition system of a planning task, we first need to define atomic factors of a planning task.

Definition 3.7 (Atomic Factor)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ *be a planning task. The* atomic factor *for variable* $v \in \mathcal{V}$ *is the transition system* $\Theta(v) = \langle dom(v), \mathcal{O}, c, T, s_0[v], S_\star \rangle$ *where* c *is the cost function mapping each label* $\ell \in \mathcal{O}$ *to the cost* $cost(\ell)$ *of the operator,* $T = \{d \xrightarrow{\ell} d' \mid (v \notin vars(pre(\ell)) \vee pre(\ell)[v] = d) \wedge ((v \notin vars(\mathit{eff}(\ell)) \wedge d = d') \vee \mathit{eff}(\ell)[v] = d')\}$*, and* $S_\star = \{s_\star[v]\}$ *if* $v \in vars(s_\star)$ *and* $S_\star = dom(v)$ *otherwise.*

Informally speaking, an atomic factor represents the behavior of a single state variable of a planning task, with states of the factor corresponding to possible values of its variable. As such, atomic factors are closely related to the concept of *domain transition graphs* (Jonsson & Bäckström, 1998), with the difference that atomic factors represent *all* operators of the planning task, including self-looping transitions for operators that do not change the value of the variable.

Definition 3.8 (Induced Factored Transition System)**.** *The* induced factored transition system *of a planning task* Π *with variables* $\mathcal{V}$ *is the set* $F(\Pi) := \{\Theta(v) \mid v \in \mathcal{V}\}$.

The factored transition system shown in Figure 3.2 above is the induced factored transition system of the example planning task of Figure 2.1. The transition system shown in Figure 3.2a is the atomic factor for variable v_T, and the one in Figure 3.2b is the atomic factor for variable v_P.

An important observation is that the product system of the induced factored transition system of a planning task is isomorphic (modulo names of states) to the induced transition system of the planning task. We already have seen this with our example planning task: the product of the induced factored transition system shown in Figure 3.2 corresponds to the induced transition system shown in Figure 2.2 (which we included again in this section in Figure 3.3).

3.2.3. Factored Mappings

In addition to compactly describing transition systems, we also need a way to compactly describe *mappings* (functions) defined on the states of a transition system. For example, we might need to represent an abstraction mapping that relates the states of a large ("concrete") transition system to the states of a smaller ("abstract") transition system, or we might want to represent a heuristic function.

In the merge-and-shrink framework, such mappings are represented with a tree-like data structure we call *factored mappings*. They can be used for all functions which are defined over assignments to a set of finite-domain variables, such as the state variables of a planning task. Factored mappings have previously also been called "cascading tables" (Helmert et al., 2014; Torralba, 2015) and "merge-and-shrink representations" (Helmert, Röger, & Sievers, 2015). Below, we repeat the definition by Helmert et al. (2015) adapted to our notations.

Definition 3.9 (Factored Mapping)**.** Factored mappings *(*FM*s) over a set of finite-domain variables $\mathcal{V}$ are inductively defined as follows. An FM $\mathcal{M}$ has an associated finite* value set $vals(\mathcal{M}) \neq \emptyset$ *and an associated* table $\mathcal{M}^{\text{tab}}$*. $\mathcal{M}$ is either* atomic *or a* merge*.*

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

In words, FMs can be viewed as binary trees with merges as inner nodes and atomic FMs as leaves. Leaves have an associated variable and define mappings from values of this variable to some set of values. Inner nodes determine how the mappings of their children should be combined. If $\mathcal{M}$ is an FM defined over variables $\mathcal{V}$, we write $vars(\mathcal{M}) \subseteq \mathcal{V}$ for the associated variables of all the (atomic) leaf components of $\mathcal{M}$.

Definition 3.10 (Represented Function)**.** *Let $\mathcal{M}$ be an FM over a set of finite-domain variables $\mathcal{V}$. In the following, we use the symbol $\mathcal{M}$ to denote both the FM and the function it represents.* $\mathcal{M}$ represents *a function $\mathcal{M}$ mapping assignments α of $vars(\mathcal{M})$ to $vals(\mathcal{M})$, which is inductively defined as follows:*

- *If $\mathcal{M}$ is atomic with associated variable v, then $\mathcal{M}(\alpha) := \mathcal{M}^{\text{tab}}(\alpha(v))$.*

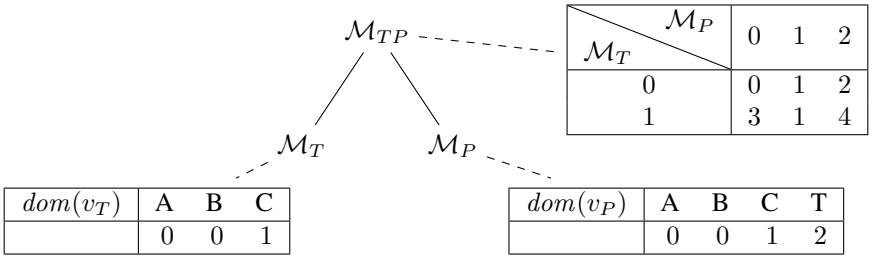


Figure 3.4.: An FM representing an abstraction mapping of the induced transition system of the example planning task of Figure 2.1.

- *If $\mathcal{M}$ is a merge, then* $\mathcal{M}(\alpha) := \mathcal{M}^{\text{tab}}(\mathcal{M}_L(\alpha), \mathcal{M}_R(\alpha))$.

Informally speaking, FMs $\mathcal{M}$ over finite-domain variables $\mathcal{V}$ represent functions defined on assignments over the subset $vars(\mathcal{M})$ of variables relevant to the FM. To compute the function value of a merge FM, the FM first recursively computes the values computed by its component FMs, and then uses these values to index its associated 2-dimensional table function to look up the result. The recursion ends at atomic FMs, which look up the values stored for the assignment to their associated variable in their associated 1-dimensional table.

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

Consider the (orthogonal) example FM shown in Figure 3.4. Each node is labeled with the FM it corresponds to and connected to its table via a dotted line. The visualization does not show the value sets, which can be inferred from the tables if we assume that they do not contain unused values. In the example, $vals(\mathcal{M}_T) = \{0, 1\}$, $vals(\mathcal{M}_P) = \{0, 1, 2\}$, and $vals(\mathcal{M}_{TP}) = \{0, 1, 2, 3, 4\}$.

To illustrate the represented function of our example, consider the variable assignment $\alpha = \{v_T \mapsto \text{A}, v_P \mapsto \text{T}\}$. We can compute $\mathcal{M}_{TP}(\alpha)$ as follows:

$$
\begin{aligned}
& \mathcal{M}_{TP}(\alpha) \\
\overset{1}{=}\ & \mathcal{M}^{\text{tab}}_{TP}(\mathcal{M}_T(\alpha), \mathcal{M}_P(\alpha)) \\
\overset{2}{=}\ & \mathcal{M}^{\text{tab}}_{TP}(\mathcal{M}^{\text{tab}}_T(\alpha(v_T)), \mathcal{M}^{\text{tab}}_P(\alpha(v_P))) \\
\overset{3}{=}\ & \mathcal{M}^{\text{tab}}_{TP}(\mathcal{M}^{\text{tab}}_T(\text{A}), \mathcal{M}^{\text{tab}}_P(\text{T})) \\
\overset{4}{=}\ & \mathcal{M}^{\text{tab}}_{TP}(0, 2) \\
\overset{5}{=}\ & 2
\end{aligned}
$$

In the first step, the variable assignment is passed down to the components of $\mathcal{M}_{TP}$. In the

second and third step, the variable assignment is projected onto the variables of the atomic FMs $\mathcal{M}_T$ and $\mathcal{M}_P$. The recursion ends at the atomic FMs and they return the values stored in their tables according to the given variable assignment in step 4. Finally, in the fifth step, these values are used to index the table of $\mathcal{M}_{TP}$, which gives 2 as the final result.

The FM $\mathcal{M}_{TP}$ shown in Figure 3.4 represents an *abstraction* of the example planning task of Figure 2.1 on page 14. The dashed boxes in Figures 3.3 and 3.2 on pages 37 and 36 aggregate states that the example FMs map to the same abstract state. (Recall that we have already seen this abstraction in the background section: Figure 2.3 shows the abstract induced abstract transition system after applying the abstraction to the transition system of Figure 3.3.) For example, $\mathcal{M}_T(\text{A}) = \mathcal{M}_T(\text{B}) = 0$, and hence states A and B are included in a dashed box annotated with the number 0 in Figure 3.2a. To continue the example of the computation above, the merge FM $\mathcal{M}_{TP}$ maps the value 0 computed by $\mathcal{M}_T$ and the value 2 computed by $\mathcal{M}_P$ to the value 2, which means that all states where the truck is at A or B (specified through $\mathcal{M}_T$) and the package is in the truck (specified through $\mathcal{M}_P$) are aggregated, as indicated by the dashed box around the states AT and AB in Figure 3.3. The dashed box is annotated with the number 2 to show that $\mathcal{M}_{TP}(\text{AT}) = \mathcal{M}_{TP}(\text{AB}) = 2$.

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

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

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

The advantage of FMs becomes apparent when we do not use bijective table functions, which makes it possible to apply abstractions "along the way" and hence obtain potentially much smaller representations for the same abstraction heuristic compared to a PDB. For example, a PDB-style abstraction heuristic that uses information about all state variables in a complex planning task is practically infeasible because it requires computing and storing heuristic values for all states of the task. In contrast, merge-and-shrink heuristics based on FMs commonly use information about all state variables. It is not difficult to prove that FMs are more general than

PDBs in the sense that there exist functions that FMs can compactly represent but PDBs cannot (e.g. Nissim et al., 2011; Helmert et al., 2014). For a full discussion of the general representational power and the required size of FMs, we refer to Chapter 4.

3.2.4. Factored Transformations

Analogously to how factored transition systems represent their product systems compactly and how FMs represent mappings compactly, we can also use *factored transformations* to represent transformations between factored transition systems compactly. A factored transformation transforms a factored transition system and provides the state mapping through a set of FMs. Hence factored transformations exclusively work on the factored representation of transition systems, without the need to explicitly represent the potentially prohibitively large product system.

Definition 3.11 (FM Defined on a Factored Transition System)**.** *An FM* defined on a factored transition system F *is an FM whose variables are the factors of* F*, and the domain of each variable* $\Theta \in F$ *is the set of states of* Θ*.*

The most simple FM defined on F is the *projection FM*, written π_Θ, whose table projects a given state of F, i.e. a set of assignments of the factors of F to their states, to its associated variable Θ. Formally: π_Θ is an atomic FM defined on F with variable Θ and a table function that, for any state $s = \{\Theta \mapsto s[\Theta] \mid \Theta \in F\}$ of F, is defined as $\pi^{\text{tab}}_\Theta(s) = s[\Theta]$. Projection FMs are useful to define identity mappings on factors of a factored transition system: their tables represent the identity function from states of Θ to states of Θ. We use projection FMs for those parts of factored transformations that do not change the factors of a factored transition system.

We further say that an FM σ is *associated with a transition system* Θ, written σ_Θ, if it maps to the states S of Θ, i.e. $vals(\sigma_\Theta) = S$. Since $vals(\pi_\Theta) = S$ for the states S of Θ, the projection FM π_Θ is associated with Θ.

In the following, we write Σ for sets of FMs and σ (instead of $\mathcal{M}$ as before) for single FMs of such a set Σ of FMs for a clear distinction (both σ and Σ represent *state* mappings). Using FMs defined on a factored transition system that are associated with the factors of the transformed factored transition system, we can represent the state mappings of factored transformations of F, as the following definition states.

Definition 3.12 (Factored Transformation)**.** *A* factored transformation *of a factored transition system* F *with label set* L *into a factored transition system* F' *with label set* L' *is a tuple* $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$*, where*

- F' *is called the* transformed factored transition system*,*
- Σ *is an indexed collection* $(\sigma_{\Theta'})_{\Theta' \in F'}$ *of FMs*[6] *called the* state mapping*, where each FM* $\sigma_{\Theta'}$ *is defined on* F *and associated with* Θ' *(i.e.maps to the states of* Θ'*), and*
- $\lambda : L \to L'$ *is called the* label mapping*.*

The transformation induced by τ_{F} *is a transformation of* $\bigotimes F$ *into* $\bigotimes F'$ *which is defined as* $\tau = \langle \bigotimes F', \sigma, \lambda \rangle$*, where* $\sigma(s) = \{\Theta' \mapsto \sigma_{\Theta'}(s) \mid \Theta' \in F'\}$ *for all states* $s \in \bigotimes F$*.*

[6] That is, it is a collection of FMs, consisting of one FM $\sigma_{\Theta'}$ for each $\Theta' \in F'$.

Factored transformations generalize (non-factored) transformations in a natural way: they consist of the same components (a transformed transition system, a state mapping, and a label mapping), but use factored representations for transition systems and state mappings. In the following, we may refer to factored transformations as transformations where this does not lead to ambiguities.

In a factored transformation of F into F' with $|F| = n$ and $|F'| = m$, the state mapping Σ can be understood as a function with n inputs and m outputs. The m outputs are represented by having a different FM associated with each of the m factors of F', and the n inputs correspond to the n (input) variables of each FM (all FMs are defined on F and hence have the factors of F as associated variables). Such n-to-m functions compose naturally, and this composition can also be implemented directly based on the FM representations: if Σ' is another state mapping for a factored transformation of F' into a factored transition system F'' with $|F''| = k$, then $\Sigma' \circ \Sigma$ can be represented as a collection of FMs where each element $\sigma''_{\Theta''}$ can be constructed from the tree representation of $\sigma'_{\Theta''}$ by replacing each leaf associated with factor Θ' with the tree representation of $\sigma_{\Theta'}$. Equipped with this observation, it is easy to formalize the composition of factored transformations.

Proposition 3.1. *Let $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ be a factored transformation of a factored transition system F into a factored transition system F' with $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$, and let $\tau'_{\mathrm{F}} = \langle F'', \Sigma', \lambda' \rangle$ be a factored transformation of F' into a factored transition system F'' with $\Sigma' = (\sigma'_{\Theta''})_{\Theta'' \in F''}$. Then $\tau''_{\mathrm{F}} = \langle F'', \Sigma'', \lambda'' \rangle$ is a factored transformation of F into F'', where $\lambda'' = \lambda' \circ \lambda$ and $\Sigma'' = \Sigma' \circ \Sigma$ with $\Sigma''(s) = (\sigma'_{\Theta''}((\sigma_{\Theta'})_{\Theta' \in F'}(s)))_{\Theta'' \in F''}$ for all states $s \in F$. We also write $\tau'_{\mathrm{F}} \circ \tau_{\mathrm{F}}$ for τ''_{F}.*

Algorithm 1 shows pseudocode for the construction of the set $\Sigma'' = \Sigma' \circ \Sigma$ of FMs of a composed factored transformation, given the two sets Σ, Σ' of FMs used for the two component factored transformations. The outer function, ComposeSetsOfFMs, iterates over the elements of Σ', i.e. the FMs of the second factored transformation (line 3). As stated above, starting from the representation of each such $\sigma'_{\Theta''}$ mapping to one factor Θ'' of the transformed factored transition system F'', the algorithm builds the composed FM $\sigma''_{\Theta''}$ associated with the factor Θ'' and stores it at $\Sigma''[\Theta'']$ (line 4). Constructing $\sigma''_{\Theta''}$ is done in the function ComposeElement in line 8 as follows. If $\sigma'_{\Theta''}$ is a merge, the algorithm recursively "follows the tree" underlying $\sigma'_{\Theta''}$ by computing a merge FM consisting of the same components as those of $\sigma'_{\Theta''}$, however composed with Σ, and the same table function as $\sigma'_{\Theta''}$ (line 9). Otherwise, if $\sigma'_{\Theta''}$ is atomic, the recursion ends and instead of also copying the leaves of $\sigma'_{\Theta''}$, the algorithm uses the relevant outputs of the FMs of Σ: in line 12, it copies the FM $\Sigma[\mathrm{var}(\sigma'_{\Theta''})]$ of Σ associated with the variable of $\sigma'_{\Theta''}$, i.e. $\Sigma[\mathrm{var}(\sigma'_{\Theta''})]$ is the FM in Σ that computes a result relevant to the input of $\sigma'_{\Theta''}$, and adapts the table of that copy by applying the table function of $\sigma'_{\Theta''}$ to it (line 13), i.e. the resulting FM σ_{new} is defined on F and maps to states of Θ'' as desired.

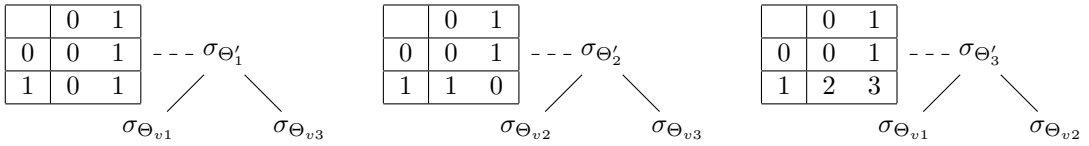
We illustrate the composition just described with the example shown in Figure 3.5. There are three variables v_1, v_2, and v_3, each with domain $\{0, 1\}$, and hence the original factored transition system F consists of three factors Θ_{v1}, Θ_{v2}, Θ_{v3}. The transformed factored transition system of the first transformation, F', consists of three factors Θ'_1, Θ'_2, and Θ'_3. As we will see, Θ'_1 and Θ'_2 are the products of two atomic factors, however further abstracted, and Θ'_3 is a (unmodified) merge of two atomic factors. F'', the transformed factored transition system of the second

Algorithm 1 Composition of sets of FMs in a composed factored transformation.

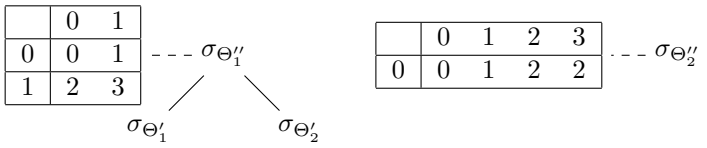
Input: Set Σ of FMs defined on F and associated with the factors of F', and set Σ' of FMs defined on F' and associated with the factors of F''.

Output: Set $\Sigma'' = \Sigma' \circ \Sigma$ of FMs defined on F and associated with the factors of F''.

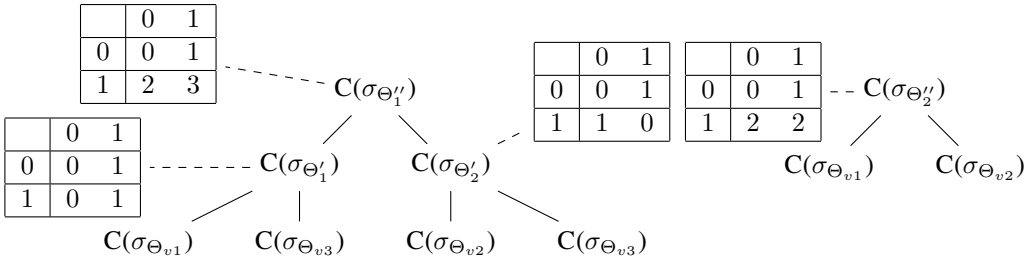
function COMPOSESETSOFFMS(Σ, Σ')
  $\Sigma'' = \emptyset$
  for $\sigma'_{\Theta''} \in \Sigma'$ **do**
    ▷ *Build the FM $\Sigma''[\Theta'']$ (also written $\sigma''_{\Theta''}$) by composing $\sigma'_{\Theta''}$ (associated with $\Theta'' \in F''$) with Σ.*
    $\Sigma''[\Theta''] \leftarrow$ COMPOSEELEMENT($\Sigma, \sigma'_{\Theta''}$)
  end for
  return Σ''
end function
function COMPOSEELEMENT($\Sigma, \sigma'_{\Theta''}$)
  if $\sigma'_{\Theta''}$ is a merge with components σ_L and σ_R **then**
    ▷ *Build the merge FM from the given components and table.*
    return MERGEFM(COMPOSEELEM.(Σ, σ_L), COMPOSEELEM.(Σ, σ_R), $\sigma'^{\text{tab}}_{\Theta''}$)
  else
    ▷ *var($\sigma'_{\Theta''}$) is the associated variable of the atomic FM $\sigma'_{\Theta''}$ and hence a factor of F.*
    $\sigma_{\text{new}} \leftarrow$ COPYOF(Σ[var($\sigma'_{\Theta''}$)])
    ▷ *Apply $\sigma'^{\text{tab}}_{\Theta''}$ to $\sigma^{\text{tab}}_{new}$.*
    $\sigma^{\text{tab}}_{\text{new}} \leftarrow$ COMPOSETABLE($\sigma'^{\text{tab}}_{\Theta''}, \sigma^{\text{tab}}_{\text{new}}$)
    return σ_{new}
  end if
end function



(a) Σ: set of FMs of the first factored transformation.



(b) Σ': set of FMs of the second factored transformation.



(c) $\Sigma' \circ \Sigma$: set of FMs of the composed factored transformation.

Figure 3.5.: Illustration of the composition of sets of FMs as used in factored transformations.

transformation, consists of two factors Θ''_1 and Θ''_2. As we will see, Θ''_1 is the (unmodified) product of Θ'_1 and Θ'_2 and Θ''_2 is an abstraction of Θ'_3. Before explaining how the FMs of these transformations can be composed using Algorithm 1, we discuss the two transformations and their FMs in detail.

In part (a) of the figure, we see three FMs forming the set Σ, representing the state mapping of the first factored transformation, i.e. the mapping from states from F to states of F'. We omit tables of leaves that represent the identity function on the variables of the leaves. The three FMs $\sigma_{\Theta'_1}$, $\sigma_{\Theta'_2}$, and $\sigma_{\Theta'_3}$ are defined on F and associated with the factors Θ'_1, Θ'_2, and Θ'_3 of F'. They are all merge FMs with two (different) atomic component FMs, where the first two FMs ($\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$) represent non-identity mappings on the product of their component factors, and the third FM ($\sigma_{\Theta'_3}$) represents the identity mapping on the product of its component factors. Since the variables of the FMs are not disjoint, this is an example of a non-orthogonal set of FMs. (For this reason, $\sigma_{\Theta_{v1}}$, $\sigma_{\Theta_{v2}}$, and $\sigma_{\Theta_{v3}}$ are not unique but rather should be understood as different copies of the same FMs.)

The FMs of the second transformation, defined on F' and representing the state mapping from F' to F'', are shown in part (b), again omitting tables of leaves that represent the identity function on the variables of the leaves: $\sigma_{\Theta''_1}$ is associated with factor Θ''_1 and is a merge with components $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ and hence variables Θ'_1 and Θ'_2, representing the identity mapping of on the product of its two component factors. $\sigma_{\Theta''_2}$ is associated with factor Θ''_2 and is atomic with variable Θ'_3, representing an abstraction of its factor. Note that since we consider the second

transformation in isolation here, $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ' are not the same FMs as those of the same name in Σ: the two FMs are merges in Σ, mapping from states of F to Θ'_1 and Θ'_2, whereas the two FMs are atomic in Σ', representing identity mappings on the states of Θ'_1 and Θ'_2.

We now describe how to compute the composition. Part (c) shows the set Σ'' of composed FMs, using C(...) to show that the FMs are copies of the FMs in Σ and Σ'. To compute $\sigma_{\Theta''_1}$ of Σ'', Algorithm 1 starts with the representation of $\sigma_{\Theta''_1}$ of Σ' by calling COMPOSEELEMENT(Σ, $\sigma_{\Theta''_1}$). Since $\sigma_{\Theta''_1}$ of Σ' is a merge, the algorithm returns in line 9, computing the result as the merge of the components of $\sigma_{\Theta''_1}$ of Σ', however recursively composed with Σ. Both recursive calls end recursion in the next layer, where the FMs $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ (the FMs in Σ associated with the associated variables Θ'_1 and Θ'_2 of $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ') are used by copying them and applying the table function of $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ' to the copies. Hence the resulting FM $\sigma_{\Theta''_1}$ of Σ'' has the same root table as $\sigma_{\Theta''_1}$ of Σ', however its components (since they are atomic and hence leaves) have been replaced with copies of $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ. In this example, $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ'' have the same tables as $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ because $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ of Σ' have tables representing the identity function on their variables.

To compute $\sigma_{\Theta''_2}$ of Σ'', the algorithm calls COMPOSEELEMENT($\Sigma, \sigma_{\Theta''_2}$) with $\sigma_{\Theta''_2}$ of Σ'. Since $\sigma_{\Theta''_2}$ of Σ' is atomic, the algorithm copies the FM $\sigma_{\Theta'_3}$ of Σ (which is associated with Θ'_3, the associated variable of $\sigma_{\Theta''_2}$ of Σ'), and applies the table of $\sigma_{\Theta''_2}$ of Σ' to that copy. The resulting FM $\sigma_{\Theta''_2}$ of Σ'' hence is a modified copy of $\sigma_{\Theta'_3}$ of Σ: the table of $\sigma_{\Theta''_2}$ of Σ' has been applied to the table of $\sigma_{\Theta'_3}$ of Σ, which now maps $\langle 1, 1\rangle$ to 2 rather than to 3 as in the first transformation.

To conclude the example, in both cases, the resulting FMs are the trees that result from replacing the leaves of the FMs of the second transformation with the correct FMs of the first transformation, namely those that map to states of the factor for which an associated FM should be composed, but with the tables of the copies modified through applying the table function of the FM of the second transformation.

To summarize the concept of factored and non-factored transformations, Figure 3.6 illustrates the relationship between non-factored and factored transformations, and the compositions of both. It shows factored transformations and their compositions on the left and the induced transformations and their compositions on the right. We conclude that for the final result, it does not matter whether we work on a factored representation or a non-factored representation of a transition system; in the end, we can always obtain a transformation of the originally given transition system (or the product of the originally given factored transition system) into the transformed transition system.

3.2.5. Merge-and-Shrink Algorithm

Armed with factored representations of transition systems, (state) mappings, and transformations, we are now ready to present the merge-and-shrink algorithm in its general form.

Given a factored transition system F, such as a factored transition system induced by a planning task, the goal of the merge-and-shrink algorithm is to compute a heuristic for the product system $\bigotimes F$ of F. It does so by computing an induced abstraction (for the reachable states) of $\bigotimes F$, which induces a heuristic for $\bigotimes F$. This means the algorithm has to compute both the transition system induced by the abstraction and the abstraction function itself, i.e. a state

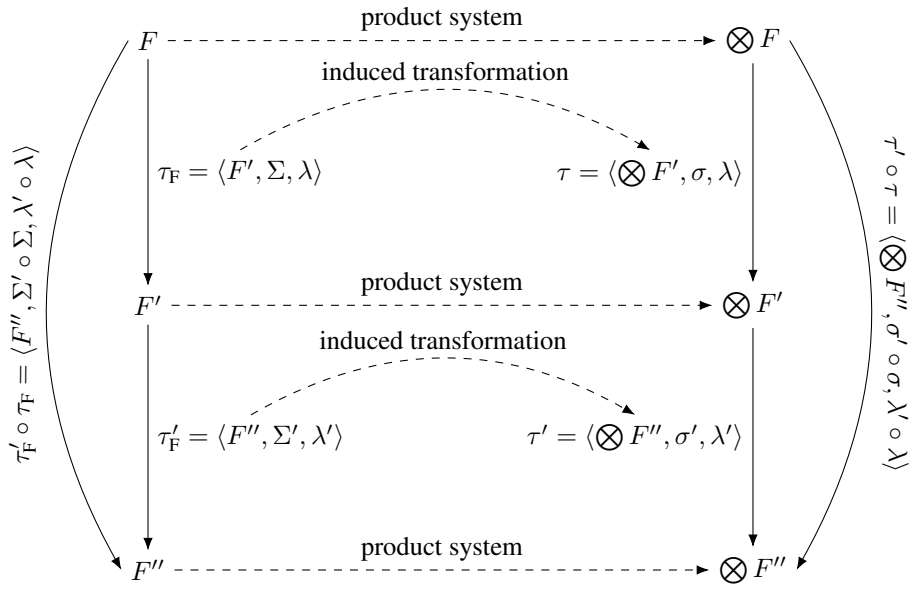


Figure 3.6.: Illustration of the composition of transformations, factored transformations, and the relationship between factored and non-factored (compositions of) transformations.

mapping from the given to the abstract transition system. Since transition systems of practical interest are usually too large to be explicitly represented, the merge-and-shrink framework directly operates on the factored representation of all involved components. In particular, it represents the input transition system as a factored transition system and then repeatedly modifies this factored transition system through a sequence of factored transformations until only one factor is left. To represent the state mapping, the algorithm also maintains a set of FMs that jointly represent the mapping from the original factored transition system to the current one.

Algorithm 2 shows pseudocode of the merge-and-shrink framework. During the initialization, the algorithm first copies the input factored transition system F' for further modifications (line 2). It then computes the set Σ of projection FMs $\pi_{\Theta'}$ for each factor Θ' of F' (line 3), which thus represent the identity state mapping on F'. As a last step of the initialization, the algorithm also computes the identity mapping λ on the labels L of F' (line 4). Since $F' = F$ after the initialization, $\langle F', \Sigma, \lambda \rangle$ is a (trivial or identity) factored transformation of F into F' at this point. This is also the invariant of the algorithm: after applying a transformation, it is guaranteed that $\langle F', \Sigma, \lambda \rangle$ is a factored transformation of F into the current F'.

The algorithm continues with the main loop, where in each iteration, it selects a transformation $\langle F'', \Sigma', \lambda' \rangle$ of the current factored transition system F' in line 6. It "applies the transformation to F'", which means to compose it with the current factored transformation $\langle F', \Sigma, \lambda \rangle$ (lines 7–9), hence ensuring that the invariant is restored. If there is only one factor Θ' left in F', and hence only one FM $\sigma_{\Theta'}$ left in Σ that is defined on F and associated with Θ', the algorithm returns these two elements. As we will see in the following sections, the merge-and-shrink transformations all satisfy properties such that the final transition system Θ' is an abstraction of $\bigotimes F$, and $\sigma_{\Theta'}$ is the abstraction function, mapping from states of $\bigotimes F$ to states of Θ'. The

Algorithm 2 Merge-and-shrink algorithm operating on factored representations.

Input: Factored transition system F.
Output: Transition system Θ and FM σ mapping from states of $\bigotimes F$ to states of Θ.

1: **function** MERGEANDSHRINK(F)
    ▷ *Copy the input factored transition system.*
2: $F' \leftarrow F$
    ▷ *For each factor Θ' of F', compute the projection FM $\pi_{\Theta'}$ that projects states of F' to the component states of Θ. Σ represents the identity mapping on F'.*
3: $\Sigma \leftarrow \{\pi_{\Theta'} \mid \Theta' \in F'\}$
    ▷ *Initialize λ as the identity function on labels L of F'.*
4: $\lambda \leftarrow \mathrm{id}_L$
5: **while** $|F'| > 1$ **do**
6: $\langle F'', \Sigma', \lambda' \rangle \leftarrow$ SELECTTRANSFORMATION(F', Σ, λ)
        ▷ *Update the current transformation $\langle F', \Sigma, \lambda \rangle$ to be the composition of the current with the selected transformation $\langle F'', \Sigma', \lambda' \rangle$.*
7: $F' \leftarrow F''$
8: $\Sigma \leftarrow \Sigma' \circ \Sigma$
9: $\lambda \leftarrow \lambda' \circ \lambda$
10: **end while**
11: $\Theta' \leftarrow$ the single element of F'
12: $\sigma_{\Theta'} \leftarrow$ the single element of Σ
13: **return** $\langle \Theta', \sigma_{\Theta'} \rangle$
14: **end function**

merge-and-shrink heuristic for F is the heuristic for $h^{\sigma_{\Theta'}}_{\bigotimes F}$ induced by $\bigotimes F$ and $\sigma_{\Theta'}$.

There are four types of transformations that the algorithm can choose from in SELECTTRANSFORMATION which we describe in more detail in the following sections: *merge* transformations combine two factors of F' into one, reducing the size of F' by 1; *shrink* transformations apply abstractions to one of the factors of F'; *prune* transformations "discard" (a subset of the) dead states; and *label reductions* map the common label set of F' to a smaller set. Each of the transformation types also represents a parameter of the merge-and-shrink framework: a *merge strategy* specifies which two factors to merge, and similarly, a *shrink strategy*, a *prune strategy* and a *label reduction strategy* determine how exactly to shrink, prune and reduce labels whenever such a transformation shall be applied. We will collectively refer to these as *transformation strategies*.

Furthermore, a concrete instantiation of the algorithm also has to specify which transformation to choose in each iteration, and it may provide further global parameters that influence the behavior of the transformation strategies, such as a global limit on the number of permitted abstract states. In Section 3.7, we describe a specific instantiation of the merge-and-shrink algorithm in detail and also provide more details on transformation strategies and other parameters of the algorithm.

Before proceeding with the merge-and-shrink transformations in the following sections, we briefly comment on two key aspects of such transformations that we need to consider: firstly, we need to define how each factored transformation works, i.e. what kind of state and label mappings it applies and how the transformed factored transition system is defined. Secondly, we must be able to compose the factored transformation with any previous transformation. To do so efficiently, we need to build upon the state and label mapping of the previous factored transformations. While this is simple for label mappings, which are (non-factored) functions that can just be composed, this can be more involved for composing sets of FMs, as we have discussed before. In particular, for an efficient computation of the merge-and-shrink algorithm, it will be important to not copy all of the FMs each time we compose two sets of FMs that are part of two factored transformations that should be composed.

3.3. Shrink Transformation

In this section, we define the shrink transformation of the merge-and-shrink framework. For a transition system $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ and a state mapping α defined on S, the *transition system induced by* Θ *and* α is defined as $\Theta^\alpha = \langle \alpha(S), L, c, \{\langle \alpha(s), \ell, \alpha(t) \rangle \mid \langle s, \ell, t \rangle \in T\}, \alpha(s_0), \alpha(S_\star) \}\rangle$. (These definitions match our definition of abstraction mappings and induced abstract transition systems as we used them to define abstractions in the background section on page 17.)

Definition 3.13 (Shrink Transformation)**.** *Let F be a factored transition system, and let $\Theta \in F$. Let α be a state mapping for Θ. A* shrink transformation *is a factored transformation $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ of F into a factored transition system F', where:*

- $F' = (F \setminus \{\Theta\}) \cup \{\Theta^\alpha\}$.

- $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ *is an indexed collection of FMs, where each FM* $\sigma_{\Theta'}$ *is defined on* F *and associated with* $\Theta' \in F'$*:* $\sigma_{\Theta'} := \pi_{\Theta'}$ *for all* $\Theta' \in F'$ *with* $\Theta' \neq \Theta^\alpha$*, and* σ_{Θ^α} *is an atomic FM with variable* Θ *and* $\sigma^{\text{tab}}_{\Theta^\alpha}(s) := \alpha(s[\Theta])$*.*
- $\lambda = \text{id}$ *is the identity label mapping.*

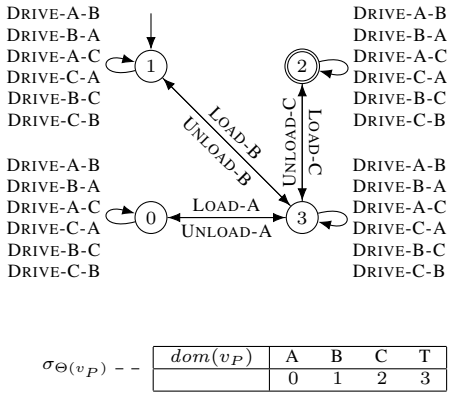
In words, a shrink transformation applies a state mapping α to a single factor Θ of a given factored transition system F, not changing the labels at all. The transformed factored transition system F' is identical to the original one except for the single factor Θ to which the state mapping is applied, which is replaced by Θ^α, i.e. the transition system induced by Θ and α. The set Σ of FMs representing the state mapping from F to F' consists of projection FMs for all unmodified factors, since these projection FMs represent the identity function on the level of single factors (their tables are identity functions on the domain of their variable, a factor of F, to their associated factor in F', which are identical). The FM associated with the modified factor represents the function which applies the state mapping α to a state of F projected onto Θ. This means that for a given state s of F, interpreted as an assignment over its factors, Σ maps s to a state s' by mapping all component states of variables different from Θ to themselves, and by mapping the component state s_Θ of s in Θ to the state $\alpha(s_\Theta)$ of Θ^α.

We call such a transformation *shrinking* because we usually choose a state mapping α that maps the states of Θ to some smaller set, hence effectively reducing the number of states in the transformed (factored) transition system. In that way, shrinking offers a controlled way to effectively reduce the size of the factors of a factored transition system. This is useful since computing products as the merge transformation quickly grows large factors. The way in which the state mapping α reduces the states of a factor determines how big the loss of information is, hence offering a trade-off between heuristic quality and size of individual factors.

We now consider composing a shrink transformation with a previous transformation, as it is required by the merge-and-shrink framework. While we already defined a general procedure for composing sets of FMs of two factored transition systems, shown in Algorithm 1, a shrink transformation can be composed with a previous transformation more efficiently. Instead of copying all FMs of the previous transformation and then applying the table functions of the atomic FMs of Σ (all of which except one are identity functions) to the copies, we can directly reuse all FMs associated with factors that are not affected by the transformation. Also the FM of the previous transformation that is associated with the factor for which α is defined can be reused, however its table function needs to be updated by applying the state mapping α to it.

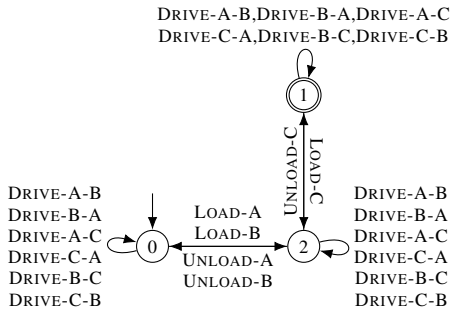
Formally, let F be the original input factored transition system to the merge-and-shrink algorithm, and let F' be a factored transition system that results from applying a previous transformation $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ to F with $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$. Let further $\tau'_{\text{F}} = \langle F'', \Sigma', \lambda' \rangle$ be a shrink transformation of F' into factored transition system F'', based on a state mapping $\alpha : \Theta_{\text{shrink}} \to \Theta^\alpha_{\text{shrink}}$ for some $\Theta_{\text{shrink}} \in F'$. Then we can compute Σ'' as the composition of Σ' with Σ as follows: $\Sigma'' = (\sigma''_{\Theta''})_{\Theta'' \in F''}$, where $\sigma''_{\Theta''} = \sigma_{\Theta''}$ if $\Theta'' \neq \Theta^\alpha_{\text{shrink}}$ (because then $\Theta'' = \Theta'$ for some $\Theta' \in F'$, and hence $\sigma_{\Theta''} \in \Sigma$), and $\sigma''_{\Theta^\alpha_{\text{shrink}}} = \sigma_{\Theta_{\text{shrink}}}$, however with $\sigma''^{\text{tab}}_{\Theta^\alpha_{\text{shrink}}}(s) := \alpha(\sigma^{\text{tab}}_{\Theta_{\text{shrink}}})$.

In the example of Figure 3.5 on page 44 that illustrates the composition of sets of FMs, the second transformation illustrates, among others, the application of a state mapping to a single



$\sigma_{\Theta(v_P)}$	$dom(v_P)$	A	B	C	T
		0	1	2	3

(a) Atomic factor $\Theta(v_P)$ (top) and associated FM $\sigma_{\Theta(v_P)}$ (bottom).



$\tau(\sigma_{\Theta(v_P)})$	$dom(v_P)$	A	B	C	T
		0	0	1	2

(b) Transformed factor $\tau(\Theta(v_P))$ (top) and transformed associated FM $\tau(\sigma_{\Theta(v_P)})$ (bottom), after applying a shrink transformation τ that combines states 0 and 1 into a new state 0, and renames 2 to 1 and 3 to 2.

Figure 3.7.: Example of a shrink transformation that applies a state mapping to the factor $\Theta(v_P)$ of the induced factored transition system of the example planning task of Figure 2.1.

FM as done by shrink transformations: the FM $\sigma_{\Theta''_2}$ applies a state mapping to Θ'_3 that combines states 2 and 3 into a single state 2 in the transformed factor. In the text of the example, we explained the composition of that particular FM with the sets of FMs of the previous transformation, however using the general algorithm and not the above approach tailored to compose a shrink transformation.

Consider the factored transition system induced by the example planning task of Figure 2.1. An example shrink transformation τ applies a state mapping α to the atomic factor $\Theta(v_P)$, combining states 0 and 1 (which correspond to values A and B of variable v_P). Figure 3.7a shows the atomic factor $\Theta(v_P)$ (top) and the associated FM $\sigma_{\Theta(v_P)}$ representing the identity state mapping (bottom). In Figure 3.7b, we see the shrunk transition system $\tau(\Theta(v_P))$ (top) and the transformed associated FM $\tau(\sigma_{\Theta(v_P)})$ (bottom). In the shrunk transition system, states 0 and 1 are combined into a new state 0, and states 2 and 3 are renamed to 1 and 2 for continuous numbering. The transitions are transformed accordingly, i.e. each original transition induces a transition in the transformed transition system. In the transformed associated FM, we see that both values A and B are now mapped to the new state 0 of the shrunk transition system, and also C and T are mapped to 1 and 2 to correspond to the states of the shrunk transition system.

Theorem 3.6. *Let F be a factored transition system. A shrink transformation τ_{F} of F into a factored transition system F' is a strict homomorphism, i.e. satisfies* **CONS** + **IND**. *It additionally satisfies* $\mathbf{REF_C}$.

Proof. Let τ_{F} be based on a state mapping α defined on some $\Theta_{\text{shrink}} \in F$, where $\Theta^{\alpha}_{\text{shrink}} \in F'$ is the transition system induced by Θ_{shrink} and α, i.e. $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ with $\lambda = \mathrm{id}$ and $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ where $\sigma_{\Theta'}(s) = \pi_{\Theta'}$ for all $\Theta' \in F'$ with $\Theta' \neq \Theta^{\alpha}_{\text{shrink}}$, and $\sigma^{\text{tab}}_{\Theta^{\alpha}_{\text{shrink}}}(s) = \alpha(s[\Theta_{\text{shrink}}])$. Let $\tau = \langle \bigotimes F', \sigma, \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}. In

the following, we show that the induced transformation satisfies each of the claimed properties $\mathbf{CONS_T}$, $\mathbf{CONS_G}$, $\mathbf{CONS_C}$, $\mathbf{IND_S}$, $\mathbf{IND_L}$, $\mathbf{IND_T}$, $\mathbf{IND_G}$, $\mathbf{IND_C}$, and $\mathbf{REF_C}$. Since $\lambda = \mathrm{id}$, we simplify the notation by dropping the use of λ for transformed transitions.

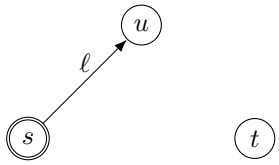
- Consider a state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta']$ is a state of Θ' for all $\Theta' \in F'$. Because α is surjective, there exists a state s_Θ of Θ_{shrink} with $\alpha(s_{\Theta_{\text{shrink}}}) = s'[\Theta^\alpha_{\text{shrink}}]$. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^\alpha_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta']$ is a state of Θ. Put together, with setting $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{shrink}}, \Theta' \neq \Theta^\alpha_{\text{shrink}}, \Theta' = \Theta\} \cup \{\Theta_{\text{shrink}} \mapsto s_{\Theta_{\text{shrink}}}\}$ for some $s_{\Theta_{\text{shrink}}} \in \alpha^{-1}(s'[\Theta^\alpha_{\text{shrink}}])$, we have that s is a state of $\bigotimes F$ with $\sigma(s) = s'$, which shows that σ is surjective and thus τ satisfies $\mathbf{IND_S}$.

- Consider a transition $s \xrightarrow{\ell} t \in \bigotimes F$. By the definition of products, $s[\Theta] \xrightarrow{\ell} t[\Theta] \in \Theta$ for all $\Theta \in F$. Furthermore, for all $\Theta' \in F'$ with $\Theta' \neq \Theta^\alpha_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence we have $\sigma_{\Theta'}(s) \xrightarrow{\ell} \sigma_{\Theta'}(t) \in \Theta'$ because $\sigma_{\Theta'}$ is the projection onto Θ' and $\Theta' = \Theta$. For $\Theta^\alpha_{\text{shrink}}$, we also have that $\sigma_{\Theta^\alpha_{\text{shrink}}}(s) \xrightarrow{\ell} \sigma_{\Theta^\alpha_{\text{shrink}}}(t) \in \Theta^\alpha_{\text{shrink}}$ because $\Theta^\alpha_{\text{shrink}}$ is the transition system induced by Θ_{shrink} and α, and $\sigma_{\Theta^\alpha_{\text{shrink}}}$ is defined based on α and the projection onto Θ_{shrink}. Together, this implies that $\sigma_{\Theta'}(s) \xrightarrow{\ell} \sigma_{\Theta'}(t) \in \Theta'$ for all $\Theta' \in F'$, and hence by the definition of products, $\sigma(s) \xrightarrow{\ell} \sigma(t) \in \bigotimes F'$, which shows that τ satisfies $\mathbf{CONS_T}$.

- Consider a transition $s' \xrightarrow{\ell} t' \in \bigotimes F'$. By the definition of products, $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta'$ for all $\Theta' \in F'$. Because $\Theta^\alpha_{\text{shrink}}$ is induced by Θ_{shrink} and α, there exists $s_{\Theta_{\text{shrink}}} \xrightarrow{\ell} t_{\Theta_{\text{shrink}}} \in \Theta_{\text{shrink}}$ with $\alpha(s_{\Theta_{\text{shrink}}}) = s'[\Theta^\alpha_{\text{shrink}}]$ and $\alpha(t_{\Theta_{\text{shrink}}}) = t'[\Theta^\alpha_{\text{shrink}}]$. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^\alpha_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta$. Put together, with setting $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{shrink}}, \Theta' \neq \Theta^\alpha_{\text{shrink}}, \Theta' = \Theta\} \cup \{\Theta_{\text{shrink}} \mapsto s_{\Theta_{\text{shrink}}}\}$ for some $s_{\Theta_{\text{shrink}}} \in \alpha^{-1}(s'[\Theta^\alpha_{\text{shrink}}])$, and analogously $t = \{\Theta \mapsto t'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{shrink}}, \Theta' \neq \Theta^\alpha_{\text{shrink}}, \Theta' = \Theta\} \cup \{\Theta_{\text{shrink}} \mapsto t_{\Theta_{\text{shrink}}}\}$ for some $t_{\Theta_{\text{shrink}}} \in \alpha^{-1}(t'[\Theta^\alpha_{\text{shrink}}])$, we have that $s \xrightarrow{\ell} t \in \bigotimes F$ with $\sigma(s) = s'$ and $\sigma(t) = t'$, which shows that τ satisfies $\mathbf{IND_T}$.

- Consider a goal state $s \in \bigotimes F$. By the definition of products, $s[\Theta]$ is a goal state of Θ for all $\Theta \in F$. Furthermore, for all $\Theta' \in F'$ with $\Theta' \neq \Theta^\alpha_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence we have that $\sigma_{\Theta'}(s)$ is a goal state of Θ' because $\sigma_{\Theta'}$ is the projection onto Θ' and $\Theta' = \Theta$. For $\Theta^\alpha_{\text{shrink}}$, we also have that $\sigma_{\Theta^\alpha_{\text{shrink}}}(s)$ is a goal state of $\Theta^\alpha_{\text{shrink}}$ because $\Theta^\alpha_{\text{shrink}}$ is the transition system induced by Θ_{shrink} and α, and $\sigma_{\Theta^\alpha_{\text{shrink}}}$ is defined based on α and the projection onto Θ_{shrink}. Together, this implies that $\sigma_{\Theta'}(s)$ is a goal state of Θ' for all $\Theta' \in F'$, and hence by the definition of products, $\sigma(s)$ is a goal state of $\bigotimes F'$, which shows that τ satisfies $\mathbf{CONS_G}$.

- Consider a goal state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta']$ is a goal state of Θ' for all $\Theta' \in F'$. Because $\Theta^\alpha_{\text{shrink}}$ is induced by Θ_{shrink} and α, there exists a goal state $s_{\Theta_{\text{shrink}}}$ of Θ_{shrink} with $\alpha(s_{\Theta_{\text{shrink}}}) = s'[\Theta^\alpha_{\text{shrink}}]$. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^\alpha_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta']$ is a goal state of Θ. Put together, with setting $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{shrink}}, \Theta' \neq$



(a) Transition system Θ.

(b) Transition system Θ^α induced by α.

Figure 3.8.: Transition system Θ and transition system Θ^α induced by state mapping α mapping states s and t to u'.

$\Theta^\alpha_{\text{shrink}}, \Theta' = \Theta\} \cup \{\Theta_{\text{shrink}} \mapsto s_{\Theta_{\text{shrink}}}\}$ for some $s_{\Theta_{\text{shrink}}} \in \alpha^{-1}(s'[\Theta^\alpha_{\text{shrink}}])$, we have that s is a goal state of $\bigotimes F$ with $\sigma(s) = s'$, which shows that τ satisfies $\mathbf{IND_G}$.

- Finally, it is easy to see that τ_F satisfies $\mathbf{IND_L}$, $\mathbf{CONS_C}$, $\mathbf{IND_C}$, and $\mathbf{REF_C}$ because $\lambda = \text{id}$ and hence labels (and their cost) are not changed.

□

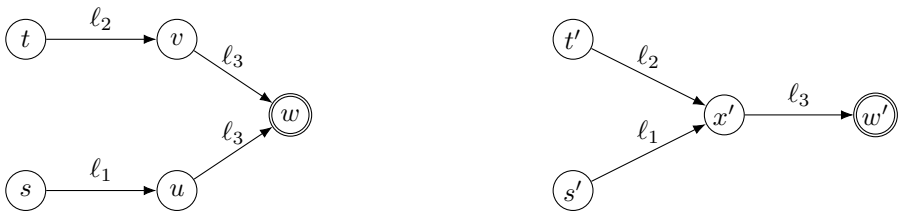
This theorem states that shrink transformations are strict homomorphisms that additionally preserve costs. Together with Theorem 3.3, an immediate consequence is that the heuristic for the factored transition system induced by a shrink transformation is admissible and consistent. If we did not define the shrink transformation to replace a factor by the transition system induced by the state mapping, but allowed to use a non-surjective state mapping, then shrink transformations would not be (state-)induced and thus only be a non-strict homomorphism. However, as discussed before, induced abstractions are in some sense the best abstractions we can have, and hence it is a positive result that shrink transformations based on surjective state mappings are strict homomorphisms.

To see that a shrink transformation in general is not exact, consider the following counter example: let Θ be the only factor of some factored transition system, shown in Figure 3.8a. Let α be a state mapping of a shrink transformation τ that maps s and t to v and u to w. In the transition system Θ^α induced by Θ and α, shown in Figure 3.8b, v is a goal state and there is a transition $v \xrightarrow{\ell} w$. τ does not satisfy $\mathbf{REF_T}$ because $v \xrightarrow{\ell} w$ cannot be refined for *all* preimages s, t of v, i.e. there is no transition from t labeled ℓ in Θ. τ does not satisfy $\mathbf{REF_G}$ because not *all* preimages s, t of the goal state v are goal states in Θ.

However, it is desirable and possible to find criteria for shrinking such that shrink transformations also satisfy $\mathbf{REF_T}$ and $\mathbf{REF_G}$, turning them into exact transformations. One such criterion is bisimulation (Milner, 1990).

Definition 3.14 (Bisimulation). *Let $\Theta = \langle S, L, c, T, s_0, S_\star\rangle$ be a transition system. An equivalence relation $\sim$ on S is a* bisimulation *for Θ if*

1. *$s \sim t$ implies that either $s, t \in S_\star$ or $s, t \notin S_\star$, and*
2. *for every pair of states $s, t \in S$ with $s \sim t$, and for every label $\ell \in L$, if $\langle s, \ell, s'\rangle \in T$, then there exists t' with $\langle t, \ell, t'\rangle \in T$ and $s' \sim t'$.*



(a) Transition system Θ.

(b) Transition system Θ^α induced by α.

Figure 3.9.: Transition system Θ and transition system Θ^α induced by state mapping α, based on bisimulation, mapping states u and v to x'.

Intuitively, states are bisimilar if they are both goal states or both non-goal states, and outgoing transitions labeled with the same label lead to bisimilar states. Figure 3.9a shows an example transition system Θ where states u and v are bisimilar, i.e. $u \sim v$, because both have outgoing transitions labeled with the same label (ℓ_3) that lead to bisimilar states (here: the same state w). On the other hand, s and t are not bisimilar ($s \not\sim t$) because their outgoing transitions, although leading to bisimilar states, are labeled with different labels.

Shrinking based on bisimulation (Nissim et al., 2011) applies a transformation based on a state mapping that combines only bisimilar states of a factor. Nissim et al. (2011) show that this kind of shrinking is information-preserving. We repeat this result, adapted to our transformation framework.

Theorem 3.7. *Let F be a factored transition system. Let τ_F be a shrink transformation of F into a factored transition system F' based on a state mapping α defined on some $\Theta_\text{shrink} \in F$, where $\Theta^\alpha_\text{shrink} \in F'$ is the transition system induced by Θ_shrink and α. Furthermore, let α be based on bisimulation, i.e. $\alpha(s) = \alpha(t)$ for all states $s, t \in \Theta_\text{shrink}$ iff s and t are bisimilar. Such a transformation τ_F is exact, i.e. satisfies* **CONS** + **IND** + **REF**.

Proof. Let $\tau_\mathrm{F} = \langle F', \Sigma, \lambda\rangle$ with $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ and $\sigma_{\Theta'}(s) = \pi_{\Theta'}$ for all $\Theta' \in F'$ with $\Theta' \neq \Theta^\alpha_\text{shrink}$, and $\sigma^\text{tab}_{\Theta^\alpha_\text{shrink}}(s) = \alpha(s[\Theta_\text{shrink}])$. Let $\tau = \langle \bigotimes F', \sigma, \lambda\rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_F. Since shrink transformations are strict homomorphisms that satisfy $\mathbf{REF_C}$, we only need to show that the induced transformation τ satisfies $\mathbf{REF_T}$ and $\mathbf{REF_G}$.

- Consider a transition $s' \xrightarrow{\ell} t' \in \bigotimes F'$. By the definition of products, $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta'$ for all $\Theta' \in F'$. Because $\Theta^\alpha_\text{shrink}$ is induced by Θ_shrink and α, and because α combines only states that are bisimilar, for all $s_{\Theta_\text{shrink}} \in \alpha^{-1}(s'[\Theta^\alpha_\text{shrink}])$, there exists $s_{\Theta_\text{shrink}} \xrightarrow{\ell} t_{\Theta_\text{shrink}} \in \Theta_\text{shrink}$ with $\alpha(t_{\Theta_\text{shrink}}) = t'[\Theta^\alpha_\text{shrink}]$ (and s_{Θ_shrink} and t_{Θ_shrink} are bisimilar, cf. point 2. of the definition of bisimulation). For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^\alpha_\text{shrink}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence for the only preimage $s'[\Theta'] \in \Theta$ of $s'[\Theta'] \in \Theta'$, $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta$ because $\Theta = \Theta'$. Put together, for all states $s \xrightarrow{\ell} t \in \bigotimes F$ with $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_\text{shrink}, \Theta' \neq \Theta^\alpha_\text{shrink}, \Theta' = \Theta\} \cup \{\Theta_\text{shrink} \mapsto s_{\Theta_\text{shrink}}\}$ with $s_{\Theta_\text{shrink}} \in \alpha^{-1}(s'[\Theta^\alpha_\text{shrink}])$, and thus with $\sigma(s) = s'$, there exists $s \xrightarrow{\ell} t \in \bigotimes F$ with $t = \{\Theta \mapsto t'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_\text{shrink}, \Theta' \neq \Theta^\alpha_\text{shrink}, \Theta' = \Theta\} \cup \{\Theta_\text{shrink} \mapsto$

$t_{\Theta_{\text{shrink}}}\}$ for some $t_{\Theta_{\text{shrink}}} \in \alpha^{-1}(t'[\Theta^{\alpha}_{\text{shrink}}])$, and thus with $\sigma(t) = t'$, which shows that τ satisfies $\mathbf{REF_T}$.

- Consider a goal state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta']$ is a goal state of Θ' for all $\Theta' \in F'$. Because $\Theta^{\alpha}_{\text{shrink}}$ is induced by Θ_{shrink} and α, and because alpha combines only bisimilar states, all preimage states of goal states are goal states, i.e. all states $s_{\Theta_{\text{shrink}}} \in \alpha^{-1}(s'[\Theta^{\alpha}_{\text{shrink}}])$ of are goal states. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^{\alpha}_{\text{shrink}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta'] \in \Theta$ is the only preimage state of $s'[\Theta'] \in \Theta'$, and because $\Theta = \Theta'$, it is a goal state. Put together, all states $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{shrink}}, \Theta' \neq \Theta^{\alpha}_{\text{shrink}}, \Theta' = \Theta\} \cup \{\Theta_{\text{shrink}} \mapsto s_{\Theta_{\text{shrink}}}\}$ with $s_{\Theta_{\text{shrink}}} \in \alpha^{-1}(s'[\Theta^{\alpha}_{\text{shrink}}])$ are goal states of $\bigotimes F$ with $\sigma(s) = s'$, which shows that τ satisfies $\mathbf{REF_G}$.

□

Figure 3.9a shows a transition system Θ and Figure 3.9b shows the transition system Θ^{α} induced by Θ and a state mapping α based on bisimulation. It is easy to verify that the states and transitions of the induced transition system can be refined to the corresponding states and transitions of the original transition system Θ. In particular, for the transition $x' \xrightarrow{\ell} w'$, we have that for *all* preimages v, u of the state x', there exist transitions labeled with ℓ_3 that lead to w, a preimage of w': both $v \xrightarrow{\ell_3} w$ and $u \xrightarrow{\ell_3} w$ are transitions of Θ.

3.4. Merge Transformation

In this section, we define the merge transformation of the merge-and-shrink framework.

Definition 3.15 (Merge Transformation)**.** *Let F be a factored transition system, and let $\Theta_1, \Theta_2 \in F$. A* merge transformation *is a factored transformation $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ of F into a factored transition system F', where:*

- *$F' = (F \setminus \{\Theta_1, \Theta_2\}) \cup \{\Theta^{\otimes}\}$ with $\Theta^{\otimes} = \Theta_1 \otimes \Theta_2$.*
- *$\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ is an indexed collection of FMs, where each FM $\sigma_{\Theta'}$ is defined on F and associated with $\Theta' \in F'$: $\sigma_{\Theta'} := \pi_{\Theta'}$ for all $\Theta' \in F'$ with $\Theta' \neq \Theta^{\otimes}$, and $\sigma_{\Theta^{\otimes}}$ is a merge FM with two atomic component FMs $\sigma_{\Theta_1} := \pi_{\Theta_1}$ and $\sigma_{\Theta_2} := \pi_{\Theta_2}$, and a table defined as $\sigma^{\text{tab}}_{\Theta^{\otimes}}(s) := \langle s[\Theta_1], s[\Theta_2] \rangle$ for all states s of F.*
- *$\lambda = \text{id}$ is the identity label mapping.*

Informally speaking, the merge transformation replaces two factors of the given factored transition system by their product system, leaves all other factors unchanged, and does not change the set of labels. The set Σ of FMs representing the state mapping from F to F' consists of projection FMs for all unmodified factors, since these projection FMs represent the identity function on the level of single factors (their tables are identity functions on the domain of their variable, a factor of F, to their associated factor in F', which are identical). The FM associated with the modified factor is a merge FM with two projection FMs for the merged factors Θ_1 and Θ_2

as its components, and its table is the identity function for merge FMs, i.e. it maps each pair of values of its component FMs, which correspond to states of Θ_1 and Θ_2, to a unique value corresponding to their product state in $\Theta^{\otimes}$.

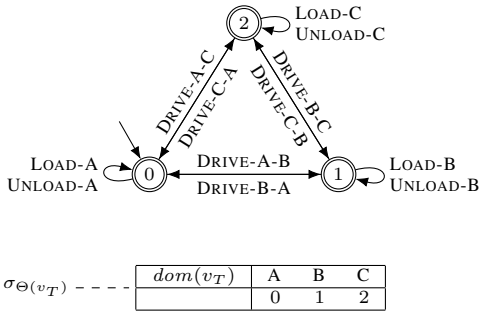
We now consider composing a merge transformation with a previous transformation, as it is required by the merge-and-shrink framework. While we already defined a general procedure for composing sets of FMs of two factored transition systems, shown in Algorithm 1, a merge transformation can be composed with a previous transformation more efficiently. Instead of copying all FMs of the previous transformation and then applying the table functions of the atomic FMs of Σ (all of which except one are identity functions) to the copies, we can directly reuse all FMs associated with factors that are not affected by the transformation. Also the two FMs of the previous transformation that are associated with the factors that are being merged can be reused: instead of copying them and constructing the merge FM based on the two copies, we can directly use the two FMs (even without the need to modify their table functions because the leaf FMs of the merge FM represent the identity function).

Formally, let F be the original input factored transition system to the merge-and-shrink algorithm, and let F' be a factored transition system that results from applying a previous transformation $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ to F with $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$. Let further $\tau'_{\mathrm{F}} = \langle F'', \Sigma', \lambda' \rangle$ be a merge transformation of F' into factored transition system F'' with $F'' = F' \setminus \{\Theta'_1, \Theta'_2\} \cup \{\Theta^{\otimes}\}$. Then we can compute Σ'' as the composition of Σ' with Σ as follows: $\Sigma'' = (\sigma''_{\Theta''})_{\Theta'' \in F''}$, where $\sigma''_{\Theta''} = \sigma_{\Theta''}$ if $\Theta'' \neq \Theta^{\otimes}$ (because then $\Theta'' = \Theta'$ for some $\Theta' \in F'$, and hence $\sigma_{\Theta''} \in \Sigma$), and $\sigma''_{\Theta^{\otimes}}$ is the merge FM with components $\sigma_{\Theta'_1}$ and $\sigma_{\Theta'_2}$ and table function as defined by the merge transformation.

In the example of Figure 3.5 on page 44 that illustrates the composition of sets of FMs, the second transformation illustrates, among others, merging two FMs: the FM $\sigma_{\Theta'_1}$ represents a merge of the factors Θ'_1 and Θ'_2. In the text of the example, we explained the composition of that particular FM with the sets of FMs of an arbitrary previous transformation, however using the general algorithm and not the above approach tailored to compose a merge transformation.

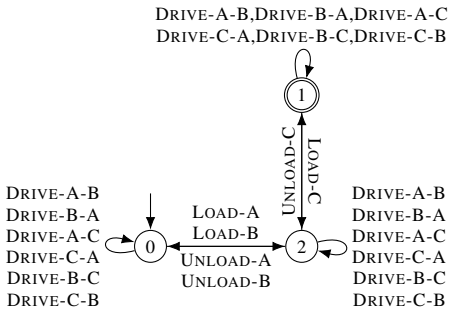
Consider the factored transition system of the example planning task of Figure 2.1 that consists of the atomic factor $\Theta(v_T)$ with associated FM $\sigma_{\Theta(v_T)}$, and the factor $\tau(\Theta(v_P))$ with associated FM $\tau(\sigma_{\Theta(v_P)})$ as in the shrinking example in Figure 3.7. Figures 3.10a and 3.10b show the two factors (top) with their associated FMs (bottom). In the factored transition system transformed by a merge transformation, shown in Figure 3.10c (top) with the associated FM (bottom), the two factors are replaced by their product $\Theta^{\otimes} := \Theta(v_T) \otimes \tau(\Theta(v_P))$ (cf. the illustration of computing products in Section 3.2 on page 35), and the associated FM $\sigma_{\Theta^{\otimes}}$ is the merge FM of the two component FMs $\sigma_{\Theta(v_T)}$ and $\tau(\sigma_{\Theta(v_P)})$, with a table function that maps each pair of component values to a unique value, i.e. maps each pair of states of the replaced factors to their product state in the product system. (To better allow identifying product states by their component states, the figure indicates the component states x and y of the product states by showing pairs $\langle x, y \rangle$ below or above the product state. Of course, the state mapping can also be read from the table of $\sigma_{\Theta^{\otimes}}$.)

As a remark on how shrinking and merging interacts, we observe that $\Theta^{\otimes}$ differs from the induced transition system $\Theta(\Pi)$ shown in Figure 2.2. The difference, apart from names of states, stems from the merge transformation being performed on the shrunk factor. The same result could have been obtained from shrinking $\Theta(\Pi)$ by combining each pair of states where



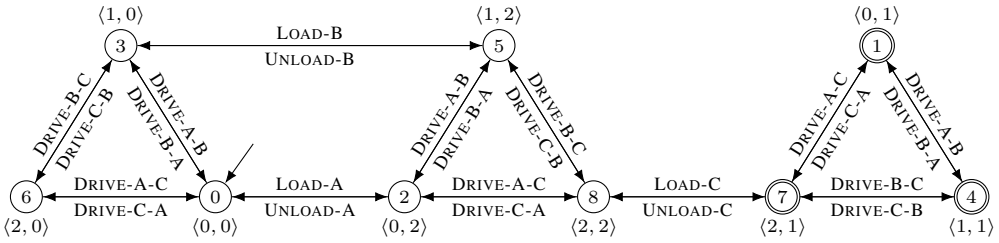
$dom(v_T)$	A	B	C
	0	1	2

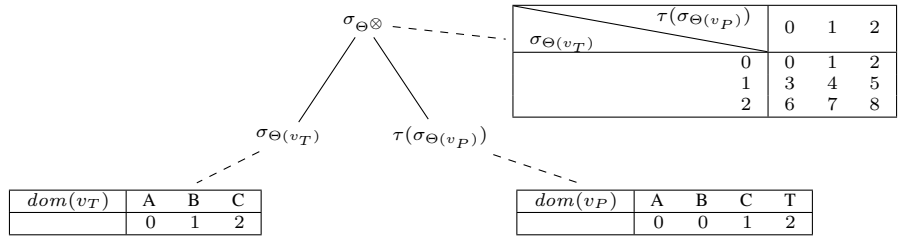
(a) Atomic factor $\Theta(v_T)$ (top) and associated FM $\sigma_{\Theta(v_T)}$ (bottom).



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

(b) Shrunk factor $\tau(\Theta(v_P))$ (top), cf. Figure 3.7, and associated FM $\tau(\sigma_{\Theta(v_P)})$ (bottom).





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

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

(c) Transformed factored transition system with a single factor $\Theta^\otimes$ (top) and associated FM $\sigma_{\Theta^\otimes}$ (bottom).

Figure 3.10.: Example of a merge transformation applied to the factored transition system of the example planning task of Figure 2.1, with the factor $\Theta(v_P)$ shrunk as in Figure 3.7.

(a) Example transition system Θ.

(b) Product of Θ and Θ.

Figure 3.11.: Example transition system Θ and its product with itself.

the truck is at one specific location and the package is at A or B into a single state, which corresponds to the abstraction applied to the atomic factor $\Theta(v_P)$ in the shrinking example in Figure 3.7.

Theorem 3.8. *Let F and F' be factored transition systems with the same label sets and label costs. A merge transformation τ_{F} of F into F' is exact induced, i.e. satisfies properties* **CONS** + **IND** + **REF**.

Proof. From $\bigotimes F$ being defined as the product over all transition systems in F and $\otimes$ being a commutative and associative operator, it is clear that replacing two factors from F by their product does not change $\bigotimes F$, i.e. $\bigotimes F \backsimeq \bigotimes F'$, modulo names of states. With $\lambda = \mathrm{id}$ and σ mapping corresponding states of $\bigotimes F$ and $\bigotimes F'$, we immediately have that τ_{F} is exact induced. □

Together with Theorem 3.1, an immediate consequence is that we can recover the entire state space of an underlying planning task from merging all individual transition systems of the induced factored transition system, e.g. the global transition system $\bigotimes F$ of the induced factored transition system F of a planning task Π equals $\Theta(\Pi)$, modulo names of states (a result already shown in previous work, e.g. Helmert et al., 2014).

3.4.1. Non-orthogonal Merge Transformations

In the following, we briefly discuss the possibility of non-orthogonal merge transformations. Recall that a non-orthogonal FM may contain several atomic FMs with the same associated variable. In the context of merge-and-shrink, this means that the factored transition system could contain the same atomic factors several times (or several products resulting from merging the same atomic factors). As we have shown above, the merge transformation is state-induced, and as such the involved FMs are always orthogonal. This is also true for all other merge-and-shrink transformations. (Intuitively, they do not modify the number of factors or FMs, and hence can never turn a orthogonal FM into a non-orthogonal one.)

A simple way to allow non-orthogonal merge-and-shrink is to add a new "clone" transformation that clones a factor and the associated FM and adds it to the factored transition system and the set of FMs. Then, using the regular orthogonal merge transformation we defined above, we can replace the clones by their product and still retain the original components as part of the

factored transition system. It is easy to see that the clone transformation is not state-induced, i.e. the state mapping is not surjective. Consider a transformation τ_{F} of F into factored transition system F' that clones the single element $\Theta \in F$, shown in Figure 3.11a. Hence F' contains two copies of Θ. The transformation τ induced by τ_{F} transforms $\bigotimes F$ into $\bigotimes F'$, using state mapping $\sigma : \bigotimes F \mapsto \bigotimes F'$. We observe that $\bigotimes F$ equals Θ. $\bigotimes F'$ is the product of Θ and Θ, shown in Figure 3.11b. Clearly τ does not satisfy $\mathbf{IND_S}$, i.e. the state mapping σ is not surjective, because states st and ts have no preimages in $\bigotimes F$ ($\sigma(s) = ss$ and $\sigma(t) = tt$). For the same reason, τ is not goal-induced, i.e. does not satisfy $\mathbf{IND_G}$. It is also not transition-induced ($\mathbf{IND_T}$) because $ss \xrightarrow{\ell} st$ and $ss \xrightarrow{\ell} ts$ have no corresponding transitions in $\bigotimes F$ that induce them.

The clone transformation satisfies all other properties: it satisfies **CONS** because all (goal) states and transitions of the original product system have corresponding (goal) states and transitions in the transformed product system. Perhaps less obvious, the clone transformation is also refinable, i.e. satisfies **REF**: each state of the transformed transition system either has an empty preimage (if it is not induced), or the preimage contains a single state inducing the state in question. Thus, since we need to quantify over all states of the preimages, if the preimage is empty, the condition is vacuously true, and otherwise, the single element of the preimage is mapped to the state in question. A similar argument can be made for goal states and transitions. Finally, for all properties related to label costs, it is clear that they hold since the set of labels is not affected by the clone transformation. To attempt an intuitive explanation, cloning only adds spurious states and transitions but does not remove any, and hence the transformed transition system contains the states and transitions of the original transition system, plus additional ones. In the above example, ss and tt and the transitions $ss \xrightarrow{\ell} ss$ and $ss \xrightarrow{\ell} tt$ correspond to s, t, $s \xrightarrow{\ell} s$, and $s \xrightarrow{\ell} t$.

In the application to planning tasks, the merge-and-shrink algorithm starts with the induced factored transition of a planning task which is orthogonal. Since except the clone transformation just described, all merge-and-shrink transformations use orthogonal FMs, we cannot obtain non-orthogonal merge-and-shrink abstractions for planning tasks with the usual transformations.

3.5. Prune Transformation

In this section, we define the prune transformation. Pruning has only been treated as a side-note previously (e.g. Helmert et al., 2014, Section 4.3), but like label reduction, it has been an important ingredient of the merge-and-shrink framework since its introduction for planning.

Definition 3.16 (Pruning State Mapping)**.** *Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system. A* pruning state mapping *ρ for Θ is a function $\rho : S \to S'$ where $S' = S_{\text{sub}} \cup \{\bot\}$ with $S_{\text{sub}} \subseteq S$ and $\rho(s) = s$ for all $s \in S_{\text{sub}}$ and $\rho(s) = \bot$ otherwise. The* transition system Θ pruned by *ρ is defined as $\Theta^\rho = \langle S', L, c, \{\langle \rho(s), \ell, \rho(s') \rangle \mid \langle s, \ell, s' \rangle \in T \text{ and } s, s' \in S_{\text{sub}}\}, \rho(s_0), S_\star \cap S_{\text{sub}} \rangle$.*

In words, a pruning state mapping is a state mapping, restricted to be the identity mapping on a subset S_{sub} of the original set of states and to map all other states to a special state $\bot$. The transition system pruned by a pruning state mapping is similar to the transition system induced by an arbitrary state mapping, with the difference that all original transitions from and to states

not in S_{sub} are not part of $\rho(\Theta)$, and that all goal states not in S_{sub} are mapped to the non-goal state $\bot$.

Definition 3.17 (Prune Transformation)**.** *Let F be a factored transition system, and let $\Theta \in F$. Let ρ be a pruning state mapping for Θ. A* prune transformation *is a factored transformation $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ of F into a factored transition system F', where:*

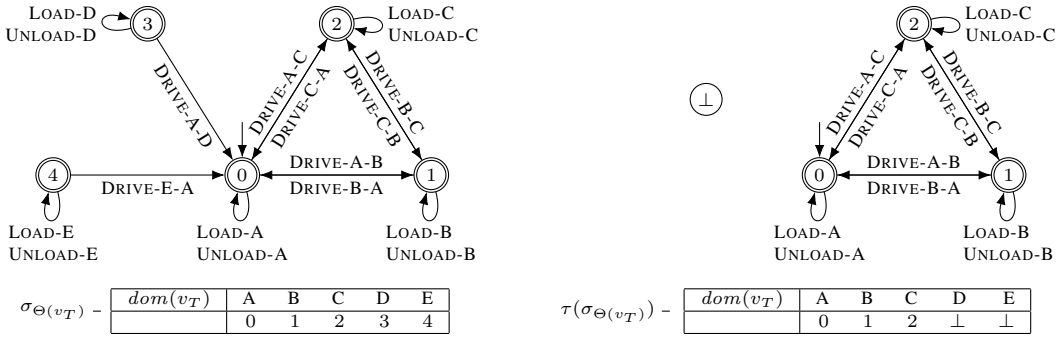
- *$F' = (F \setminus \{\Theta\}) \cup \{\Theta^\rho\}$ with Θ^ρ the transition system Θ pruned by ρ.*
- *$\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ is an indexed collection of FMs, where each FM $\sigma_{\Theta'}$ is defined on F and associated with $\Theta' \in F'$: $\sigma_{\Theta'} := \pi_{\Theta'}$ for all $\Theta' \in F'$ with $\Theta' \neq \Theta^\rho$, and σ_{Θ^ρ} is an atomic FM with variable Θ and $\sigma^{\text{tab}}_{\Theta^\rho}(s) := \rho(s[\Theta])$.*
- *$\lambda = \text{id}$ is the identity label mapping.*

Informally speaking, a prune transformation is very similar to a shrink transformation in that it also applies a state mapping to a single factor of the factored transition system, leaving all other factors as they are. The difference is that the transformed transition system is not induced, but pruned as specified by the pruning state mapping of the transformation. As with shrink transformations, the set Σ of FMs representing the state mapping from F to F' consists of projection FMs for all unmodified factors representing the identity function on the level of single factors. The FM associated with the modified factor represents the function which applies the pruning state mapping ρ to a state of F projected onto Θ.

We call such a transformation *pruning* because the transformation isolates all states that are not in the subset S_{sub} of states by removing transitions coming from or leading to states of S_{sub}. Such isolated states are necessarily unreachable and irrelevant, i.e. dead states, and as such not part of any plans of the transformed transition system. Pruning thus offers a controlled way of removing parts of a factor which are deemed irrelevant to the computation of merge-and-shrink abstractions. Usual merge-and-shrink algorithms use pruning transformations to isolate dead states. The purpose of such pruning of dead states is to reduce the size of factors, improving the efficiency of the merge-and-shrink computation, but also allowing the use of fewer lossy shrink transformations.

We now consider composing a prune transformation with a previous transformation, as it is required by the merge-and-shrink framework. The way this composition works is analogous to how shrink transformations can be composed with previous transformations, but we still repeat the procedure here for self-containedness of this section. While we already defined a general procedure for composing sets of FMs of two factored transition systems, shown in Algorithm 1, a prune transformation can be composed with a previous transformation more efficiently. Instead of copying all FMs of the previous transformation and then applying the table functions of the atomic FMs of Σ (all of which except one are identity functions) to the copies, we can directly reuse all FMs associated with factors that are not affected by the transformation. Also the FM of the previous transformation that is associated with the factor for which ρ is defined can be reused, however its table function needs to be updated by applying the pruning state mapping ρ to it.

Formally, let F be the original input factored transition system to the merge-and-shrink algorithm, and let F' be a factored transition system that results from applying a previous transformation $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ to F with $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$. Let further $\tau'_{\text{F}} = \langle F'', \Sigma', \lambda' \rangle$ be a



(a) Atomic factor $\Theta(v_T)$ (top) and associated FM $\sigma_{\Theta(v_T)}$ (bottom).

(b) Pruned factor $\tau(\Theta(v_T))$ (top) and associated FM $\tau(\sigma_{\Theta(v_T)})$ (bottom).

Figure 3.12.: Example of a prune transformation applied to the factored transition system of the example planning task of Figure 2.1, modified to contain two more locations D and E which are only connected to A, but not from A.

prune transformation of F' into factored transition system F'', based on a pruning state mapping $\rho : \Theta_{\text{prune}} \to \Theta^{\rho}_{\text{prune}}$ for some $\Theta_{\text{prune}} \in F'$. Then we can compute Σ'' as the composition of Σ' with Σ as follows: $\Sigma'' = (\sigma''_{\Theta''})_{\Theta'' \in F''}$, where $\sigma''_{\Theta''} = \sigma_{\Theta''}$ if $\Theta'' \neq \Theta^{\rho}_{\text{prune}}$ (because then $\Theta'' = \Theta'$ for some $\Theta' \in F'$, and hence $\sigma_{\Theta''} \in \Sigma$), and $\sigma''_{\Theta^{\rho}_{\text{prune}}} = \sigma_{\Theta_{\text{prune}}}$, however with $\sigma''^{\text{tab}}_{\Theta^{\rho}_{\text{prune}}}(s) := \rho(\sigma^{\text{tab}}_{\Theta_{\text{prune}}})$.

Consider the example planning task of Figure 2.1, this time slightly modified such that there are a new locations D and E from where the truck T can move to A but where it *cannot* move to. Figure 3.12a shows the modified factor for v_T (top) and the associated FM representing the identity state mapping (bottom). The states corresponding to the truck being at the new locations D and E are numbered 3 and 4. (We observe that both states are unreachable, and thus both are dead already before applying the prune transformation.) A prune transformation τ that prunes these two states and thus maps them both to $\bot$ also removes all in- and outgoing transition and leaves everything else unchanged. Figure 3.12b shows the transformed factor of such a transformation τ (top) with the transformed associated FM (bottom). In the transformed transition system, we see that states 3 and 4 have been removed and there is a new state $\bot$ that is disconnected as intended. All remaining states and transitions are as in the original factor. In the transformed associated FM, we see that both values D and E of variable v_T are mapped to $\bot$, and the remainder of the table function is as in the original associated FM.

Theorem 3.9. *Let F be a factored transition system and let $\Theta_{\text{prune}} \in F$ be a factor of F with states S. Let τ_{F} be a prune transformation of F into a factored transition system F' based on a pruning state mapping $\rho : S \to S'$ where $S' = S_{\text{sub}} \cup \{\bot\}$ with $S_{\text{sub}} \subseteq S$. Let $S^{\otimes}_{\text{sub}} := \prod_{\Theta \in F, \Theta \neq \Theta_{\text{prune}}} S_{\Theta} \times S_{\text{sub}}$ where S_{Θ} denotes the states of Θ, i.e. $S^{\otimes}_{\text{sub}}$ is the subset of states of $\bigotimes F$ that are not pruned by ρ. τ_{F} is exact induced for $S^{\otimes}_{\text{sub}}$, i.e. satisfies* **CONS**$^{S^{\otimes}_{\text{sub}}}$+**IND**+**REF**.

Proof. Let $\Theta^{\rho}_{\text{prune}}$ be the transition system Θ_{prune} pruned by ρ. Then $\tau_{\text{F}} = \langle F', \Sigma, \lambda \rangle$ with $\lambda = \text{id}$ and $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ and $\sigma_{\Theta'}(s) = \pi_{\Theta'}$ for all $\Theta' \in F'$ with $\Theta' \neq \Theta^{\rho}_{\text{prune}}$, and $\sigma^{\text{tab}}_{\Theta^{\rho}_{\text{prune}}}(s) =$

$\rho(s[\Theta_{\text{prune}}])$. Let $\tau = \langle \bigotimes F', \sigma, \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}. In the following, we show that the induced transformation satisfies each of the claimed properties $\mathbf{CONS}_{\mathbf{T}}^{S_{\text{sub}}^{\otimes}}$, $\mathbf{CONS}_{\mathbf{G}}^{S_{\text{sub}}^{\otimes}}$, $\mathbf{CONS_C}$, $\mathbf{IND_S}$, $\mathbf{IND_L}$, $\mathbf{IND_T}$, $\mathbf{IND_G}$, $\mathbf{IND_C}$, $\mathbf{REF_T}$, $\mathbf{REF_G}$, and $\mathbf{REF_C}$. Since $\lambda = \text{id}$, we simplify the notation by dropping the use of λ for transformed transitions. As a general observation, we remark remark that the set T' of transitions of $\Theta_{\text{prune}}^{\rho}$ is a subset of the set of transitions induced by the set T of transitions of Θ_{prune} and ρ, and similarly, the set $S'_\star$ of goal states of $\Theta_{\text{prune}}^{\rho}$ is a subset of the set of states induced by the set $S_\star$ of goal states of Θ_{prune} and ρ. Furthermore, we have that all states $s \in S_{\text{sub}}$ are also states of $\Theta_{\text{prune}}^{\rho}$ because ρ is the identity mapping if restricted to these states, and hence $|\rho^{-1}(s)| = 1$ for all states $s \neq \bot$ of $\Theta_{\text{prune}}^{\rho}$.

- Consider a state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta']$ is a state of Θ' for all $\Theta' \in F'$. Because ρ is surjective, there exists a state s_Θ of Θ_{prune} with $\rho(s_{\Theta_{\text{prune}}}) = s'[\Theta_{\text{prune}}^{\rho}]$. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta_{\text{prune}}^{\rho}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta']$ is a state of Θ. Put together, there exists a state $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{prune}}, \Theta' \neq \Theta_{\text{prune}}^{\rho}, \Theta' = \Theta\} \cup \{\Theta_{\text{prune}} \mapsto s_{\Theta_{\text{prune}}}\}$ of $\bigotimes F$, for some $s_{\Theta_{\text{prune}}} \in \rho^{-1}(s'[\Theta_{\text{prune}}^{\rho}])$, and $\sigma(s) = s'$, which shows that τ satisfies $\mathbf{IND_S}$.

- Consider a transition $s \xrightarrow{\ell} t \in \bigotimes F$ within $S_{\text{sub}}^{\otimes}$. By the definition of products, $s[\Theta] \xrightarrow{\ell} t[\Theta] \in \Theta$ for all $\Theta \in F$. For $\Theta_{\text{prune}}^{\rho}$, because $s \xrightarrow{\ell} t$ is within $S_{\text{sub}}^{\otimes}$, we have $s[\Theta_{\text{prune}}]$, $t[\Theta_{\text{prune}}] \in S_{\text{sub}}$ and therefore $\sigma_{\Theta_{\text{prune}}^{\rho}}(s) = s[\Theta_{\text{prune}}] = s'[\Theta_{\text{prune}}^{\rho}]$ and $\sigma_{\Theta_{\text{prune}}^{\rho}}(t) = t[\Theta_{\text{prune}}] = t'[\Theta_{\text{prune}}^{\rho}]$. Due to the definition of the transitions T' of $\Theta_{\text{prune}}^{\rho}$, we also have $s[\Theta_{\text{prune}}] \xrightarrow{\ell} t[\Theta_{\text{prune}}] \in \Theta_{\text{prune}}^{\rho}$ which means $\sigma_{\Theta_{\text{prune}}^{\rho}}(s) \xrightarrow{\ell} \sigma_{\Theta_{\text{prune}}^{\rho}}(t) \in \Theta_{\text{prune}}^{\rho}$. For all other $\Theta' \in F'$ with $\Theta' \neq \Theta_{\text{prune}}^{\rho}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence we have $\sigma_{\Theta'}(s) \xrightarrow{\ell} \sigma_{\Theta'}(t) \in \Theta'$ because $\sigma_{\Theta'}$ is the projection onto Θ' and $\Theta' = \Theta$. Together, this implies that $\sigma_{\Theta'}(s) \xrightarrow{\ell} \sigma_{\Theta'}(t) \in \Theta'$ for all $\Theta' \in F'$, and hence by the definition of products, $\sigma(s) \xrightarrow{\ell} \sigma(t) \in \bigotimes F'$, which shows that τ satisfies $\mathbf{CONS}_{\mathbf{T}}^{S_{\text{sub}}^{\otimes}}$.

- Consider a transition $s' \xrightarrow{\ell} t' \in \bigotimes F'$. By the definition of products, $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta'$ for all $\Theta' \in F'$. Because there are no transitions from and to $\bot$, $s'[\Theta_{\text{prune}}^{\rho}] \neq \bot$ and $t'[\Theta_{\text{prune}}^{\rho}] \neq bot$, and thus $s'[\Theta_{\text{prune}}^{\rho}], t'[\Theta_{\text{prune}}^{\rho}] \in S_{\text{sub}}$. Hence, with our general observation on ρ, $s'[\Theta_{\text{prune}}^{\rho}]$ and $t'[\Theta_{\text{prune}}^{\rho}]$ are the only elements of the preimages of themselves, and thus also states of Θ_{prune}. Furthermore, since transitions are either exactly preserved or removed, the transition $s'[\Theta_{\text{prune}}^{\rho}] \xrightarrow{\ell} t'[\Theta_{\text{prune}}^{\rho}] \in \Theta_{\text{prune}}^{\rho}$ is induced by exactly one transition in Θ_{prune}, namely itself. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta_{\text{prune}}^{\rho}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta'] \xrightarrow{\ell} t'[\Theta'] \in \Theta$. Put together, with setting $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{prune}}, \Theta' \neq \Theta_{\text{prune}}^{\rho}, \Theta' = \Theta\} \cup \{\Theta_{\text{prune}} \mapsto s_{\Theta_{\text{prune}}}\}$ for the only element $s_{\Theta_{\text{prune}}} \in \rho^{-1}(s'[\Theta_{\text{prune}}^{\rho}])$, and analogously $t = \{\Theta \mapsto t'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{prune}}, \Theta' \neq \Theta_{\text{prune}}^{\rho}, \Theta' = \Theta\} \cup \{\Theta_{\text{prune}} \mapsto t_{\Theta_{\text{prune}}}\}$ for the only element $t_{\Theta_{\text{prune}}} \in \rho^{-1}(t'[\Theta_{\text{prune}}^{\rho}])$, we have that $s \xrightarrow{\ell} t \in \bigotimes F$ with $\sigma(s) = s'$ and $\sigma(t) = t'$, which shows both that τ satisfies $\mathbf{IND_T}$ and $\mathbf{REF_T}$.

- Consider a goal state $s \in S_{\text{sub}}$ of F. By the definition of products, $s[\Theta]$ is a goal state of

Θ for all $\Theta \in F$. For $\Theta^\rho_{\text{prune}}$, we have that $s[\Theta_{\text{prune}}] \in S_{\text{sub}}$ because $s \in S^{\bigotimes}_{\text{sub}}$, and hence, by the definition of $\sigma_{\Theta^\rho_{\text{prune}}}$ and the set of goal states of $\Theta^\rho_{\text{prune}}$, $\sigma_{\Theta^\rho_{\text{prune}}}(s) = s[\Theta_{\text{prune}}] = s'[\Theta^\rho_{\text{prune}}]$ is a goal state of $\Theta^\rho_{\text{prune}}$. For all other $\Theta' \in F'$ with $\Theta' \neq \Theta^\rho_{\text{prune}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence we have that $\sigma_{\Theta'}(s)$ is a goal state of Θ' because $\sigma_{\Theta'}$ is the projection onto Θ' and $\Theta' = \Theta$. Together, this implies that $\sigma_{\Theta'}(s)$ is a goal state of Θ' for all $\Theta' \in F'$, and hence by the definition of products, $\sigma(s)$ is a goal state of $\bigotimes F'$, which shows that τ satisfies $\mathbf{CONS}^{S^{\bigotimes}_{\text{sub}}}_{\mathbf{G}}$.

- Consider a goal state $s' \in \bigotimes F'$. By the definition of products, $s'[\Theta']$ is a goal state of Θ' for all $\Theta' \in F'$. Because $\bot$ is not a goal state, $s'[\Theta^\rho_{\text{prune}}] \neq \bot$ and thus $s'[\Theta^\rho_{\text{prune}}] \in S_{\text{sub}}$. Hence, with our general observation on ρ, $s'[\Theta^\rho_{\text{prune}}]$ is the only element of the preimage of itself, and thus also a state of Θ_{prune}. Furthermore, since goal states are either exactly preserved or mapped to $\bot$, $s'[\Theta^\rho_{\text{prune}}]$ is also a goal state of Θ_{prune}. For all other $\Theta' \in F'$, i.e. $\Theta' \neq \Theta^\rho_{\text{prune}}$, there exists exactly one $\Theta \in F$ with $\Theta = \Theta'$, and hence $s'[\Theta']$ is a goal state of Θ. Put together, with setting $s = \{\Theta \mapsto s'[\Theta'] \mid \Theta \in F, \Theta' \in F', \Theta \neq \Theta_{\text{prune}}, \Theta' \neq \Theta^\rho_{\text{prune}}, \Theta' = \Theta\} \cup \{\Theta_{\text{prune}} \mapsto s_{\Theta_{\text{prune}}}\}$, for the only element $s_{\Theta_{\text{prune}}} \in \rho^{-1}(s'[\Theta^\rho_{\text{prune}}])$, we have that s is a goal state of $\bigotimes F$ with $\sigma(s) = s'$, which shows both that τ satisfies $\mathbf{IND}_{\mathbf{G}}$ and $\mathbf{REF}_{\mathbf{G}}$.
- Finally, it is easy to see that τ_F satisfies $\mathbf{IND}_{\mathbf{L}}$, $\mathbf{CONS}_{\mathbf{C}}$, $\mathbf{IND}_{\mathbf{C}}$, and $\mathbf{REF}_{\mathbf{C}}$ because $\lambda = \text{id}$ and hence labels (and their cost) are not changed.

□

This theorem states that prune transformations are exact induced transformations for the subset of states that are not pruned. Intuitively, this makes sense, because the transformed (factored) transition systems is identical with respect to all states and transitions between states that are not pruned, and the single isolated pruned state is not a goal state and does not have any transitions (which is why it is refinable).

As mentioned above, in practice, we want to prune dead states, i.e. both unreachable and irrelevant states. Since a heuristic induced by a transformation is defined as the perfect heuristic of the transformed transition system, a heuristic induced by a prune transformation assigns states mapped to $\bot$ a heuristic value of ∞. This clearly violates admissibility for all states not evaluated as ∞ already before. In particular, unreachable (but relevant) states would usually have a finite heuristic value. We can still use pruning of unreachable states, though, because due to Theorem 3.5, we have that exact transformations for the reachable subset of states induce forward-exact heuristics. Thus, by setting $S_{\text{sub}} = R$ for the reachable states R in the above theorem, we obtain that prune transformations pruning unreachable states are exact on the reachable subset of states.

It is also easy to see that pruning irrelevant states can never change the heuristic induced by such a transformation: irrelevant states always have a heuristic value of ∞. Since in- and outgoing transitions of irrelevant states can never be part of any plan, removing their transitions also does not affect the heuristic. (Also recall that if we were interested in properties of heuristics relating to other subsets of states, Theorems 3.3 and 3.5 could directly be generalized to other such subsets, and the result of this section could still be combined with these results. We only

restricted these theorems to the reachable subset of states because pruning these states causes heuristics to not be admissible anymore for such states.)

To conclude this section, we have two remarks concerning the formalization of the prune transformation. The first concerns the implementation of the prune transformation in Fast Downward (Helmert, 2006) as it has been present since the introduction of merge-and-shrink for planning. The implementation differs from the conceptual definition in that it actually prunes states by removing them entirely from transition systems, not only removing their transitions. In the FMs, pruned entries are denoted by a special symbol $\perp$ like in our conceptual definition, but the tables of merge FMs do not store entries for the $\perp$ values of their component FMs, but implicitly map all pairs of values where one value is $\perp$ to $\perp$. While the final representation of the heuristic hence is slightly different, it is easy to see that the represented function is the same under the interpretation of $\perp$ as "pruned state". This implementation presumably is more efficient and closer to the intuitive concept of "pruning", but we cannot formalize this kind of pruning within our framework of transformations. The reason is that our definition of transformations require state mappings to be functions, and as such they must map *every* state to a state of the transformed transition system, hence not allowing states to "disappear" from the transition systems and FMs. We call this variant *original pruning*, since this is how pruning has been implemented since the first implementation of merge-and-shrink in Fast Downward.

Secondly, we remark that as another alternative to both the above formalization and our implementation, we could achieve pruning of dead states also by using an abstraction that maps all dead states to a single state but does not remove the transitions (otherwise, it would not be an abstraction). The result would semantically be equivalent because transitions from and to dead states cannot occur in any plans. Furthermore, due to the prune transformation then also being an abstraction like the shrink transformation, we would not need to consider properties for "subset of states" or "forward properties" of heuristics for any of the transformations. However, this approach would very likely induce an overhead associated with storing and computing the additional "dead transitions" that our current formalization removes, and the presence of these transitions would also influence all other transformations where the information of how to perform the transformation is based on labels and transitions. We call this variant *abstracting pruning*, and evaluate both alternatives of pruning in our experimental study in Section 6.7.

3.6. Generalized Label Reduction

The previous theory of label reduction could not be described as a general transformation of transition systems, due to only being applicable under certain conditions. With generalized label reduction, this is now possible, turning it into a transformation of the merge-and-shrink toolbox just like shrinking, merging and pruning. Some parts of this section, in particular definitions and proofs, stem from our original paper (Sievers, Wehrle, & Helmert, 2014).

For a transition system $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$, a label mapping λ defined on L, and a label cost function c' defined on $\lambda(L)$, the *transition system induced by Θ, λ, and c'* is defined as $\Theta^{\lambda,c'} = \langle S, \lambda(L), c', \{\langle s, \lambda(\ell), t\rangle \mid \langle s, \ell, t\rangle \in T\}, s_0, S_\star\}\rangle$.

Definition 3.18 (Label Reduction Transformation)**.** *Let F be a factored transition system with label set L and let λ be a label mapping defined on L. A* label reduction transformation *(*label

reduction *for short) is a factored transformation* $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ *of* F *into a factored transition system* F' *with label costs* c'*, where:*

- $F' = \{\Theta^{\lambda,c'} \mid \Theta \in F\}$ *with* $\Theta^{\lambda,c'}$ *the transition system induced by* Θ*,* λ*, and* c'*, which is defined as* $c'(\ell') := \min_{\ell \in \lambda^{-1}(\ell')}(c(\ell))$ *for all* $\ell' \in \lambda(L)$*.*
- $\Sigma = (\sigma_{\Theta'})_{\Theta' \in F'}$ *is an indexed collection of FMs, where each FM* $\sigma_{\Theta'}$ *is defined on* F *and associated with* $\Theta' \in F'$*:* $\sigma_{\Theta'} := \pi_{\Theta'}$ *for all* $\Theta' \in F'$*.*

Informally speaking, a label reduction applies a label mapping to the common label set of a factored transition system. The transformed factored transition system consists of the factors of the original factored transition system, however induced by the label mapping of the transformation. The cost of a mapped label is the smallest cost of all original labels that are mapped to the label. Note that states, including the initial states and the sets of goal states, are the same in the transformed factors, and the set of FMs of the transformation represents the identity state mapping accordingly.

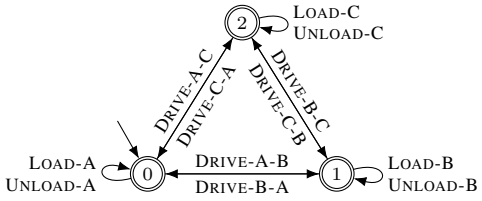
We call such a transformation a label *reduction* because we usually choose a label mapping λ that maps the labels of F to some smaller set. Hence, a label reduction potentially collapses parallel transitions of previously different labels to identical transitions under the new labeling, thus reducing the effective number of transitions in the transformed (factored) transition system. While we require that the cost of a mapped label ℓ' is the maximum possible label cost such that the cost is not larger than the cost of any original label mapped to ℓ', we could in principle also allow smaller label costs, but this could only decrease plan costs and with that, the quality of heuristics induced by label reduction transformations. Allowing label reductions to increase label costs would clearly not be cost-conservative.

Unlike the other merge-and-shrink transformations, composing label reductions with previous transformations is straightforward. Label mappings are non-factored functions that can directly be composed by using the regular composition of functions, and the set of FMs that represents the state mapping of the transformation is the identity state mapping. Hence, the set of FMs of a previous transformation can be entirely reused, without the need to compose this set of FMs with the set of FMs of the label reduction transformation.

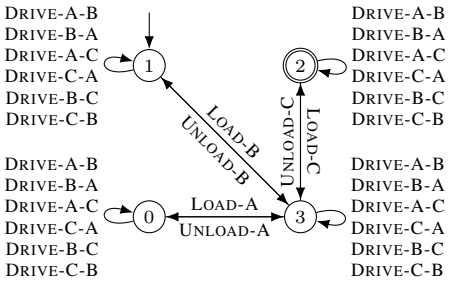
Consider the induced factored transition system of the example planning task of Figure 2.1. Figures 3.13a and 3.13b show the two atomic factors (not including the associated FMs because they are not affected by label reductions). Below, in Figures 3.13c and 3.13d, we see the label reduced factors after applying a label reduction τ that combines all labels that represent a "LOAD" operator into a new label x, all "UNLOAD" labels into y, and all "DRIVE" labels into z. (All new labels x, y, and z cost 1 since the example is a unit-cost case.) By the reduced number of labels labeling the transitions in the figure, we observe that the label reduced transition systems require far fewer transitions because many originally parallel transitions are collapsed into single transitions.

Before proceeding with showing under which conditions label reductions have which transformation properties, the following proposition formalizes that label reduction does not modify the (goal) states of the factored transition system.

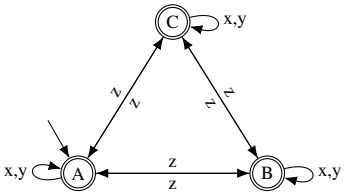
Proposition 3.2. *Let* F *and* F' *be factored transition systems, and let* τ_{F} *be a label reduction transformation of* F *into* F'*. Then* τ_{F} *satisfies* $\mathbf{CONS_G} + \mathbf{IND_G} + \mathbf{REF_G} + \mathbf{IND_S}$.



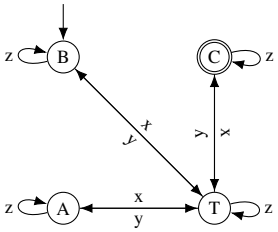
(a) Atomic factor $\Theta(v_T)$.



(b) Atomic factor $\Theta(v_P)$.



(c) Transformed factor $\tau(\Theta(v_T))$.



(d) Transformed factor $\tau(\Theta(v_P))$.

Figure 3.13.: Example of a label reduction τ applied to the induced factored transition system of the example planning task of Figure 2.1. FMs are not shown for simplicity. Top two figures: factors of the induced factored transition system. Bottom two figures: transformed factors after applying τ which combines all labels of type "LOAD" into label x, "UNLOAD" into y, and "DRIVE" into z.

We first show that any label reduction is a strict homomorphism.

Theorem 3.10. *Let F be a factored transition system. A label reduction transformation $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ of F into factored transition system F' is a strict homomorphism, i.e. satisfies* **CONS** $+$ **IND**.

Proof. Let $F = \{\Theta_1, \ldots, \Theta_n\}$ with labels L and label costs c, and let $F' = \{\Theta'_1, \ldots, \Theta'_n\}$ where each $\Theta'_i = \Theta_i^{\lambda, c'}$ is the factor induced by Θ_i, λ, and c', with $c'(\ell') := \min_{\ell \in \lambda^{-1}(\ell')}(c(\ell))$ for all $\ell' \in L'$. Let $\tau = \langle \bigotimes F', \sigma, \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}. In the following, we show that the induced transformation satisfies each of the claimed properties $\mathbf{CONS_T}$, $\mathbf{CONS_G}$, $\mathbf{CONS_C}$, $\mathbf{IND_S}$, $\mathbf{IND_L}$, $\mathbf{IND_T}$, $\mathbf{IND_G}$, and $\mathbf{IND_C}$. Since $\sigma = \mathrm{id}$, we simplify the notation by dropping the use of FMs for transformed states, and because we do not need to refer to FMs, we write states s of $\bigotimes F$ as tuples $\langle s_1, \ldots, s_n \rangle$ where each s_i is a state of factor Θ_i.[7]

- Because $L' = \lambda(L)$, λ is surjective, and thus τ satisfies $\mathbf{IND_L}$.
- Consider a transition $\langle s_1, \ldots, s_n \rangle \xrightarrow{\ell} \langle t_1, \ldots, t_n \rangle \in \bigotimes F$. By the definition of products, we have $s_i \xrightarrow{\ell} t_i \in \Theta_i$ for all $1 \leq i \leq n$, from which by the definition of the transformed factors being induced by λ and c' we get $s_i \xrightarrow{\lambda(\ell)} t_i \in \Theta'_i$ for all $1 \leq i \leq n$. Finally, again by definition of products, we have $\langle s_1, \ldots, s_n \rangle \xrightarrow{\lambda(\ell)} \langle t_1, \ldots, t_n \rangle \in \bigotimes F'$, which shows that τ satisfies $\mathbf{CONS_T}$.
- Consider a transition $\langle s_1, \ldots, s_n \rangle \xrightarrow{\ell'} \langle t_1, \ldots, t_n \rangle \in \bigotimes F'$. By the definition of products, we have $s_i \xrightarrow{\ell'} t_i \in \Theta'_i$ for all $1 \leq i \leq n$. Because each Θ'_i is induced by Θ_i, λ, and c', $s_i \xrightarrow{\ell'} t_i \in \Theta'_i$ implies that there exists $s_i \xrightarrow{\ell} t_i \in \Theta_i$ with $\ell \in \lambda^{-1}$ for all $1 \leq i \leq n$. Again by the definition of products, we get $\langle s_1, \ldots, s_n \rangle \xrightarrow{\ell} \langle t_1, \ldots, t_n \rangle \in \bigotimes F$ with $\ell \in \lambda^{-1}$, which shows that τ satisfies $\mathbf{IND_T}$.
- Consider $\ell \in L$. By the definition of label reductions, $c'(\lambda(\ell)) = \min_{\ell'' \in \lambda^{-1}(\lambda(\ell))} c(\ell)$, and hence $c'(\lambda(\ell)) \leq c(\ell)$, which shows that τ satisfies $\mathbf{CONS_C}$.
- Consider $\ell' \in L'$. By the definition of label reductions, $c'(\lambda(\ell)) = \min_{\ell'' \in \lambda^{-1}(\lambda(\ell))} c(\ell)$, and hence $c'(\ell') = c(\ell)$ for at least one label $\ell \in \lambda^{-1}(\ell')$, which shows that τ satisfies $\mathbf{IND_C}$.
- Furthermore, from Proposition 3.2, we also have that τ satisfies $\mathbf{CONS_G} + \mathbf{IND_G} + \mathbf{IND_S}$, which concludes the proof.

□

This theorem states that label reduction transformations are strict homomorphisms. Together with Theorem 3.3, an immediate consequence is that the heuristic for the factored transition system induced by a label reduction is admissible and consistent. If we did not require the label reduction to apply a surjective label mapping but an arbitrary one, the transformation would not be (label-)induced and thus only be a non-strict homomorphism. The same is true if we

[7] The same comment to the notation as in footnote 5 applies.

did not require that label costs of mapped labels are maximally high such that they are still not larger than any of the original labels, but simply allowed setting costs such that they are not increasing label costs. However, as discussed before, induced abstractions are in some sense the best abstractions we can have, and hence it is a positive result that label reductions based on surjective label mappings and the above definition of label costs are strict homomorphisms.

As we will see in the following, we can strengthen the conditions of label reductions such that they are even exact transformations. To do so, we first need some more terminology for labels of factored transition systems.

Definition 3.19 (Properties of Labels)**.** *Let F be a factored transition system with common label set L, and let $\ell, \ell' \in L$ be labels, and let $\Theta \in F$ be a transition system.*

- *Label ℓ is* alive *in F if all transition systems $\Theta' \in F$ have some transition $s \xrightarrow{\ell} t \in \Theta'$. Otherwise, ℓ is* dead*.*
- *Label ℓ* locally subsumes *label ℓ' in Θ if for all $s \xrightarrow{\ell'} t \in \Theta$ we also have $s \xrightarrow{\ell} t \in \Theta$. Label ℓ* globally subsumes *label ℓ' in F if ℓ locally subsumes ℓ' in all $\Theta' \in F$.*
- *Labels ℓ and ℓ' are* locally equivalent *in Θ if they label the same transitions in Θ, i.e. if ℓ locally subsumes ℓ' in Θ and vice versa.*
- *Labels ℓ and ℓ' are* Θ-combinable *in F if they are locally equivalent in all transition systems $\Theta' \in F \setminus \{\Theta\}$. (It does not matter whether or not they are locally equivalent in Θ.)*

Dead labels of F do not induce any transition in $\bigotimes F$, and hence removing such dead labels from L and their induced transitions from the factors of F clearly is an exact transformation. We also remark that all applications of label reductions can be expressed as chains of "minimal" label reductions which only combine two labels into a new one, leaving all other labels unchanged.

Theorem 3.11. *Let F be a factored transition system with label set L not containing dead labels and with label cost function c. Let λ be a label mapping defined on L with $\lambda(\ell_1) = \lambda(\ell_2) = \ell_{12}$ for $\ell_1, \ell_2 \in L$ and $\lambda(\ell) = \ell$ for all $\ell \in L \setminus \{\ell_1, \ell_2\}$. A label reduction $\tau_{\mathrm{F}} = \langle F', \Sigma, \lambda \rangle$ of F into factored transition system F' with label costs c' is an exact induced transformation, i.e. satisfies* **CONS** + **IND** + **REF***, iff $c'(\ell_{12}) = c(\ell_1) = c(\ell_2)$ and*

1. *ℓ_1 globally subsumes ℓ_2, or*
2. *ℓ_2 globally subsumes ℓ_1, or*
3. *ℓ_1 and ℓ_2 are Θ-combinable for some $\Theta \in F$.*

Proof. Let $F = \{\Theta_1, \ldots, \Theta_n\}$ and $F' = \{\Theta'_1, \ldots, \Theta'_n\}$ where $\Theta'_i = \Theta_i^{\lambda, c'}$ is the factor induced by Θ_i, λ, and c' for all $1 \leq i \leq n$. Let $\tau = \langle \bigotimes F', \sigma, \lambda \rangle$ be the transformation of $\bigotimes F$ into $\bigotimes F'$ induced by τ_{F}.

From Theorem 3.10, we already have that τ satisfies **CONS** + **IND**. From Proposition 3.2, we additionally get $\mathbf{REF_G}$. Hence the proof burden reduces to showing that τ satisfies $\mathbf{REF_T}$ +

$\mathbf{REF_C}$ iff $c'(\ell_{12}) = c(\ell_1) = c(\ell_2)$ and the conditions 1., 2., and 3. hold. Clearly, any label reduction satisfying $c'(\ell_{12}) = c(\ell_1) = c(\ell_2)$ is cost-refinable, i.e. satisfies $\mathbf{REF_C}$. To show that τ satisfies $\mathbf{REF_T}$ iff the conditions 1., 2., and 3. hold, we distinguish three cases:

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

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

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

We have to show that for any transition $s' \xrightarrow{\ell'} t' \in \bigotimes F'$, for all $s \in \sigma^{-1}(s')$, there exists $s \xrightarrow{\ell} t \in F$ with $t \in \sigma^{-1}(t')$ and $\ell \in \lambda^{-1}(\ell')$. Since $\sigma = \mathrm{id}$ and hence $|\sigma^{-1}(s')| = 1$, the universal quantification over $\sigma^{-1}(s')$ reduces to showing the claimed property for the single state $s \in \sigma^{-1}(s')$. Analogously, there is also only one state $t \in \sigma^{-1}(t')$, and thus we only have to show that for all transition $s' \xrightarrow{\ell'} t' \in \bigotimes F'$, there exists a transition $s \xrightarrow{\ell} t \in F$ with $\ell \in \lambda^{-1}(\ell')$, where $s = s'$ and $t = t'$ because σ is the identity label mapping.

From here on, as in the proof of Theorem 3.10, we simplify the notation by dropping any use of FMs when referring to transformed states, and the same comment regarding the notion of factored states as above applies.

On (A): We say that a transition system $\Theta \in F$ has an ℓ_1-*only* transition if there exists a transition $s \xrightarrow{\ell_1} t \in \Theta$ with $s \xrightarrow{\ell_2} t \notin \Theta$. Symmetrically, it has an ℓ_2-*only* transition if there exists a transition $s \xrightarrow{\ell_2} t \in \Theta$ with $s \xrightarrow{\ell_1} t \notin \Theta$.

We try to find two transition systems $\Theta_i, \Theta_j \in F$ with $i \neq j$ such that there is an ℓ_1-only transition $s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ and an ℓ_2-only transition $s_j \xrightarrow{\ell_2} t_j \in \Theta_j$. Then $\Theta_i \otimes \Theta_j$ does not contain a transition $\langle s_i, s_j\rangle \xrightarrow{\ell} \langle t_i, t_j\rangle$ for either $\ell = \ell_1$ or $\ell = \ell_2$, but $\Theta'_i \otimes \Theta'_j$ does contain the transition $\langle s_i, s_j\rangle \xrightarrow{\ell_{12}} \langle t_i, t_j\rangle$ because $s'_i \xrightarrow{\lambda(\ell)} t'_i \in \Theta'_i$ and $s'_j \xrightarrow{\lambda(\ell)} t'_j \in \Theta'_j$ for both choices of $\ell = \ell_1$ or $\ell = \ell_2$. By induction over the remaining transition systems, it is then easy to show that $\bigotimes F'$ contains a transition that does not correspond to a transition in $\bigotimes F$, proving that $\mathbf{REF_T}$ does not hold. (Here, we use that there are no dead labels: the argument fails if ℓ_1 and ℓ_2 are dead, because then also $\bigotimes F'$ would not contain a transition labeled with ℓ_{12}.) It remains to show that ℓ_1-only and ℓ_2-only transitions in different transition systems of F exist.

Because 1. does not hold, there exists an ℓ_2-only transition in some transition system $\Theta_i \in F$. Because 2. does not hold, there exists an ℓ_1-only transition in some transition system $\Theta_j \in F$. If Θ_i and Θ_j are different transition systems, we have found the required transitions and are done.

So let us assume that $\Theta_i = \Theta_j$. Because 3. does not hold, there exist at least two transition systems where ℓ_1 and ℓ_2 are not locally equivalent, so there is at least one transition system $\Theta_k \neq \Theta_i$ where they are not locally equivalent. This means that Θ_k must have an ℓ_1-only transition or an ℓ_2-only transition. In the former case, we select the ℓ_1-only transition in Θ_k and the ℓ_2-only transition in Θ_i. Otherwise, we select the ℓ_2-only transition in Θ_k and the ℓ_1-only transition in Θ_j $(= \Theta_i)$.

On (B): Consider Case 1., where ℓ_1 globally subsumes ℓ_2. Case 2. is identical with ℓ_1 and ℓ_2 swapped. For $\ell' \neq \ell_{12}$, the claim is trivial because $\bigotimes F$ and $\bigotimes F'$ are exactly identical regarding labels other than ℓ_1, ℓ_2 and ℓ_{12}. So consider the case $\ell' = \ell_{12}$. From $s \xrightarrow{\ell_{12}} t \in \bigotimes F'$, by the definition of products, we get $s_i \xrightarrow{\ell_{12}} t_i \in \Theta'_i$ for all $1 \leq i \leq n$, and hence $s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ or $s_i \xrightarrow{\ell_2} t_i \in \Theta_i$ for all $1 \leq i \leq n$ by the definition of label reductions. Because ℓ_1 globally

subsumes ℓ_2, this implies $s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ for all $1 \leq i \leq n$, and hence by the definition of products, $s \xrightarrow{\ell_1} t \in \bigotimes F$, concluding this part of the proof.

On (C): As in (B), we choose $\ell' = \ell_{12}$, and by the definition of products, we obtain that for all $1 \leq i \leq n$, $s_i \xrightarrow{\ell_{12}} t_i \in \Theta'_i$ and hence, by the definition of τ_{F}, $s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ or $s_i \xrightarrow{\ell_2} t_i \in \Theta_i$ for all $1 \leq i \leq n$. Choose $\ell \in \{\ell_1, \ell_2\}$ such that $s_j \xrightarrow{\ell} t_j \in \Theta_j$, where $j \in \{1, \ldots, n\}$ is chosen in such a way that ℓ_1 and ℓ_2 are Θ_j-combinable in F. (Such a transition system Θ_j exists because we are in Case 3.) By the definition of Θ-combinable, ℓ_1 and ℓ_2 are locally equivalent for all transition systems in F other than Θ_j, and hence ($s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ or $s_i \xrightarrow{\ell_2} t_i \in \Theta_i$) implies ($s_i \xrightarrow{\ell_1} t_i \in \Theta_i$ *and* $s_i \xrightarrow{\ell_2} t_i \in \Theta_i$) for all $i \neq j$. This shows that $s_i \xrightarrow{\ell} t_i \in \Theta_i$ for all $1 \leq i \leq n$, and hence $s \xrightarrow{l} t \in \bigotimes F$, concluding the final part of the proof. □

Together with Theorem 3.5, an immediate consequence is that the heuristic for the factored transition system induced by a label reduction based on Θ-combinability that preserves label costs is perfect. We have already seen such a label reduction based on Θ-combinability: in Figure 3.13, all labels of type "DRIVE" are $\Theta(v_T)$-combinable because these labels are locally equivalent in the atomic factor $\Theta(v_P)$. Hence combining them into a new label z is an exact transformation (if not changing the other labels as done in the example, because these label reductions do not satisfy any of the exactness conditions), preserving plans and their costs.

This concludes the theoretical presentation of label reduction transformations, and we refer to Section 3.8 for a discussion of generalized label reduction, including a comparison to the previous theory of label reduction, an algorithm to compute exact label reductions based on Θ-combinability, and a description of efficiency improvements possible in the merge-and-shrink implementation based on generalized label reduction.

3.7. Algorithm

In this section, we describe a concrete instantiation of the merge-and-shrink algorithm in full detail, sticking to the particular instantiation as implemented in Fast Downward (Helmert, 2006). To begin, recall the general outline of the merge-and-shrink algorithm as discussed and illustrated in Algorithm 2 on page 47. A concrete implementation needs to decide which transformation to apply in each iteration of the main loop. Also note that the number of factors can only be reduced through merge transformations, hence the algorithm performs exactly $|F| - 1$ many merge transformations before terminating.[8] The layout of the algorithm we consider here changes the loop to apply exactly one merge transformation and an arbitrary number of other non-merge transformations in each iteration.

Algorithm 3 shows pseudocode. For a given factored transition system, the algorithm first stores a copy F' of it, computes the set Σ of associated FMs that represent the identity state mapping on $\bigotimes F'$, and sets λ to the identity label mapping on the labels of F' (line 2), just like in the algorithm discussed previously. As another preprocessing step before the main-loop,

[8] However, if a factor of F is shown to be unsolvable already during the computation of the algorithm, there are two alternatives of handling such cases: continue until $|F| = 1$, which can cause the computation not to finish under given resource limits, or to terminate the computation early on, returning a trivially unsolvable transition system with a single state together with a state mapping that maps all states to that single state. Practical implementations typically opt for the second choice.

Algorithm 3 The merge-and-shrink algorithm as implemented in Fast Downward.

Input: Factored transition system F, merge strategy MS, shrink strategy SS, prune strategy PS, label reduction strategy LRS, size limit $N \in \mathbb{N}$.

Output: Transition system Θ and FM σ mapping from states of $\bigotimes F$ to states of Θ.

function MERGEANDSHRINK(F)
  ▷ *Copy input factored transition system, compute Σ to represent the identity state mapping on $\bigotimes F'$, set λ to the identity label mapping.*
  $\langle F', \Sigma, \lambda \rangle \leftarrow \langle F, \{\pi_\Theta \mid \Theta \in F'\}, \mathrm{id} \rangle$
  for $\Theta \in F$ **do**
    ▷ *Prune atomic factor Θ with PS.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(PRUNE(F', Θ))
  end for
  while $|F'| > 1$ **do**
    ▷ *With MS, select two factors from F to be merged in this iteration.*
    $\Theta_1, \Theta_2, \leftarrow$ SELECT(F')
    ▷ *With LRS, apply a label reduction to F.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(LABELREDUCTION(F'))
    ▷ *With SS, shrink Θ_1 and Θ_2 so that the size of their product respects N.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(SHRINK($F', \Theta_1, \Theta_2, N$))
    ▷ *With LRS, apply a label reduction to F.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(LABELREDUCTION(F'))
    ▷ *Apply the merge transformation.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(MERGE(F', Θ_1, Θ_2))
    ▷ *With PS, prune the product factor $\Theta^\otimes$ of Θ_1 and Θ_2.*
    $\langle F', \Sigma, \lambda \rangle \leftarrow$ COMPOSETRANSFORMATION(PRUNE($F', \Theta^\otimes$))
  end while
  return single elements $\Theta \in F$ and $\sigma \in \Sigma$
end function

it also applies a prune transformation to all atomic factors of F' according to the prune strategy PS (line 4). Here and in the following, the method COMPOSETRANSFORMATION is to be understood to take the transformation as computed by the method given as its parameter (i.e. PRUNE, LABELREDUCTION, SHRINK, or MERGE) and to compose it with the current transformation $\langle F', \Sigma, \lambda \rangle$. While we do not explicate the composition further, it can be computed efficiently, in particular with respect to the FMs, as described in the individual sections on each transformation.

The main loop is then laid out as follows. In a first step, with the merge strategy MS, the algorithm selects the two factors to be merged in the current iteration (line 7). It then possibly applies a label reduction to the set of labels using the label reduction strategy LRS (line 8). Afterwards, the algorithm possibly shrinks the selected two factors, using the shrink strategy SS, so that their product respects the specified size limit N (line 9). Then, the algorithm applies a label reduction again (line 10). Finally, the selected merge can be performed, i.e. the merge transformation with the two chosen factors is applied (line 11). Before terminating the current iteration, though, the algorithm prunes the product factor of the merge, using the prune strategy PS (line 12).

With the theoretical results of the previous sections, we know that the final transition system $\Theta \in F'$ is an induced abstraction of $\bigotimes F$, possibly only for the subset of reachable states, and the only element σ left in Σ is an FM associated with Θ representing the corresponding abstraction mapping from states of $\bigotimes F$ to states of Θ. Furthermore, we know that the heuristic $h^{\tau}_{\bigotimes F}$ induced by the transformation $\langle F', \Sigma, \lambda \rangle$ is a (forward-)admissible heuristic for $\bigotimes F$. If all applied transformations are exact (for the subset of reachable states), then $h^{\tau}_{\bigotimes F}$ is even (forward-)exact.

The algorithm has several parameters. The obvious ones are the transformation strategies SS, MS, PS, and LRS, which we discuss in more detail below, as well as the size limit N. Besides these, there are also a few hidden parameters. The two places where the algorithm performs label reductions are considered optional, i.e. the algorithm can apply no label reduction at all, only before shrinking, only before merging, or twice in each iteration. The reason to have it twice in the main loop is that there exist different shrink strategies, some of which profit from operating on label reduced transition systems, while others do not. Furthermore, while shrinking could also be applied on the product, i.e. after merging, this would require to first compute the full product which potentially could violate the given size limit N on transition systems. Hence we shrink before merging because then the given size limit is always respected. A consequence of this decision is that we also have to decide which of the two factors to shrink to what extent. In practice, we choose balanced size limits. That means we shrink both transition systems to the same size such that their product respects the given size limit N, unless it suffices to shrink only one of the factors. In that case, the factor is shrunk as much as needed such that the product stays the limit N. The algorithm also allows performing shrink transformations even if the size limit N would not be violated by the product to allow shrink strategies to compute exact shrink transformations that actually reduce the size of a transition system.

We now discuss the concept of transformation strategies. A *shrink strategy* gets a single factor and a target size as input. It then computes a shrink transformation which reduces the number of states of the factor so that it is at most the target size. Shrinking can be understood as performing a sequence of minimal transformations that combine exactly two states of a transition system.

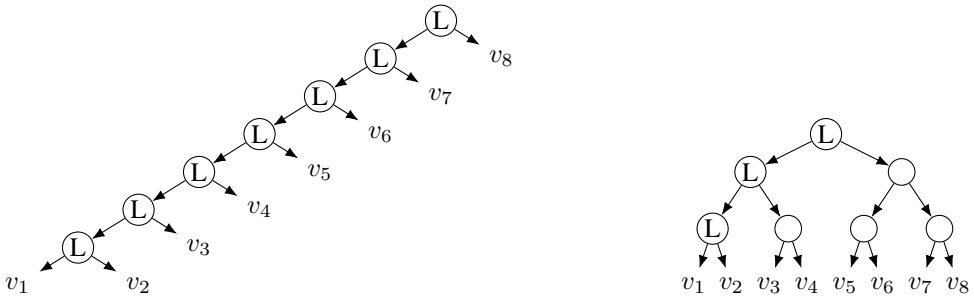


Figure 3.14.: Two merge trees for a problem with 8 finite-domain variables. Previous theory allows reducing labels in the intermediate abstractions marked with an "L" when v_1 is the pivot.

This allows shrink strategies to be described as the simple choice of selecting two states of a transition system to combine. We discuss shrink strategies from the literature in Chapter 5.

More important for this thesis is the concept of a *merge strategy*. Given a factored transition system, it needs to choose two factors that should be merged in the current iteration of the algorithm. Given that a merge strategy thus performs a sequence of decisions selecting two elements of a set, it can be viewed as a binary tree, the so-called *merge tree*, over the factors of the original factored transition system, which form the leaves of the tree. All inner nodes correspond to factors that arise from merging the factors corresponding to their child nodes, and hence whenever two nodes have the same ancestor, this means that the two corresponding factors must be merged at some point during the merge-and-shrink computation.[9]

For example, consider a state space defined through 8 finite-domain variables $v_1, \ldots, v_8$. The induced factored transition system hence consists of 8 atomic factors. Figure 3.14 shows two possible merge trees for this state space. Ignore the labels of the nodes for the moment. The merge tree shown on the left dictates to first merge the two atomic factors $\Theta(v_1)$ and $\Theta(v_2)$, then their product with the atomic factor $\Theta(v_3)$, and so on. However, a merge tree does not always dictate a fully defined order in which the next pair of transition systems is selected, but sometimes only a partial order: in the right tree, for example, we are free to first merge $\Theta(v_1)$ with $\Theta(v_2)$ and then $\Theta(v_3)$ with $\Theta(v_4)$ before merging the two resulting product transition systems, or the other way around. Assuming a fixed order in which leaf nodes are considered solves this ambiguity.

We also distinguish two types of merge trees (and with that, merge strategies): the merge tree on the left degenerates to a list, i.e. every right child node is always a leaf node, and for that reason is called a *linear* merge tree, representing a linear merge strategy. The merge tree on the right is not a list and is called *non-linear*, representing a non-linear merge strategy. Like for shrink strategies, the merge-and-shrink literature describes various merge strategies, most of which we discuss in Chapter 5.

The third transformation strategy is the *prune strategy*. Given a single factor, it must decide which states to prune. As we have seen in Section 3.5, pruning of unreachable states results in

[9]Note that the trees underlying FMs have the same structure, i.e. in that sense, a merge tree basically is an FM without value sets and table functions.

the heuristic being forward-exact. Furthermore, pruning of irrelevant states never hurts heuristic quality. Thus, for simplicity, we consider only two binary choices for pruning strategies: to prune or not to prune unreachable and irrelevant states, which amounts to a total of four combinations. We call the option that prunes both unreachable and irrelevant states *full pruning*, which has been the default in the merge-and-shrink implementation since its addition to Fast Downward.

Finally, a *label reduction strategy* needs to decide which labels to combine, given a factored transition system and its implied set of labels and label costs. Analogously to shrink strategies, we can describe label reductions as a sequence of minimal label reductions that combine exactly two labels. In Section 3.6, we have seen that label reductions are always strict homomorphisms and can also be exact induced transformations. In the next section, we describe an algorithm to compute exact label reductions. We consider different instantiations of this algorithm as label reduction strategies.

3.8. Discussion of Generalized Label Reduction

In this section, we discuss the consequences of the introduction of generalized label reduction to the merge-and-shrink framework. In particular, we compare it to the previous theory of label reduction, showing the advantages of the new theory. Furthermore, we provide an algorithm to compute exact label reductions based on Θ-combinable labels. Finally, we explain how we can make the the implementation of the merge-and-shrink algorithm computationally more efficient compared to previous implementations due to detaching labels from their meaning as operators and due to exploiting equivalence classes of combinable labels.

3.8.1. Comparison to Previous Label Reduction

The following discussion, to some extent, stems from our original paper introducing generalized label reduction (Sievers et al., 2014). Having defined generalized label reduction in Section 3.6, we can now better understand the weaknesses of the previous theory of label reduction. We describe these weaknesses without formally introducing the previous theory.

Firstly, the old theory largely attempts to define label reduction as a local concept considering individual transition systems: the central notion is a label-reduced transition system. This is fundamentally at odds with the purpose of labels in the merge-and-shrink framework to coordinate the joint behavior of *all* transition systems in the factored transition system F maintained during the computation. If we change the labels in some, but not all factors of F, synchronization cannot work correctly.

The earlier papers address this difficulty by performing a kind of "just-in-time label reduction" that makes the labels of two transition systems correspond just before they are merged (which is the only point at which labels matter). This works, but the resulting theory is complex to understand and reason about, as different parts of the merge tree work with different labels. Consequently, the previous theory only permits reducing labels in certain cases, with other cases deemed to be unsafe and hence forbidden. Complications mainly arise in the case of non-linear merge strategies, and consequently, these were never correctly implemented.

To better understand this restriction to linear merge strategies, consider again the state space

defined through 8 finite-domain variables $v_1, \ldots, v_8$ and the two possible merge trees shown in Figure 3.14 on page 72. To use the previous label reduction, one must select a leaf node as a *pivot* element, and may only apply label reduction in transition systems that are ancestors of the pivot or the pivot itself. In the example, we choose v_1 as the pivot. All nodes marked with an "L" are ancestors of the pivot and hence the factors corresponding to these nodes (and the leaf v_1 itself) can be modified through label reduction. This means that for the linear merge tree on the left, all intermediate factors and the atomic one for v_1 can be label reduced using the previous label reduction. On the other hand, for complete merge trees over n variables such as the merge tree on the right, only $O(\log n)$ of the factors can be label reduced using the previous label reduction, a restriction strong enough to have limited all prior work to use linear merge strategies.

The second weakness of the previous theory of label reduction is that it is in a certain sense syntax-based: it needs to "look inside" the labels in order to decide which labels can be combined into one. For planning tasks, label reduction must treat labels as structured pairs of preconditions and effects, reintroducing and critically depending on the syntactic descriptions. This contrasts merging and shrinking, which also in the previous description of merge-and-shrink are considered purely semantic operations.

Thirdly, the previous theory cannot exploit label reductions that are enabled by shrinking. The decision how to reduce labels is completely independent of the shrink steps of the algorithm and hence needs to be correct for *all possible* shrink strategies. This severely limits simplification possibilities.

With generalized label reduction, all of these restrictions are taken away. We think that the new theory is much easier to understand than the previous one. Furthermore, it can be used at every intermediate transition system also with non-linear merge strategies, reducing labels in all $2n - 1$ factors of a merge tree with n variables. Like the other transformations, it is purely semantic and treats labels as opaque tokens that do not need to "stand for" anything, which allows interleaving it with arbitrary merge, shrink, and prune transformations. Finally, it is also more powerful than the previous theory because it allows reducing more labels. For example, consider an example planning problem with three variables v_1, v_2, v_3 and recall that we need to use a linear merge strategy. Consider further two labels ℓ_1 and ℓ_2 corresponding to planning operators o_1 and o_2 that have different preconditions on v_1, i.e. $pre(o_1)[v_1] \neq pre(o_2)[v_1]$, and have the same preconditions and effects on all other variables. With the previous theory of label reduction, these two labels could only be combined in the single maintained product transition system Θ (which serves as the pivot in the sense that only the product transition system can be label-reduced) iff the factor $\Theta(v_1)$ representing variable v_1 has been merged into Θ, because then the preconditions and effects of the two labels would agree on all other variables "not merged yet". With our theory, ℓ_1 and ℓ_2 are $\Theta(v_1)$-combinable from the beginning on and hence can immediately be combined into a new label.

The new theory has already been proven useful in several papers (discussed in Section 5) that directly make use of generalized label reduction or non-linear merge-and-shrink. Furthermore, the interpretation of labels as opaque tokens, without attached preconditions and effects, allows for a more memory- and time-efficient implementation of the whole merge-and-shrink framework in the Fast Downward planning system, which we discuss below and evaluate in Section 6.4.

Figure 3.15.: A factored transition system with two transition systems Θ_1 (left) and Θ_2 (right).

3.8.2. Computation of Exact Label Reduction

We now describe how to compute label reductions based on Θ-combinability. Let us recall the conditions of Theorem 3.11 under which two labels ℓ_1 and ℓ_2 can be combined without losing information: either one globally subsumes the other, or ℓ_1 and ℓ_2 are Θ-combinable for some transition system Θ of a factored transition system. Concerning the computation of globally subsumed labels, we note that this involves finding subset relationships in a set family, for which to the best of our knowledge no linear-time algorithms are known. However, as demonstrated with the general example comparing linear to non-linear merge strategies in the previous section, already with only using Θ-combinable labels, we can reduce more labels than with the previous label reduction, and hence we restrict exact label reductions to this condition.

Before describing the computation of exact label reductions based on Θ-combinable labels (combinable labels for short in the following), we make the following important observation. If ℓ_1 and ℓ_2 are combinable and ℓ_2 and ℓ_3 are combinable for different transition systems, this does *not* imply that ℓ_1 and ℓ_3 are combinable for any transition system, i.e. *combinable* is *not* a transitive relation. More specifically, the relation on locally equivalent labels for a single transition system *is* transitive. It is even an equivalence relation, called *local equivalence relation* (on labels) in the following. However, the combinable relation is the union over these local equivalence relations, and this union relation is not transitive. An attempt at an intuitive explanation: *after* merging ℓ_1 and ℓ_2, the fresh label that replaces ℓ_1 and ℓ_2 is in general not locally equivalent with ℓ_3 in all (but one) component transition systems because we now have transitions with the new label that originated from ℓ_1 and have no matching transition for ℓ_3.

An immediate consequence of this observation is that if we want to apply all possible label reductions by computing combinable labels for all transition systems until there are no more such combinable labels, the result depends on the order in which we consider the transition systems. As an example, consider the factored transition systems shown in Figure 3.15. It has two transition systems Θ_1 (left) and Θ_2 (right), with states s_1, t_1, and s_2, t_2. If we combine ℓ_1 and ℓ_2 into a new label ℓ because they are Θ_1-combinable, then ℓ and ℓ_3 are not combinable afterwards for any of the two transition systems. On the other hand, if we combine ℓ_2 and ℓ_3 into a new label ℓ because they are Θ_2-combinable, we cannot combine ℓ with ℓ_1 afterwards.

With the above observations, our algorithm of choice to compute exact label reductions based on Θ-combinability is the fixed point algorithm illustrated in Algorithm 4. It keeps track of the number of iterations in which no more labels could be combined (line 2) and reaches a fixed point if the number of such unsuccessful iterations equals the number of factors in the given factored transition system F, i.e. if for no (more) factor $\Theta \in F$, we can combine labels based on Θ-combinability (line 3). In each iteration, the algorithm chooses the next factor

Algorithm 4 Fixed point label reduction algorithm.

Input: Factored transition system F with labels L and label costs c; Order O on factors of F.
Output: Transformed factored transition system F.

1: **function** LABELREDUCTION(F)
    ▷ *Number of unsuccessful iterations.*
2: `#-unsucc-it` $\leftarrow 0$
3: **while** `#-unsucc-it` $< |F|$ **do**
        ▷ *Get next transition system according to the order O of transition systems on F.*
4: $\Theta \leftarrow$ NEXT(F', O)
5: `equiv-rel` $\leftarrow \Theta$-COMBINABLE-RELATION(F, Θ))
6: **if** `equiv-rel` is trivial **then**
            ▷ *There are no Θ-combinable labels.*
7: `#-unsucc-it` $\leftarrow$ `#-unsucc-it` $+ 1$
8: **else**
            ▷ *Compute label mapping $\lambda : L \mapsto L'$ and new label cost function c' from equivalence relation over L.*
9: $\lambda, c' \leftarrow$ LABELMAPPING(L, `equiv-rel`)
            ▷ *Apply label mapping λ and cost function c' to F, i.e. replace L by L' and c by c'.*
10: APPLY(F, λ, c')
11: `#-unsucc-it` $\leftarrow 0$
12: **end if**
13: **end while**
14: **return** F
15: **end function**
16: **function** Θ-COMBINABLERELATION(F, $\Theta \in F$)
    ▷ *Universal equivalence relation over L: single equivalence class containing all $\ell \in L$*
17: `equiv-rel` $\leftarrow$ UNIVEQUIVREL(L)
18: **for** $\Theta' \in F$ with $\Theta' \neq \Theta$ **do**
        ▷ *Refine the current equivalence relation over L with the local equivalence relation over L of Θ'.*
19: `equiv-rel` $\leftarrow$ REFINE(`equiv-rel`, EQUIVREL(Θ', L))
20: **end for**
21: **return** `equiv-rel`
22: **end function**

$\Theta \in F$ according to some user-specified *order* O in which factors of F should be considered (line 4). For the chosen factor, the algorithm computes the Θ-combinable equivalence relation on labels L of F using the method Θ-COMBINABLERELATION which works as follows. Starting with the universal equivalence relation on L in which all labels are considered equal (line 17), the algorithm then subsequently refines this equivalence relation through the local equivalence relation on L of all factors other than Θ (line 19). (For this iterative refinement, the order in which we consider the factors of F does *not* matter.)

If this Θ-combinable equivalence relation over L is trivial in the sense that there are no two labels considered equal under the equivalence relation, the iteration counts as unsuccessful (line 7) and the algorithm repeats or terminates, depending on the count of unsuccessful iterations (line 3). Otherwise, the algorithm turns the Θ-combinable equivalence relation into a label mapping λ (line 9) by combining all labels within a single equivalence class to a new label ℓ, setting the cost $c'(\ell)$ of the new label to the minimum cost of all labels mapped to ℓ by λ to satisfy the definition of label reductions (cf. Definition 3.18). Note that to obtain an exact label reduction, we have to split each equivalence class within the computed equivalence relation further according to label costs. Applying the label mapping to the factored transition system (line 10) means to re-label all transitions labeled by labels that have been combined into new labels. Finally, the algorithm continues with the next iteration.

A first remark concerns the interaction of the algorithm with the merge-and-shrink algorithm. Recall that Algorithm 3 provides two *single* calls to the method LABELREDUCTION which corresponds to the label reduction algorithm presented here. However, due to the fixed point nature of the label reduction algorithm, it does not compute a single label reduction but rather applies a sequence of label reductions. Hence each call to APPLY(F, λ, c') in above algorithm is to be understood as composing the transformation based on the label mapping computed in the current iteration of the algorithm with the transformation maintained by the merge-and-shrink algorithm.

Secondly, we remark that the computation of the local equivalence relations over the set of labels L for a factor Θ (EQUIVREL(Θ, L)) is possible in time polynomial in the size of the factor. Refining an equivalence relation A through another one called B (REFINE(A, B)) is possible in time linear in the number of elements of the equivalence relations (here: the number of labels) when using suitable data structures such as linked lists.

Assuming that the fixed point algorithm, which we call *FP* in the following and in the experimental study, is a somewhat expensive algorithm in terms of runtime, we also consider two cheaper variants of computing exact label reductions. An easy possibility is to not iterate until reaching a fixed point, but to stop after having computed combinable labels for all factors once. For this variant, called *ONCE*, the above algorithm never resets the *#-unsucc-it* counter. An even cheaper variant only computes combinable labels for the two factors that will be merged next, with the rationale that we want to combine labels that are not locally equivalent in these two factors, which renders these factors simpler in that more transitions are induced by the same labels, hence reducing the required runtime to compute their product. In the algorithm, this can be achieved by not using the loop of LABELREDUCTION at all, but only performing two manual "iterations" using Θ-COMBINABLE-RELATION for each of the to-be-merged factors. This variant is called *2TS* (consider only two transition systems).

As observed before, the order in which the algorithm considers the factors Θ to compute Θ-

combinable labels matters (cf. the method NEXT(F)) if using FP or ONCE. In the experiments reported in our previous work (Sievers et al., 2014), and to the best of our knowledge also in all other work making use of generalized label reduction with FP, factors are considered in a randomized order, fixed a priori. In our experiments for this work, we also evaluated two alternative orders with FP, based on the variable order of Fast Downward. The variant *RL* considers atomic factors according to the regular variable order (called reverse level) and then all non-atomic ones in the order in which they originated. A third variant inverts the order of RL and is called *L*. However, we found that the order does not matter a lot in the practice of the IPC benchmarks: coverage of FP with RND, RL, and FP only varied marginally. We conclude that there are not many cases where one has to decide between reducing combinable labels ℓ_1 and ℓ_2 or labels ℓ_1 and ℓ_3. We hence stick with the commonly used randomized order, both when using FP and ONCE.

3.8.3. Efficient Implementation

In the following, we describe the techniques used for an efficient implementation of the merge-and-shrink framework in Fast Downward. Some of these techniques have already been used in the older implementation using the previous theory of label reduction (cf. Section 4.3 by Helmert et al., 2014) and in our first, non-optimized implementation of generalized label reduction in Fast Downward, used in our original conference paper (Sievers et al., 2014). All improvements based on generalized label reduction are implemented in the state-of-the-art optimized code base of merge-and-shrink. Hence there are two implementations we compare here, and we describe both of them in the following, distinguishing clearly between optimizations present in the previous implementation and in the optimized implementation.

In both implementations, we do not store transitions as an adjacency list as it is common with graphs, but we store all transitions grouped by labels. This allows an efficient application of all transformations of transition systems, as we will see below. Additionally, and in contrast to the previous implementation, we store *label groups* of locally equivalent labels (i.e. we store the local equivalence relation on labels for each transition system), disregarding their cost (the cost of a label group is the minimum cost of any participating label) and store their transitions only once rather than separately for each label. This was not possible in the previous implementation where labels were associated with preconditions and effects of operators. In addition, due to this more memory-efficient representation, we can also represent so-called *irrelevant* labels,[10] i.e. have one group that represents irrelevant labels. The previous implementation did not store transitions of irrelevant labels explicitly, which led to various special-casing.

Furthermore, we maintain a *valid-state* invariant for transition systems, which states that labels and their transitions are grouped (i.e. local equivalence relations on labels have been computed), that the transitions are sorted and unique within label groups (which eases the computation of locally equivalent labels, because this requires comparing sets of transitions), that g- and/or h-values for each transition system are computed, depending on the requirements on distance information of the chosen merge, shrink, and prune strategy, and that all factors are

[10] A label l is irrelevant in Θ if for all states s of Θ, there is exactly one transition $s \xrightarrow{l} s \in \Theta$, i.e. the label induces self-looping transitions for all states. A label l is relevant if it is not irrelevant. See also Definition 5.5 on page 109.

pruned according to the prune strategy.[11] We make sure that after applying any transformation to the factored transition system, all transition systems are in a valid state again. This contrasts the previous implementation which also guaranteed that transitions were sorted and unique (for single labels), but did not enforce this as an invariant, but only on demand (before shrinking and merging, but not after merging), and which always computed both g- and h-value distance information (because it always used full pruning).

Another difference is due to the changed layout of the main loop: we apply prune transformations according the prune strategy *PS* after merging, because only when computing the product system, new dead states can arise. The previous implementation performed pruning while shrinking, and hence only pruned transition systems before they were further processed.

In both implementations, we compute distance information of transition systems by performing Dijkstra's algorithm (Dijkstra, 1959). This is the only place where we need an explicit adjacency list representation of transition systems.

We now turn our attention to the different merge-and-shrink transformations. When applying a shrink transformation to the factored transition system, the shrink strategy computes a state mapping in the form of an equivalence relation on states. We first compute the explicit state mapping from this equivalence relation, assigning a consecutive number to each equivalence class. This state mapping can then be applied to both the transition system with its representation of transitions grouped by labels and to the FM. From the equivalence relation on states, we update the information on goal states. This implementation of shrinking is identical to the previous implementation by Helmert et al.

When applying a merge transformation to the factored transition system, merging two transition systems Θ_1 and Θ_2, we do not compute the full product of states and their transitions as it has been done in the previous implementation by Helmert et al., because this would require to compute the local equivalence relation on labels from scratch after computing the product. Instead, we use a bucket-based approach to directly compute the refinement of the local equivalence relations on labels of Θ_1 and Θ_2, collecting their transitions accordingly. To compute the merge FM, all that needs to be done is to set its table to map pairs of component states to their product state.

When applying a prune transformation, we first need to compute the set of unreachable and/or irrelevant states for the to-be-pruned transition system, depending on the chosen options for pruning. As the required distance information is always up-to-date for all transition systems (valid-state invariant), we simply collect all states whose g- and/or h-value is infinity and either prune them by entirely removing them and their transitions from the transition system (original pruning), or by applying an abstraction that maps all unreachable/irrelevant states to two single states (pruning as abstraction), as described at the end of Section 3.5. The FM is updated by applying a state mapping that maps all pruned states to $\perp$ and leaves all other entries unchanged. The former implementation of pruning (removing states and transitions) is identical to the previous implementation by Helmert et al.

Computing a label reduction with the algorithm shown on page 76 is also favored by our rep-

[11] We remark that the layout of the algorithm as discussed above is chosen to accommodate the needs of our invariant. In particular, concerning pruning, we prune all atomic factors once, and since new pruning opportunities can only arise in product factors, we also prune immediately after merging.

resentation that stores the local equivalence relations on labels for all transition systems and all transitions grouped by labels. In particular, the loop of the method Θ-COMBINABLE-RELATION simply uses the already computed local equivalence relations on labels (EQUIVREL) rather than computing it from scratch (valid-state invariant). For efficient REFINE operations as discussed in the description of the algorithm, we store labels in linked lists. Computing the label mapping (LABELMAPPING) is straight-forward and is done as computing the state mapping from the equivalence relation on states when shrinking. As we store locally equivalent labels of different costs in the same group, for exact label reductions, we need to further split the computed label mapping according to label costs.

Applying the label mapping (APPLY) can be efficiently done as follows. When updating a transition system for which we know that only locally equivalent labels are combined (which is the case for all transition systems other than Θ when applying a label mapping based on Θ-combinability), all we have to do is to remove the old labels from their group (they are all in the same group) and adding the new label to it; the transitions remain. Otherwise, we need to remove the old labels from their groups (potentially different ones) and add a new singleton group for the new label. While doing so, we collect the transitions of all to be removed labels and combine them to form the transitions of the new label. If label groups become empty, we remove them together with their transitions. We also need to recompute the costs of all modified label groups. Finally, we recompute the local equivalence relation on labels to restore the valid-state invariant for the transitions system.

In contrast to the previous implementation by Helmert et al., we directly encode the heuristic values into the final FM σ rather than keeping σ together with the distance information of the final transition system Θ. This means that the FM does not store the abstraction mapping from $\Theta(\Pi)$ to Θ but represents the heuristic function, i.e. a mapping from states s of $\Theta(\Pi)$ to $h_\Theta(s)$.

We remark that the above improvements compared to the previous implementation also affect some of the transformation strategies of the merge-and-shrink toolbox. We come back to that impact when discussing these strategies in the next chapter.

To conclude this section, the following list summarizes the differences of the optimized implementation compared to the previous one. All of these changes, except the last point of the list, are only possible due to the addition of generalized label reduction.

- We store labels and their transitions grouped according to the equivalence relation on labels within each transition system, including representing irrelevant labels.
- We maintain a valid-state invariant of transition systems. This means that we always store groups of locally equivalent labels, keep their transitions sorted and unique, and have distance information computed as required by all transformation strategies.
- We compute product systems without computing the full product of all transitions but rather by directly computing the label groups and their transitions as a refinement of the local equivalence relations on labels of the two components.
- We encode the heuristic directly into the final FM.

In our experimental study, we will compare merge-and-shrink based on generalized label reduction using the previous and the optimized implementation.

4. Expressiveness of Merge-and-Shrink

Throughout this thesis, we use FMs to store abstraction mappings or heuristic functions as part of the merge-and-shrink framework. However, it is also useful to consider FMs as a general mechanism for representing functions mapping variable assignments into some set of values, independently of their concrete use in state-space search (where the variable assignments are states, and they are mapped to numerical heuristic values). In this chapter, we consider FMs as such general-purpose data structures and discuss their *expressive power*. More precisely, as it is clear which functions FMs can represent, the more interesting question we answer here is which functions can be *compactly* represented with FMs. The results then directly transfer to merge-and-shrink heuristics, which are represented using FMs.

The existing theoretical studies of the expressive power of FMs are limited to the *linear* case, partly because the existing theory did not allow for efficient implementations of general non-linear abstractions until the introduction of generalized label reduction. It is well-known that linear FMs correspond to *binary* (Bryant, 1985) and *algebraic* (Bahar et al., 1993) *decision diagrams* (*BDDs* and *ADDs*), as first observed by Bonet (personal communications) and recently studied in more depth (Edelkamp et al., 2012; Torralba et al., 2013; Helmert et al., 2014; Torralba, 2015). Regarding the non-linear case, Helmert et al. (2014) make a number of conjectures entailing that general FMs are strictly more powerful than linear ones. In this chapter, we prove these conjectures correct.

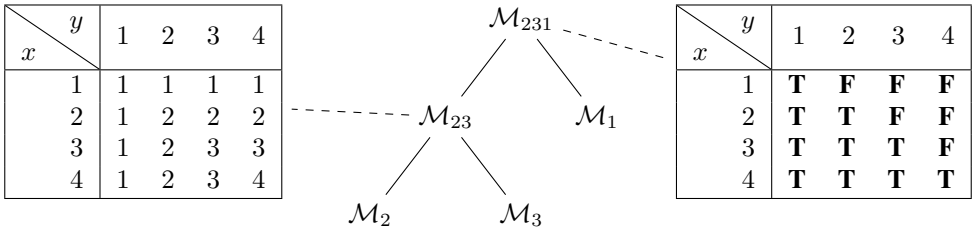
The chapter closely follows and in large parts stems from the paper *On the Expressive Power of Non-Linear Merge-and-Shrink Representations* (Helmert et al., 2015), adapted to the notation used in this thesis. It is structured as follows. After a reminder of how FMs work, we briefly repeat results concerning the expressive power in previous work. Then we prove that general FMs are strictly more powerful than linear ones by showing that there exist problem families that can be represented compactly with general FMs but not with linear ones. Furthermore, we give a precise bound that quantifies the necessary blowup incurred by conversions from general FMs to linear ones. We conclude with a discussion and directions for future work.

4.1. Factored Mappings

As we already defined FMs in Section 3.2.3, cf. Definition 3.9, we do not repeat their definition here. However, we need some extra notations and we give an additional example illustrating the definition and the computation of FMs which we use throughout the remainder of this chapter.

FMs represent functions defined on assignments over finite-domain variables, which the following definition formalizes.

Definition 4.1. *Let $\mathcal{V}$ be a set of* variables*, each with a finite* domain $dom(v) \neq \emptyset$.



x \ y	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	2	3	3
4	1	2	3	4

x \ y	1	2	3	4
1	**T**	**F**	**F**	**F**
2	**T**	**T**	**F**	**F**
3	**T**	**T**	**T**	**F**
4	**T**	**T**	**T**	**T**

Figure 4.1.: Example FM (value sets and function tables for identity functions at the leaves omitted).

An assignment *of a variable set $\mathcal{V}$ is a function α defined on $\mathcal{V}$ with $\alpha(v) \in dom(v)$ for all $v \in \mathcal{V}$. We write $vars(\alpha)$ for the variable set $\mathcal{V}$ (i.e. for the domain of definition of α).*

Throughout the chapter, we assume that some underlying finite set of variables $\mathcal{V}$ is given.

Recall that FMs have an associated value set. If the FM is atomic, it has an associated variable, and its table is a function mapping from the domain of that variable to its value set. If it is a merge, then it has a left and right component, both FMs, and its table is a function mapping from the product of the value sets of the components to its own value set. Sometimes, we need to reason about the variables of an FM $\mathcal{M}$, i.e. the associated variables of all atomic FMs that are (recursive) components of $\mathcal{M}$. We write $vars(\mathcal{M})$ for this set of variables.

Figure 4.1 illustrates the definition of FMs. The variables are $\mathcal{V} = \{v_1, v_2, v_3\}$ with domains $dom(v_1) = dom(v_2) = dom(v_3) = \{1, 2, 3, 4\}$. The overall FM, $\mathcal{M}_{231}$, is a merge with left component $\mathcal{M}_{23}$ and right component $\mathcal{M}_1$. $\mathcal{M}_{23}$ is a merge with left component $\mathcal{M}_2$ and right component $\mathcal{M}_3$. $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ are atomic.

For the atomic FMs $\mathcal{M}_i$, the variable is v_i, the value set is $\{1, 2, 3, 4\}$, and the table is simply the identity function. The value set of $\mathcal{M}_{23}$ is also $\{1, 2, 3, 4\}$, and its table is the minimum function: $\mathcal{M}_{23}^{\text{tab}}(x, y) = \min(x, y)$. Finally, the value set of $\mathcal{M}_{231}$ consists of the truth values $\{\mathbf{T}, \mathbf{F}\}$, and its table is defined by $\mathcal{M}_{231}^{\text{tab}}(x, y) = \mathbf{T}$ iff $x \geq y$.

Recall the definition of the represented function of an FM: an atomic FM defines an arbitrary function on the values of the associated variable, and a merge FM recursively computes a function value for each of its two components and combines the subresults into the final result.

To illustrate, consider the computation of the example FM $\mathcal{M}_{231}$ for a given example assignment:

$$
\begin{aligned}
&\mathcal{M}_{231}(\{v_1 \mapsto 2, v_2 \mapsto 4, v_3 \mapsto 3\}) \\
&= \mathcal{M}_{231}^{\text{tab}}(\mathcal{M}_{23}(\{v_2 \mapsto 4, v_3 \mapsto 3\}), \mathcal{M}_1(\{v_1 \mapsto 2\})) \\
&= \mathcal{M}_{231}^{\text{tab}}(\mathcal{M}_{23}^{\text{tab}}(\mathcal{M}_2(\{v_2 \mapsto 4\}), \mathcal{M}_3(\{v_3 \mapsto 3\})), \\
&\qquad\qquad \mathcal{M}_1(\{v_1 \mapsto 2\})) \\
&= \mathcal{M}_{231}^{\text{tab}}(\mathcal{M}_{23}^{\text{tab}}(\mathcal{M}_2^{\text{tab}}(4), \mathcal{M}_3^{\text{tab}}(3)), \mathcal{M}_1^{\text{tab}}(2)) \\
&= \mathcal{M}_{231}^{\text{tab}}(\mathcal{M}_{23}^{\text{tab}}(4, 3), 2) \\
&= \mathbf{T} \text{ iff } \min(4, 3) \geq 2 \\
&= \mathbf{T}
\end{aligned}
$$

More generally, $\mathcal{M}_{231}$ represents a predicate that is true for assignment α iff $\min(\alpha(v_2), \alpha(v_3)) \geq \alpha(v_1)$.

Recall the notion of linear and non-linear FMs, and with that, merge trees and merge strategies (cf. Section 3.4). We make this notion more precise here by calling an FM linear if it is atomic or if it is a merge whose w.l.o.g. right component is atomic and whose left component is linear. The important defining property of a linear FM is that it never contains merges of two merges. We also remark that the structure of a linear FM is fully defined by the sequence of associated variables for the leaves of its tree representation, read from left to right. We call this sequence the *variable order* of the FM. The example FM in Figure 4.1 is linear, and its variable order is $\langle v_2, v_3, v_1 \rangle$. Sometimes we speak of *general* FMs to emphasize that we want to include both linear and non-linear FMs.

We will study the question which functions can be represented compactly with linear and with general FMs. For this purpose, we need to define a measure for the representation size of an FM. We assume that the tables are represented explicitly by their entries, and hence the total number of table entries dominates the overall representation size and defines a suitable size measure.

Definition 4.2. *The* size *of an FM $\mathcal{M}$, written as $\|\mathcal{M}\|$, and the* size *of its table, written as $\|\mathcal{M}^{\text{tab}}\|$, are defined inductively as follows. If $\mathcal{M}$ is atomic with variable v, then $\|\mathcal{M}\| = \|\mathcal{M}^{\text{tab}}\| = |dom(v)|$. If $\mathcal{M}$ is a merge, then $\|\mathcal{M}^{\text{tab}}\| = |vals(\mathcal{M}_{\text{L}})| \cdot |vals(\mathcal{M}_{\text{R}})|$ and $\|\mathcal{M}\| = \|\mathcal{M}_{\text{L}}\| + \|\mathcal{M}_{\text{R}}\| + \|\mathcal{M}^{\text{tab}}\|$.*

In the example, we obtain $\|\mathcal{M}_{231}\| = 2 \cdot 4 \cdot 4 + 3 \cdot 4 = 44$, which counts the two 4×4 tables for the merges and the three size-4 tables for the atomic FMs. If we generalize the example to a larger domain with D values, we obtain a representation size of $\Theta(D^2)$ to define a function on D^3 possible variable assignments.

In order to analyze the scalability of FMs, we need to consider representation sizes for *families* of functions.

Definition 4.3. *Let $\mathcal{F}$ be a family of functions on variable assignments. We say that $\mathcal{F}$* has compact FMs *if there exists a polynomial p such that for each function $f \in \mathcal{F}$ defined over variables $\mathcal{V}_f$, there exists an FM $\mathcal{M}_f$ for f with $\|\mathcal{M}_f\| \leq p(\sum_{v \in \mathcal{V}_f} |dom(v)|)$.*

We say that $\mathcal{F}$ has compact linear FMs *if additionally all FMs $\mathcal{M}_f$ are linear.*

In words, the size of compact representations is at most polynomial in the sum of the variable domain sizes (i.e. in the number of variable/value pairs).

4.2. Expressive Power of Merge-and-Shrink in Previous Work

It is easy to see that FMs can represent arbitrary functions on assignments, so the study of the expressive power of the merge-and-shrink framework has focused on the (more practically important) question which functions can be represented *compactly*.

It is well-known that a family of functions can be represented compactly with *linear* FMs iff it can be represented compactly with ADDs,[1] which in turn is equivalent to having compact FMs

[1] ADDs only directly support functions over assignments to binary variables. To bridge this gap, Helmert et al. and Torralba encode non-binary variables as contiguous sequences of binary ADD variables, as commonly done in symbolic planning algorithms.

by families of BDDs (e.g. Torralba, 2015).

In more detail, Helmert et al. (2014) and Torralba (2015) both prove results that relate linear FMs to ADDs representing the same function. Translated to our notation, Helmert et al. show that every linear FM $\mathcal{M}$ can be converted into an equivalent ADD of size $O(|V|T^{\max}D^{\max})$, where $V = vars(\mathcal{M})$, $T^{\max}$ is the maximum over the table sizes of $\mathcal{M}$ and its (direct and indirect) subcomponents, and $D^{\max} = \max_{v \in V} |dom(v)|$. A finer-grained analysis, which also follows from results of Torralba, gives a slightly better bound of $O(\|\mathcal{M}\|D^{\max})$. (Note that $\|\mathcal{M}\| = O(|V|T^{\max})$ because $\|\mathcal{M}\|$ is the sum of $O(|V|)$ table sizes, each of which is individually bounded by $T^{\max}$. Hence the second bound is at least as tight as the first one.)

In the opposite direction, it is easy to convert an arbitrary ADD with N nodes into a linear FM $\mathcal{M}$ with $\|\mathcal{M}\| = O(N \cdot |V|)$, where V is the set of ADD variables. The conversion first performs the opposite of *Shannon reductions* (which, in the worst case, replaces each ADD edge by a structure of size $O(|V|)$) and then converts the resulting structure into a linear FM, which incurs no further blowup.

For *non-linear* FMs, no such bounds are known. However, Helmert et al. (2014) made the following conjecture:

> We conjecture that ... there exists a family $\mathcal{T}_n$ of (non-linear) 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.

In the following we will prove this conjecture correct, including the relation to the Horton-Strahler number. Rather than working with ADDs, we prove the results directly on the level of general vs. linear FMs; the conjectured relationship to ADDs then follows from the above discussion. Specifically, we will prove:

1. Every FM $\mathcal{M}$ can be converted into a linear FM of size at most $\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})}$, where $\mathrm{HS}(\mathcal{M})$ is the Horton-Strahler number (see next section) of the merge strategy of $\mathcal{M}$. $\mathrm{HS}(\mathcal{M})$ is bounded from above by $\log_2 n + 1$, where n is the number of variables.

2. There exist families of functions which can be compactly represented by general FMs, but every linear FM has size $\Omega(\|\mathcal{M}\|^{c \log_2 n})$, where $c > 0$ is a constant, n is the number of variables, and $\mathcal{M}$ is a general FM representing the same function.

4.3. General to Linear FMs

The Horton-Strahler number (Horton, 1945) of a rooted binary tree is a measure of its "bushiness". The measure is high for complete trees and low for thin trees.[2] For this paper, it is easiest to define the measure directly for FMs.

[2] Horton-Strahler numbers of rooted trees are closely related to the *pathwidth* (e.g. Cattell, Dinneen, & Fellows, 1996) of the tree seen as an undirected graph. It can be shown that the Horton-Strahler number is always in the range $\{k, \ldots, 2k\}$, where k is the pathwidth (Mamadou M. Kanté, personal communications).

Definition 4.4. *The* Horton-Strahler *number of an FM $\mathcal{M}$, written* $\mathrm{HS}(\mathcal{M})$*, is inductively defined as* $\mathrm{HS}(\mathcal{M}) = \begin{cases} 1 & \text{if } \mathcal{M} \text{ is atomic} \\ \max(\mathrm{HS}(\mathcal{M}_\mathrm{L}), \mathrm{HS}(\mathcal{M}_\mathrm{R})) & \text{if } \mathrm{HS}(\mathcal{M}_\mathrm{L}) \neq \mathrm{HS}(\mathcal{M}_\mathrm{R}) \\ \mathrm{HS}(\mathcal{M}_\mathrm{L}) + 1 & \text{if } \mathrm{HS}(\mathcal{M}_\mathrm{L}) = \mathrm{HS}(\mathcal{M}_\mathrm{R}) \end{cases}$

It is easy to prove inductively that the smallest tree (in terms of number of nodes) with Horton-Strahler number k is the complete binary tree with k layers, from which it follows that Horton-Strahler numbers grow at most logarithmically in the number of tree nodes.

For FMs, this implies $\mathrm{HS}(\mathcal{M}) \leq \log_2 |vars(\mathcal{M})| + 1$. If $\mathcal{M}$ is linear, then $\mathrm{HS}(\mathcal{M}) = 2$, unless $\mathcal{M}$ is atomic (in which case $\mathrm{HS}(\mathcal{M}) = 1$). More generally, FMs with small Horton-Strahler number are "close" to being linear. In this section we show that general FMs $\mathcal{M}$ can be converted to linear ones with an increase in representation size that is exponential only in $\mathrm{HS}(\mathcal{M})$. In the following section, we then show that this blow-up is unavoidable.

Before we can prove the main result of this section, we show how to convert an FM with a *single* violation of the linearity condition into a linear one. The main result uses this construction as a major ingredient.

Throughout the section, we will refer to the *parts* of an FM $\mathcal{M}$, by which we mean its direct and indirect subcomponents, including $\mathcal{M}$ itself.

Lemma 4.1. *Let $\mathcal{M}$ be a merge FM where $\mathcal{M}_\mathrm{L}$ and $\mathcal{M}_\mathrm{R}$ are linear. Then there exists a linear FM $\mathcal{M}^{\mathrm{lin}}$ representing the same function as $\mathcal{M}$ with $\|\mathcal{M}^{\mathrm{lin}}\| \leq \|\mathcal{M}_\mathrm{L}\| + (|vals(\mathcal{M}_\mathrm{L})| + 1)\|\mathcal{M}_\mathrm{R}\|$.*

Proof. We demonstrate the construction of $\mathcal{M}^{\mathrm{lin}}$ given $\mathcal{M}$. To reduce notational clutter, we set $\mathcal{L} = \mathcal{M}_\mathrm{L}$ and $\mathcal{R} = \mathcal{M}_\mathrm{R}$ for the two (linear) components of $\mathcal{M}$. Further, we set $r := |vars(\mathcal{R})|$ and write $\langle v_1, \ldots, v_r \rangle$ for the variable order underlying $\mathcal{R}$. For $1 \leq i \leq r$, we write $\mathcal{R}_i^{\mathrm{var}}$ for the atomic part of $\mathcal{R}$ with variable v_i and $\mathcal{R}_i$ for the part of $\mathcal{R}$ with $vars(\mathcal{R}_i) = \{v_1, \ldots, v_i\}$. (This implies that $\mathcal{R}_1 = \mathcal{R}_1^{\mathrm{var}}$, and hence $\mathcal{R}_1$, unlike the other $\mathcal{R}_i$, is atomic.)

If $\mathcal{R}$ is atomic, then $\mathcal{M}$ is already linear and we can set $\mathcal{M}^{\mathrm{lin}} = \mathcal{M}$. In the following we can thus assume $r \geq 2$.

We construct a sequence of linear FMs $\mathcal{M}_0, \ldots, \mathcal{M}_r$ such that $\mathcal{M}^{\mathrm{lin}} = \mathcal{M}_r$ is the desired linear FM for $\mathcal{M}$. To initialize the construction, we set $\mathcal{M}_0 = \mathcal{L}$. All other FMs $\mathcal{M}_i$ are merges, so to define them we must specify their left and right components, value sets, and tables.

For $1 \leq i \leq r$, $\mathcal{M}_i$ is a merge with left component $\mathcal{M}_{i-1}$ and right component $\mathcal{R}_i^{\mathrm{var}}$. The value sets are $vals(\mathcal{M}_i) = vals(\mathcal{L}) \times vals(\mathcal{R}_i)$ for all $i \neq r$ and $vals(\mathcal{M}_r) = vals(\mathcal{M})$.

Finally, the tables are defined as follows:

- $i = 1$: $\mathcal{M}_1^{\mathrm{tab}}(l, x) = \langle l, \mathcal{R}_1^{\mathrm{tab}}(x) \rangle$
- $1 < i < r$: $\mathcal{M}_i^{\mathrm{tab}}(\langle l, x \rangle, y) = \langle l, \mathcal{R}_i^{\mathrm{tab}}(x, y) \rangle$
- $i = r$: $\mathcal{M}_r^{\mathrm{tab}}(\langle l, x \rangle, y) = \mathcal{M}^{\mathrm{tab}}(l, \mathcal{R}_r^{\mathrm{tab}}(x, y))$

To see that this FM represents the desired function, it is perhaps easiest to view the evaluation of an FM for a given assignment as a bottom-up process. Computing $\mathcal{M}(\alpha)$ with the original FM entails first computing $\mathcal{L}(\alpha)$, then $\mathcal{R}(\alpha)$ and finally feeding the two component results into

$\mathcal{M}^{\text{tab}}$. During the evaluation of $\mathcal{R}$, the result for $\mathcal{L}$ must be "remembered" because it will be needed at the end when plugging the two component results into $\mathcal{M}^{\text{tab}}$.

The linearized FM emulates this process by first computing $\mathcal{L}(\alpha)$ and then propagating the resulting value l, unchanged, through all the computation steps of $\mathcal{R}$ as part of the value sets, until the final step, when $\mathcal{R}(\alpha)$ becomes available and hence the final result can be computed.

To prove the claimed size bound, observe that all tables of $\mathcal{L}$ occur identically in $\mathcal{M}^{\text{lin}}$, so the total size of these tables in $\mathcal{M}^{\text{lin}}$ can be bounded by $\|\mathcal{L}\|$. Similarly, the tables of the atomic parts of $\mathcal{R}$ occur identically in $\mathcal{M}^{\text{lin}}$ and their size can be bounded by $\|\mathcal{R}\|$. The remaining parts of $\mathcal{M}^{\text{lin}}$ are the merges $\mathcal{M}_i$ with $1 \leq i \leq r$. The size of $\mathcal{M}_i^{\text{tab}}$ is $|vals(\mathcal{L})|$ times the size of $\mathcal{R}_i^{\text{tab}}$ (in $\mathcal{R}$), and hence the total size of these tables can be bounded by $|vals(\mathcal{L})|\|\mathcal{R}\|$. Summing these three terms gives the desired overall bound. □

Armed with this construction, we can now proceed to prove the main result.

Theorem 4.1. *Let $D^{\text{sum}}(\mathcal{M}) = \sum_{v \in vars(\mathcal{M})} |dom(v)|$. For every FM $\mathcal{M}$ there is a linear FM $\mathcal{M}^{\text{lin}}$ representing the same function with $\|\mathcal{M}^{\text{lin}}\| \leq D^{\text{sum}}(\mathcal{M})\|\mathcal{M}\|^{\text{HS}(\mathcal{M})-1} \leq \|\mathcal{M}\|^{\text{HS}(\mathcal{M})}$.*

Proof. Clearly $D^{\text{sum}}(\mathcal{M}) \leq \|\mathcal{M}\|$, and thus it is sufficient to show the first part of the inequality, i.e. $\|\mathcal{M}^{\text{lin}}\| \leq D^{\text{sum}}(\mathcal{M})\|\mathcal{M}\|^{\text{HS}(\mathcal{M})-1}$. This holds for linear FMs with $\mathcal{M}^{\text{lin}} = \mathcal{M}$, so in the following let $\mathcal{M}$ be non-linear. In particular, we will use that $\mathcal{M}$ is not atomic.

Let B be the maximum of $|vals(\mathcal{P})|$ over all parts $\mathcal{P}$ of $\mathcal{M}$ with $\mathcal{P} \neq \mathcal{M}$. We have $B + 1 \leq \|\mathcal{M}\|$ because each value set size $|vals(P)|$ contributes to $\|\mathcal{M}\|$ (in the table of a merge), and it is not the only term that contributes to $\|\mathcal{M}\|$ (because $\mathcal{M}$ is not atomic).

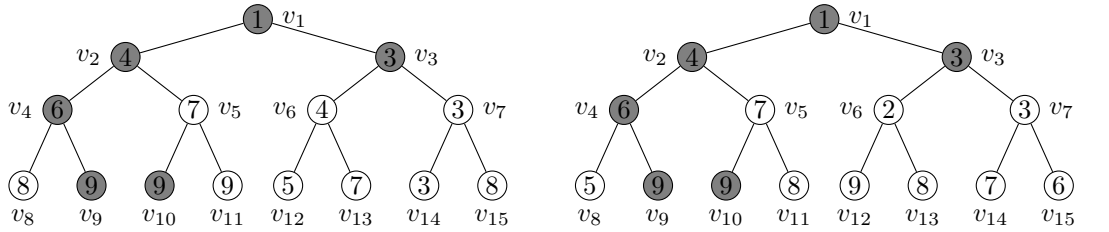
We now prove that every part $\mathcal{P}$ of $\mathcal{M}$ (including $\mathcal{P} = \mathcal{M}$) can be converted to a linear FM $\mathcal{P}^{\text{lin}}$ of size at most $D^{\text{sum}}(\mathcal{P})\|\mathcal{M}\|^{\text{HS}(\mathcal{P})-1}$. The claim then follows by setting $\mathcal{P} = \mathcal{M}$.

The proof is by induction over the number of variables in $\mathcal{P}$. If $|vars(\mathcal{P})| = 1$, $\mathcal{P}$ is atomic and the result holds because $\mathcal{P}$ is already linear and satisfies $\|\mathcal{P}\| = D^{\text{sum}}(\mathcal{P})$.

For the inductive step, let $\mathcal{P}$ be a merge. Let $k = \text{HS}(\mathcal{P})$, $k_{\text{L}} = \text{HS}(\mathcal{P}_{\text{L}})$ and $k_{\text{R}} = \text{HS}(\mathcal{P}_{\text{R}})$. We can apply the inductive hypothesis to linearize $\mathcal{P}_{\text{L}}$ and $\mathcal{P}_{\text{R}}$, obtaining linear FMs $\mathcal{P}_{\text{L}}^{\text{lin}}$ and $\mathcal{P}_{\text{R}}^{\text{lin}}$ with $\|\mathcal{P}_{\text{L}}^{\text{lin}}\| \leq D^{\text{sum}}(\mathcal{P}_{\text{L}})\|\mathcal{M}\|^{k_{\text{L}}-1}$ and $\|\mathcal{P}_{\text{R}}^{\text{lin}}\| \leq D^{\text{sum}}(\mathcal{P}_{\text{R}})\|\mathcal{M}\|^{k_{\text{R}}-1}$.

If $k_{\text{L}} > k_{\text{R}}$, let $\mathcal{Q}$ be the FM with left component $\mathcal{P}_{\text{L}}^{\text{lin}}$ and right component $\mathcal{P}_{\text{R}}^{\text{lin}}$ and the same table and value set as $\mathcal{P}$. Clearly $\mathcal{Q}$ represents the same function as $\mathcal{P}$ and satisfies the criteria of Lemma 4.1, so we can apply the lemma to receive a linear FM $\mathcal{Q}^{\text{lin}}$ with size at most:

$$
\begin{aligned}
\|\mathcal{Q}^{\text{lin}}\| &\leq \|\mathcal{P}_{\text{L}}^{\text{lin}}\| + (|vals(\mathcal{P}_{\text{L}}^{\text{lin}})| + 1)\|\mathcal{P}_{\text{R}}^{\text{lin}}\| \\
(*) \quad &\leq \|\mathcal{P}_{\text{L}}^{\text{lin}}\| + (B+1)\|\mathcal{P}_{\text{R}}^{\text{lin}}\| \\
&\leq \|\mathcal{P}_{\text{L}}^{\text{lin}}\| + \|\mathcal{M}\|\|\mathcal{P}_{\text{R}}^{\text{lin}}\| \\
&\leq D^{\text{sum}}(\mathcal{P}_{\text{L}})\|\mathcal{M}\|^{k_{\text{L}}-1} + \|\mathcal{M}\|D^{\text{sum}}(\mathcal{P}_{\text{R}})\|\mathcal{M}\|^{k_{\text{R}}-1} \\
&= D^{\text{sum}}(\mathcal{P}_{\text{L}})\|\mathcal{M}\|^{k_{\text{L}}-1} + D^{\text{sum}}(\mathcal{P}_{\text{R}})\|\mathcal{M}\|^{k_{\text{R}}} \\
(**) \quad &\leq D^{\text{sum}}(\mathcal{P}_{\text{L}})\|\mathcal{M}\|^{k-1} + D^{\text{sum}}(\mathcal{P}_{\text{R}})\|\mathcal{M}\|^{k-1} \\
&= (D^{\text{sum}}(\mathcal{P}_{\text{L}}) + D^{\text{sum}}(\mathcal{P}_{\text{R}}))\|\mathcal{M}\|^{k-1} \\
&= D^{\text{sum}}(\mathcal{P})\|\mathcal{M}\|^{k-1}
\end{aligned}
$$



(a) Assignment α, satisfying the heap condition.

(b) Assignment α', violating the heap condition because $\alpha'(v_3) > \alpha'(v_6)$ and $\alpha'(v_4) > \alpha'(v_8)$.

Figure 4.2.: Two example assignments for heap predicate $\varphi_{15,9}$.

where (*) uses that $\mathcal{P}_{\mathrm{L}}^{\mathrm{lin}}$ has the same value set as $\mathcal{P}_{\mathrm{L}}$, which is a part of $\mathcal{M}$, and hence the value set bound B applies to it; (**) uses that $k = k_{\mathrm{L}} > k_{\mathrm{R}}$ in the case considered here.

If $k_{\mathrm{L}} < k_{\mathrm{R}}$, we use essentially the same construction, but make $\mathcal{P}_{\mathrm{R}}^{\mathrm{lin}}$ the *left* and $\mathcal{P}_{\mathrm{L}}^{\mathrm{lin}}$ the *right* component of $\mathcal{Q}$, which allows us to swap the roles of k_{L} and k_{R} in the computation. To represent the same function despite the swapped subcomponents, we set $\mathcal{Q}^{\mathrm{tab}}(x, y) = \mathcal{P}^{\mathrm{tab}}(y, x)$.

Finally, if $k_{\mathrm{L}} = k_{\mathrm{R}}$, we use the same construction and computation as with $k_{\mathrm{L}} > k_{\mathrm{R}}$. Step (**) remains correct because in this case $k = k_{\mathrm{L}} + 1 = k_{\mathrm{R}} + 1$.

In all cases, $\mathcal{Q}^{\mathrm{lin}}$ is the desired linear FM: we showed that it meets the size bound, and it represents the same function as $\mathcal{Q}$, which represents the same function as $\mathcal{P}$. □

4.4. Separation of General and Linear Merge-and-Shrink

In this section, we show that the size increase in the conversion from general to linear FMs in Theorem 4.1 is unavoidable in general. To prove the result, we describe a family of functions which have compact general FMs, but do not have compact linear FMs. More precisely, we will show that if $\mathcal{M}$ is a compact general FM for such a function f, then every linear FM must have size $\Omega(\|\mathcal{M}\|^{\Theta(\log n)})$, where n is the number of variables of f. This lower bound matches the upper bound of $\|\mathcal{M}\|^{\mathrm{HS}(\mathcal{M})}$ from the previous section.

The functions we consider are *predicates*, i.e. they map to the set of truth values $\{\mathbf{T}, \mathbf{F}\}$. We say that an assignment α *satisfies* a predicate ψ if $\psi(\alpha) = \mathbf{T}$. We now introduce the family of predicates we will study.

Definition 4.5. *Let $k \in \mathbb{N}_1$, let $n = 2^k - 1$, and let $D \in \mathbb{N}_1$. The* heap predicate $\varphi_{n,D}$ *is a predicate defined over variables $V = \{v_1, \ldots, v_n\}$ with $dom(v) = \{1, \ldots, D\}$ for all $v \in V$. An assignment α of V satisfies $\varphi_{n,D}$ iff $\alpha(v_{\lfloor i/2 \rfloor}) \leq \alpha(v_i)$ for all $1 < i \leq n$. In this case, we say that α satisfies the* heap condition.

In the following, we omit k, n and D from the notation when this does not lead to confusion. Observe that $k = \Theta(\log n)$, or more precisely $k = \log_2(n + 1)$.

The heap predicate has its name because it is the condition characterizing sequences of values that represent (*min-*) *heaps*, known from algorithm theory (e.g. Cormen, Leiserson, & Rivest, 1990). Figure 4.2 shows two example assignments for heaps of height $k = 4$, i.e. with $n =$

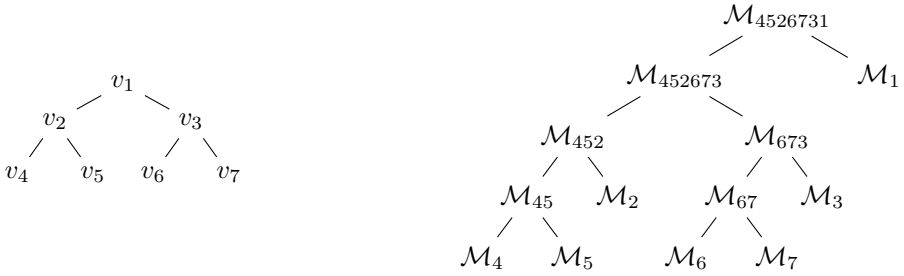


Figure 4.3.: Heap and corresponding merge strategy.

$2^4 - 1 = 15$ variables. (Ignore the gray shading of some nodes for now.) We follow the common convention in algorithm theory of displaying the assignment as a complete binary tree (not to be confused with trees representing FMs!), where the variables are numbered in breadth-first order. With this mode of display, the heap condition is satisfied iff the value assigned to each inner node is less than or equal to the values assigned to its children. The assignment shown at the left satisfies the heap condition, while the one at the right does not.

4.4.1. General FMs for Heaps

It is easy to see that heap predicates can be represented compactly with general FMs. In fact, the first FM we showed in Figure 4.1 is an example of this, representing the heap predicate $\varphi_{3,4}$.

The general case is only slightly more complex and is illustrated for $n = 7$ variables ($k = 3$) in Figure 4.3. The left-hand side shows the tree order of the heap variables, and the right-hand side shows the corresponding merge strategy, displayed as the tree structure of the (final) FM.

The idea is to build merges "along" the tree underlying the heap. For each variable v_i there is a *subtree FM*, whose variable set includes all variables in the subtree of the heap rooted at v_i, and for each inner variable (in the example: v_1, v_2, v_3) there is a *descendant FM*, whose variable set includes the same variables as the subtree FM but not v_i itself. For example, $\mathcal{M}_{452}$ is the subtree FM for v_2 and $\mathcal{M}_{45}$ is the descendant FM for v_2 in Figure 4.3. For inner variables v_i, the descendant FM is the merge of the subtree FMs of the (heap) children of v_i, and the subtree FM is the merge of the descendant FM of v_i and an atomic FM for v_i. Leaf nodes v_i of the heap do not have descendant FMs, and their subtree FM is an atomic FM for v_i.

The value sets and tables for these FMs are constructed in such a way that each FM represents the *minimum* function over all variables it covers. This can easily be computed locally by setting $\mathcal{M}^{\text{tab}}(x, y) = \min(x, y)$ everywhere. To complete the construction, we need a way of determining violations of the heap condition. This is accomplished by adding an additional value *violation* to the value sets of all merges, which is set at the subtree FM of inner variable v_i whenever $\alpha(v_i)$ is larger than the value propagated up from the descendant FM for v_i. We define $\min(x, y) =$ *violation* whenever $x =$ *violation* or $y =$ *violation*, which ensures that violations are propagated towards the root. To complete the construction, the table at the root converts *violation* to **F** and all other values to **T**.

Theorem 4.2. *The family of all heap predicates has compact FMs.*

Proof. It is easy to verify that the FMs $\mathcal{M}_{n,D}$ described in the text represent the heap functions $\varphi_{n,D}$. We can bound the representation size by $\|\mathcal{M}_{n,D}\| \leq n \cdot D + (n-1)(D+1)^2 = O(nD^2)$, where the first term counts the n tables of size D in the atomic FMs, and the other term counts the $n-1$ tables of size $(D+1)^2$ (taking into account the additional *violation* value) at merges. This is polynomial in the sum over the domain sizes, which is $\sum_{i=1}^{n} D = nD$. □

4.4.2. Linear FMs for Heaps

In our final result, we show that heap predicates cannot be represented compactly with linear FMs, no matter which variable order is used. Towards this end, we need some additional definitions relating to assignments in the context of heap predicates. Throughout the section, $V = \{v_1, \ldots, v_n\}$ refers to the variables of a given heap predicate $\varphi_{n,D}$.

We make heavy use of *partial* assignments (assignments to subsets of variables) in the following. For $V' \subseteq V$, a *V'-assignment* is an assignment for V'. When we speak of *assignments* without further qualification, we mean assignments to all variables V.

Definition 4.6. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $V' \subseteq V$. The* unassigned variables *of V' are the variables $V \setminus V'$. An assignment for the unassigned variables of V' is called a* completion *of a V'-assignment.*

A V'-assignment σ together with a completion τ forms an assignment, which we write as $\sigma \cup \tau$.

A completion τ is a valid completion *for V'-assignment σ if $\sigma \cup \tau$ satisfies the heap condition. A V'-assignment is called* consistent *if it has at least one valid completion.*

To illustrate the definition, we refer back to Figure 4.2. The variable subset $V' = \{v_1, v_2, v_3, v_4, v_9, v_{10}\}$ is shaded in gray, and the unassigned variables are shown in white. We verify that the V'-assignment in both parts of the figure is identical, so the figure shows a V'-assignment σ together with a valid completion τ at the left (which proves that σ is consistent) and an invalid completion τ' at the right.

We make the following central observation: if the variable set V' occurs as a prefix of the variable order of a given linear FM for $\varphi_{n,D}$, then the sub-FM $\mathcal{M}_{V'}$ with variables V' may only "combine" (map to the same value) V'-assignments that have exactly the same valid completions. Otherwise the final FM cannot faithfully represent the heap predicate. Hence, the number of elements in $vals(\mathcal{M}_{V'})$ is at least as large as the number of V'-assignments that are *distinguishable* in this sense.[3]

Definition 4.7. *V'-assignments σ and σ' are called* equivalent *if they have exactly the same set of valid completions. V'-assignments that are not equivalent are called* distinguishable.

We will show that, no matter which variable order is chosen, some part of the linear FM must have a large number of distinguishable V'-assignments. To prove this result, we will need some more definitions.

[3] The same observation underlies the *Myhill-Nerode theorem* on the size of minimal deterministic finite automata (e.g. Hopcroft, Motwani, & Ullman, 2001, Section 4.4) and the *Sieling-Wegener bound* for the minimal representation size of BDDs (Sieling & Wegener, 1993).

Definition 4.8. *The* neighbors *of v_i are its parent in the heap, $v_{\lfloor i/2 \rfloor}$ (if $i > 1$) and its children in the heap, v_{2i} and v_{2i+1} (if $i < 2^{k-1}$). A variable $v_i \in V'$ is called a* frontier variable *of V' if it has an unassigned neighbor.*

In the example of Figure 4.2, the frontier variables of V' are v_2 (due to its child v_5), v_3 (due to its children v_6 and v_7), v_4 (due to its child v_8) and v_{10} (due to its parent v_5).

The frontier variables are the ones that relate the variables assigned in V' to the unassigned variables, which makes them particularly important. It is easy to see that, given a consistent V'-assignment, *only* the values of the frontier variables matter when deciding whether a given completion is valid. (We do not prove this result because it does not help prove a lower bound, but it may serve to provide intuition for the role of frontier variables.)

We now show that for every variable order, there must be a sub-FM of the overall FM with a certain minimal number of frontier variables.

Lemma 4.2. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $\pi = \langle v_{j_1}, \ldots, v_{j_n} \rangle$ be any variable order of V. Then there exists some $r \in \{1, \ldots, n\}$ such that the set of variables $V' = \{v_{j_1}, \ldots, v_{j_r}\}$ defined by the length-r prefix of π includes at least $\lfloor \frac{k}{2} \rfloor$ frontier variables.*

Proof. The problem of determining an ordering π of the vertices of a graph that minimizes the maximal number of frontier variables of any prefix of π has been studied in graph theory. The minimal number of frontier variables required is known as the *vertex separation number* of the graph and is known to be equal to its *pathwidth* (Kinnersley, 1992).

Here, the graphs we must consider are the complete binary trees T_k with k layers. Cattell et al. (1996) show that the pathwidth of T_k is at least $\lfloor \frac{k}{2} \rfloor$. □

The basic idea of our main proof is that if there are many frontier variables, then there exist many V'-assignments that need to be distinguished. Towards this end, we introduce certain properties that will help us determine that given V'-assignments are distinguishable.

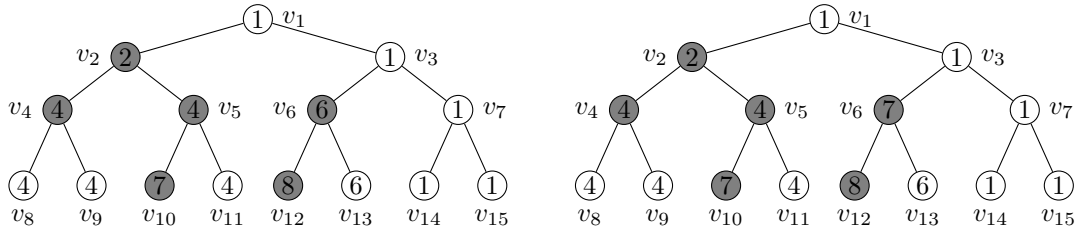
Definition 4.9. *A V'-assignment σ is called* sorted *if $\sigma(v_i) \leq \sigma(v_j)$ whenever $i < j$. A V'-assignment σ is called* fraternal *if, for any two variables $v, v' \in V'$ that are siblings in the heap (i.e. $v = v_{2i}$ and $v' = v_{2i+1}$ for some $1 \leq i < 2^{k-1}$), we have $\sigma(v) = \sigma(v')$.*

For example, the V'-assignment shown in gray in Figure 4.2 is not sorted because $\sigma(v_2) > \sigma(v_3)$ even though $2 < 3$. Furthermore, it is not fraternal because v_2 and v_3 are siblings and have different values $\sigma(v_2) \neq \sigma(v_3)$. It would be sorted and fraternal if we had $\sigma(v_2) = 3$.

Next, we show that with many frontier variables, many V'-assignments are sorted and fraternal.

Lemma 4.3. *Let V be the variable set of some heap predicate $\varphi_{n,D}$, and let $V' \subseteq V$ contain ℓ frontier variables. Then there exists a set $\Sigma(V')$ of sorted fraternal V'-assignments such that $|\Sigma(V')| = \binom{D}{\lceil \ell/2 \rceil}$ and any two assignments in $\Sigma(V')$ differ on at least one frontier variable of V'.*

Proof. We say that a variable is *sibling-bound* if it has a parent (i.e. it is not the root), it is the right child of its parent, and both the variable and its sibling are in V'. We say that a variable



(a) Sorted fraternal V'-assignment σ (gray nodes) with its minimal valid completion $\tau_{\min}$ (white nodes), satisfying the heap condition.

(b) Sorted fraternal V'-assignment σ' (gray nodes) with the same completion as in (a), violating the heap condition for v_6 and its child v_{13}.

Figure 4.4.: Minimal completions for two sorted fraternal V'-assignments σ and σ'.

is *sibling-bound in the frontier* if it is sibling-bound and additionally both the variable and its sibling are in the frontier. A *pivot candidate* is a frontier variable that is not sibling-bound in the frontier.

There can be at most $\lfloor \ell/2 \rfloor$ variables that are sibling-bound in the frontier because all such variables must have other frontier variables (that are not sibling-bound) as their siblings. Therefore, there are at least $\ell' = \ell - \lfloor \ell/2 \rfloor = \lceil \ell/2 \rceil$ pivot candidates. We arbitrarily select ℓ' pivot candidates and call them *pivots*.

We construct the set $\Sigma(V')$ by defining one sorted fraternal V'-assignment σ_X for each subset $X \subseteq \{1, \ldots, D\}$ of cardinality ℓ'. There are $\binom{D}{\ell'}$ such subsets, and the construction guarantees that σ_X and $\sigma_{X'}$ with $X \neq X'$ differ on at least one pivot and hence on at least one frontier variable.

We now describe how to construct σ_X for a given subset X. First, assign the values of X to the pivots in index order (i.e. the lowest value in X goes to the pivot with lowest index, etc.). This results in a sorted assignment to the pivots.

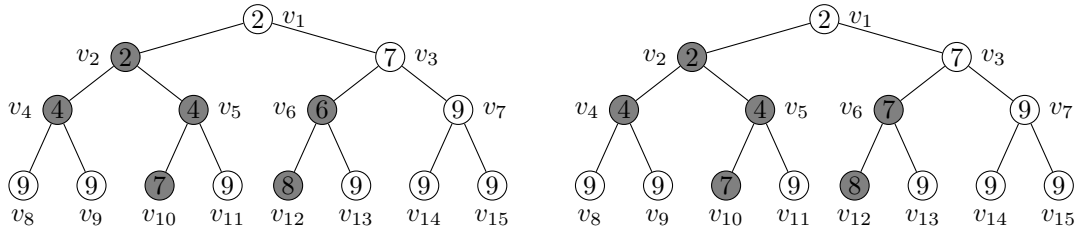
Next, to each variable in V' that is not sibling-bound and not a pivot, assign the value of the pivot that precedes it in index order, or 1 if no pivot precedes it. This maintains sortedness and results in an assignment to all non-sibling-bound variables in V'.

Finally, assign the value assigned to its sibling to each sibling-bound variable in V'. This results in a fraternal assignment to V' and again maintains sortedness.

As an example, consider the set of variables $V' = \{v_1, v_2, v_3, v_4, v_9, v_{10}\}$ from Figure 4.2 with $D = 9$. The frontier variables are $\{v_2, v_3, v_4, v_{10}\}$, giving $\ell = 4$ and $\ell' = 2$. The only sibling-bound variable in V' is v_3, and the other three frontier variables are pivot candidates. Select v_2 and v_4 as the pivots. Let $X = \{5, 7\}$. We first assign the values of X to the pivots in order: $\sigma_X(v_2) = 5$ and $\sigma_X(v_4) = 7$. Next, we fill in the remaining non-sibling-bound variables: $\sigma_X(v_1) = 1$, $\sigma_X(v_9) = 7$, $\sigma_X(v_{10}) = 7$. Finally, sibling-bound v_3 receives the value of its sibling: $\sigma_X(v_3) = 5$. □

As the final preparation for the main proof, we show that all assignments constructed in Lemma 4.3 are distinguishable.

Lemma 4.4. *Let σ and σ' be two sorted fraternal V'-assignments that differ on at least one frontier variable of V'. Then σ and σ' are distinguishable.*



(a) Sorted fraternal V'-assignment σ (gray nodes) with its maximal valid completion $\tau'_{\max}$ (white nodes), violating the heap condition between v_6 and its parent v_3.

(b) Sorted fraternal V'-assignment σ' with the same completion as in (a), satisfying the heap condition.

Figure 4.5.: Maximal completions for two sorted fraternal V'-assignments σ and σ', assuming $D = 9$.

Proof. It is easy to see that sorted V'-assignments are always consistent. More generally, a V'-assignment σ is consistent iff for all assigned variables $v_i, v_j \in V'$ where v_i is an ancestor of v_j in the tree representation of the heap, we have $\sigma(v_i) \leq \sigma(v_j)$. We start by describing two special completions of a given consistent V'-assignment.

The *minimal valid completion* $\tau_{\min}$ of a consistent V'-assignment σ assigns the lowest possible value that preserves consistency to each unassigned variable. It can be computed by iterating over all unassigned variables in index order (i.e. from the root towards the leaves) and assigning to each variable the lowest value that is consistent with its parent (if the variable has a parent). If the root variable v_1 is unassigned, this means setting $\tau_{\min}(v_1) := 1$. For any other unassigned variable v_i ($i > 1$), this means assigning the value of the parent: $\tau_{\min}(v_i) := (\sigma \cup \tau_{\min})(v_{\lfloor \frac{i}{2} \rfloor})$. (Note that when v_i is processed, the values of $\tau_{\min}$ for lower-index variables have already been constructed, so $(\sigma \cup \tau_{\min})(v_{\lfloor \frac{i}{2} \rfloor})$ is defined.) Figure 4.4a shows a V'-assignment σ together with its minimal valid completion.

Similarly, the *maximal valid completion* $\tau'_{\max}$ of a consistent V'-assignment σ' assigns the highest possible value that preserves consistency to each unassigned variable. It can be computed by iterating over all unassigned variables in reverse index order (i.e. from the leaves towards the root) and assigning to each variable the highest value that is consistent with its children (if the variable has children). For unassigned leaf variables v_i ($i \geq 2^{k-1}$), this means setting $\tau'_{\max}(v_i) := D$. For unassigned inner variables v_i ($i < 2^{k-1}$), this means assigning the minimum value of the children: $\tau'_{\max}(v_i) := \min((\sigma' \cup \tau'_{\max})(v_{2i}), (\sigma' \cup \tau'_{\max})(v_{2i+1}))$. (Again, this computation only depends on values of $\tau'_{\max}$ that have already been constructed.) Figure 4.5b shows a V'-assignment σ' together with its maximal valid completion.

We now prove the claim. Let σ and σ' be sorted fraternal V'-assignments that differ on some frontier variable v_i. Without loss of generality, we can assume $\sigma(v_i) < \sigma'(v_i)$ (otherwise swap the roles of σ and σ'). We will show that σ and σ' are distinguishable by describing a completion that is valid for one of σ or σ', but not the other.

Because v_i is a frontier variable of V', it belongs to V' and has a parent or child that is unassigned. We first consider the easier case where v_i has a child v_j ($j = 2i$ or $j = 2i + 1$) that is unassigned.

Consider the minimal valid completion $\tau_{\min}$ of σ. By construction, $\tau_{\min}$ is a valid completion of σ. From the definition of minimal valid completions, we have $\tau_{\min}(v_j) = \sigma(v_i)$. We also have $\sigma(v_i) < \sigma'(v_i)$ and hence $\tau_{\min}(v_j) < \sigma'(v_i)$, so the child v_j has a lower value than its parent v_i in $\sigma' \cup \tau$. This shows that $\tau_{\min}$ is not a valid completion for σ', which concludes this case of the proof.

Figure 4.4 illustrates this proof argument, showing two sorted fraternal V'-assignments σ (left) and σ' (right) together with the minimal valid completion $\tau_{\min}$ of σ. We see that $\sigma \cup \tau_{\min}$ at the left satisfies the heap condition, while $\sigma' \cup \tau_{\min}$ at the right violates it for v_6 (a variable in V' with $\sigma(v_6) < \sigma'(v_6)$) and its child v_{13}.

We now consider the remaining case, where v_i with $\sigma(v_i) < \sigma'(v_i)$ has an unassigned parent v_j ($j = \lfloor \frac{i}{2} \rfloor$). (The two cases are of course not disjoint, but they are exhaustive.) This time, we consider the maximal valid completion $\tau'_{\max}$ of σ', which by construction is a valid completion of σ'. We show that it is not a valid completion of σ by demonstrating $\tau'_{\max}(v_j) > \sigma(v_i)$, so that the heap condition is violated between v_j and its child v_i. More precisely, we will show $\tau'_{\max}(v_j) = \sigma'(v_i)$, from which the result follows with $\sigma'(v_i) > \sigma(v_i)$.

Let v_s be the sibling of v_i, i.e.the other child of v_j. By definition of maximal valid completions, we have $\tau'_{\max}(v_j) = \min((\sigma' \cup \tau'_{\max})(v_i), (\sigma' \cup \tau'_{\max})(v_s))$. If $v_s \in V'$, the values of v_i and v_s are both defined by σ', and we obtain $\tau'_{\max}(v_j) = \min(\sigma'(v_i), \sigma'(v_s)) = \sigma'(v_i)$ because σ' is fraternal and hence $\sigma'(v_i) = \sigma'(v_s)$.

If $v_s \notin V'$, then it is easy to see from the definition of maximal valid completions that $\tau(v_s)$ is a value assigned by σ' to some *descendant* v_d of v_s that belongs to V' (possibly several layers removed), or the maximal possible value D if no such descendant exists. In either case, this value is at least as large as $\sigma'(v_i)$ because $\sigma'(v_i) \leq D$ unconditionally, and if the value derives from a descendant v_d, then we must have $i < d$, from which $\sigma'(v_i) \leq \sigma'(v_d)$ follows because σ' is sorted. So also in this case we obtain $\tau'_{\max}(v_j) = \sigma'(v_i)$, which concludes the proof.

Figure 4.5 illustrates the proof argument for this latter case, showing the same sorted fraternal V'-assignments as before, but this time together with the maximal valid completion $\tau'_{\max}$ of σ'. We see that $\sigma' \cup \tau'_{\max}$ at the right satisfies the heap condition, while $\sigma \cup \tau'_{\max}$ at the left violates it for v_6, a variable in V' with $\sigma(v_6) < \sigma'(v_6)$, and its parent v_3. □

We are now ready to put the pieces together.

Theorem 4.3. *The heap predicates do not have compact linear FMs.*

Proof. We must show that the minimum representation size required by linear FMs for $\varphi_{n,D}$ grows faster than any polynomial in nD, no matter which merge strategy (variable order) is chosen. We describe a sequence of heap predicates with a size parameter $s \in \mathbb{N}_1$ for which the result can already be established. Then it of course extends to the family of heap predicates as a whole.

For $s \in \mathbb{N}_1$, we consider the heap predicate $\varphi_{n,D}$ with $k = 4s$ (and hence $n = 2^k - 1$) and $D = s \cdot n$. We observe that $D = \frac{1}{4}kn = \frac{1}{4}\log_2(n+1)n = O(n \log n)$, and hence a polynomial size bound in nD exists for the given subset of heap predicates iff a polynomial size bound in n exists. Therefore, to show the overall result, it is sufficient to show that heap predicates of the chosen form do not have linear FMs of size $O(n^c)$ for any constant c.

Let $s \in \mathbb{N}_1$, and let $k = 4s$, $n = 2^k - 1$ and $D = s \cdot n$. Let π be any variable order for $\varphi_{n,D}$.

From Lemma 4.2, π has a prefix consisting of a variable set V' with at least $\ell = \lfloor \frac{k}{2} \rfloor = \lfloor \frac{4s}{2} \rfloor = 2s$ frontier variables.

From Lemma 4.3, there exists a set $\Sigma(V')$ of sorted fraternal V'-assignments differing on at least one frontier variable, where $|\Sigma(V')| = \binom{D}{\lceil \ell/2 \rceil} = \binom{D}{\lceil 2s/2 \rceil} = \binom{D}{s}$.

From Lemma 4.4, all assignments in $\Sigma(V')$ are distinguishable from each other. Therefore, $\binom{D}{s}$ is a lower bound on the size of the value set in the sub-FM for V', and hence it is also a lower bound on the overall representation size.

We have $\binom{D}{s} \geq \left(\frac{D}{s}\right)^s = \left(\frac{s \cdot n}{s}\right)^s = n^s = n^{\frac{1}{4}k} = n^{\frac{1}{4}\log_2(n+1)}$, which grows faster than any polynomial in n, concluding the proof. □

We remark that for the application of merge-and-shrink and FMs to classical planning, we can construct planning tasks whose perfect heuristics are in 1:1 correspondence with the heap predicates $\varphi_{n,D}$. This implies that there exist families of planning tasks that can be perfectly solved in polynomial time with non-linear merge-and-shrink heuristics, but not with linear ones.

A planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ in correspondence with the heap predicate $\varphi_{n,D}$ is defined as follows:

- There are n variables that can take D numeric values, plus two additional values "unselected" (u) and "verified" (v), i.e. $\mathcal{V} = \{v_1, \ldots, n_n\}$ with $dom(v_i) = \{u, 1, \ldots, D, v\}$ for all $1 \leq i \leq n$.
- In the initial state, all variables are "unselected", i.e. $s_0 = \{v_i \mapsto u \mid 1 \leq i \leq n\}$.
- The goal is that all are "verified", i.e. $s_\star = \{v_i \mapsto v \mid 1 \leq i \leq n\}$.
- There are several types of operators:
  - Variables v_i can move from "unselected" to any numerical value x with operators o of the form $pre(o) = \{v_i \mapsto u\}$, $\mathit{eff}(o) = \{v_i \mapsto x\}$, $cost(o) = 1$.
  - Each "inner" variable, i.e. variables v_i for $1 \leq i \leq \lfloor n/2 \rfloor$, can move from a numerical value x to "verified" if the heap condition is satisfied for this variable w.r.t. its children. To do so, for all values $x < y \leq D$, there are operators o of the form $pre(o) = \{v_i \mapsto x\} \cup \{v_j \mapsto y \mid j = 2i \vee j = 2i + 1\}$, $\mathit{eff}(o) = \{v_i \mapsto v\}$, $cost(o) = 1$.
  - All "leaf" variables, i.e. variables v_i for $\lceil n/2 \rceil \leq i \leq D$, can move to "verified" unconditionally through operators o of the form $pre(o) = \emptyset$, $\mathit{eff}(o) = \{v_i \mapsto v\}$, $cost(o) = 1$.

It is easy to see that all assignments where all variables have a numerical value are reachable states from the initial state, and that $h^*(s)$ is finite in the resulting state iff the heap condition is satisfied. (Numerical values cannot be changed, and to move to a goal state, it suffices to move to "verified" for each variable in breadth-first order.)

4.5. Conclusions

In this chapter, we showed that general FMs are strictly more expressive than linear ones: a general-to-linear conversion is possible, but incurs an unavoidable super-polynomial representational blow-up. We bounded this blow-up from above and below, and the upper and lower bounds coincide up to a constant factor in the exponent: general FMs of size $\|\mathcal{M}\|$ can always be converted to linear ones of size $\|\mathcal{M}\|^{\Theta(\log |vars(\mathcal{M})|)}$ and more compact linear FMs do not exist in general. We also presented a refined upper bound in terms of the Horton-Strahler number of the merge strategy, which measures how close a given merge strategy is to the linear case.

Our results offer theoretical justification for the interest in non-linear merge-and-shrink abstractions. While several non-linear merge strategies have been presented since the introduction of generalized label reduction, all of which we describe in Chapter 5, our mostly experimental analysis of merge strategies (which we present and discuss in Section 6.6) suggests that there is still an untapped potential of non-linear merge strategies, hence motivating further research in this area.

The results of this chapter also indicate that it may be worth questioning the ubiquity of *BDD representations* (which are polynomially equivalent to linear FMs) for symbolic search in automated planning and other areas. General FMs retain many of the properties of BDDs that make them useful for symbolic search, but offer a larger than polynomial advantage over BDDs in cases where linear variable orders cannot capture information dependencies well.

Can we build strong symbolic search algorithms using general FMs instead of BDDs, replacing *variable orders* by *variable trees*? The same question has been asked about *sentential decision diagrams* (Darwiche, 2011), which extend BDDs with a different generalization from linear to tree structures and have shown much initial promise. We believe that there is significant scope for further investigations in this direction.

5. Transformation Strategies

In this chapter, we discuss most of the transformation strategies of the merge-and-shrink toolbox. The first section describes the merge strategies and the shrink strategies available with the previous theory of label reduction. In the second section, we present related work, including transformation strategies based on generalized label reduction or non-linear merge-and-shrink in general.

Having discussed related work, the remainder of this chapter deals with our own contributions, which are all related to merge strategies. In Section 5.3, we describe the framework to enhance merge strategies based on factored symmetries. Section 5.4 defines different types of merge strategies, presents three further non-linear merge strategies of different types, and discusses tie-breaking criteria for so-called score-based merge strategies.

Finally, in Section 5.5, we summarize all transformation strategies in an overview, discuss the different types of merge strategies that we defined and also suggest several directions of future work concerning merge strategies.

5.1. Merge-and-Shrink Before Generalized Label Reduction

In this section, we describe merge and shrink strategies presented before the addition of generalized label reduction. We proceed in chronological order.

The first work that adapted merge-and-shrink to planning (Helmert et al., 2007) presents a linear merge strategy as well as several variants of a shrink strategy. A linear merge strategy can easily be described as an order of the state variables of the given problem, a so called *variable order*. The merge strategy then initially selects the atomic factors corresponding to the first two variables of the order, and then iteratively merges each of the remaining atomic factors, according to the variable order, with the product of the previous merge(s). Helmert et al. (2007) use the following rules to determine the next variable in the order:

1. If possible, choose a variable from which there is an edge in the causal graph to any of the variables ordered before.

2. Otherwise, add a variable for which the goal description of the problem requires a specific value.

If ties need to be broken, the variable with the highest "level" according to the variable order of Fast Downward is chosen (Helmert, 2006). This order of choices caused later works to refer to this variable order as *causal graph goal level* or *CGGL* for short. Several variants of this rule based variable order were presented later, as we will see below.

In the context of model checking, Dräger et al. (2006) use an error-distance-preserving strategy. Helmert et al. (2007) extend this idea and define the *f-preserving* shrink strategy that preserves g- and h-values (and thus f-values) of abstract states of a transition system, whenever possible. To break ties, states with higher f-value are preferred, and among those with the same f-value, lower h-values are preferred, with the rationale that the farther away a state is from a goal state, the lesser the potential negative impact on heuristic quality through the introduction of inaccuracies by shrinking. The computation can be efficiently done by first distributing states into buckets according to their f-values and, within these buckets, according to their h-values. The buckets are then ordered according to the desired tie-breaking criteria. Finally, the shrink strategy only needs to repeatedly select the next bucket according to that order, and choose two states from the bucket to combine them. We call this strategy *F* in our experiments.

The second class of shrink strategies – still used in most state-of-the-art merge-and-shrink settings – is based on bisimulation (cf. Definition 3.14 on page 52) and due to Nissim et al. (2011). There always exists a unique coarsest bisimulation and it can be computed efficiently in the size of the transition system with the following bottom-up approach (Milner, 1990). The algorithm starts with the equivalence relation where all states are considered bisimilar, and then iteratively refines the equivalence relation as long as states which are not bisimilar are considered bisimilar by the equivalence relation. However, in practice, bisimulations are often too large to satisfy reasonable size limits N. A practical shrink strategy based on bisimulation uses the above bottom-up computation, preferring to separate non-bisimilar states closer to goal states, and stopping as soon as separating more states would result in a transition system violating the imposed size limit. Hence the result is not guaranteed to be a bisimulation, i.e. not guaranteed to be an exact transformation, but it still is a strict homomorphism. We refer to this shrink strategy as *B*.

Another variant, called *greedy* bisimulation, does not consider a transition $\langle s, \ell, s' \rangle$ when checking condition 2. of Definition 3.14 if that transition does not satisfy $h^*(s') + c(\ell) = h^*(s)$. We call this variant *G*.

Nissim et al. do not only show that bisimulation-based shrinking is exact (cf. our Theorem 3.7), but also that bisimulations can get exponentially smaller on label-reduced transition systems compared to the non-label-reduced one (the fewer differently labeled transitions there are, the more states are potentially bisimilar). In particular, they show that for several planning domains, bisimulation-based shrinking in conjunction with full label reduction yields perfect merge-and-shrink heuristics (which can be computed in polynomial time). Hence, with the addition of the more powerful generalized label reduction, we also expect bisimulation-based shrinking to improve by computing more exact bisimulations compared to using the old theory of label reduction.

Furthermore, while the computation of bisimulations is polynomial in the size of the transition system, it can still be a rather time-consuming process within the merge-and-shrink framework. With our efficient implementation that stores local equivalence relations on labels and the transitions of equivalent labels only once, the computation of bisimulations can be accelerated by not considering individual labels but entire label groups. This is possible because two states are bisimilar iff they are bisimilar with respect to all labels, and since all locally equivalent labels induce identical transitions, it suffices to consider the transitions of label groups once.

Nissim et al. also evaluated some variants of the CGGL merge strategy: *goal causal graph*

level (GCGL), switching the precedence of the two rules, and also the simple variants based on the variable order of Fast Downward, i.e. *reverse level (RL)* for the normal order and *level (L)* for the inverted order.

Another variant of bisimulation based shrinking only considers a subset of operators (respectively labels) for the computation of bisimulations (Katz, Hoffmann, & Helmert, 2012).

Hoffmann, Kissmann, and Torralba (2014) consider more variants of linear merge strategies, tailored towards detecting unsolvability of the given problem. Another application for linear merge-and-shrink heuristics is the use within symbolic search using binary decision diagrams (BDDs) (Edelkamp et al., 2012; Torralba et al., 2013). Since we use non-linear merge-and-shrink techniques and explicit search, we do not consider these linear merge-and-shrink heuristics in our experimental study.

5.2. Related Work Based on Generalized Label Reduction

We now turn our attention to related work dealing with merge-and-shrink presented after the introduction of generalized label reduction. Some of these papers are directly making use of generalized label reduction or the concept of combinable labels, and others use non-linear merge-and-shrink, thus also relying on generalized label reduction. We again proceed in chronological order.

Fan, Müller, and Holte (2014) present two merge strategies that are based on partitioning variables of the problem according to some criteria measuring their interaction. The strategies then first merge variables within the partitions, before merging the resulting transition systems to create the final transition system. The first strategy uses the weighted causal graph, where edges of the causal graph have a weight corresponding to the number of operators inducing the edge. It then computes the min-cut of the weighted causal graph, which results in two variable clusters. Iterative computation of min-cuts on these clusters yield a more and more fine-grained partitioning of variables. This strategy is called *undirected min-cut* merge strategy.

The second strategy by Fan et al. measures interaction of a set of variables by the ratio of the number of active states to the number of total states in the transition system that results from merging all atomic transition systems for each variable in the set. The strategy aims at maximizing the amount of pruning transformations, hence at minimizing the *maximum intermediate abstraction size minimizing*, from which the name *MIASM* is derived. The computation of MIASM begins with a best-first search in the space of variable subsets, ordering subsets using a criterion based on the ratio described above. To compute the ratio, the product transition system corresponding to the set of variables needs to be computed (temporarily), using only exact shrinking and the same size limit N that will later be used for the actual merge-and-shrink computation. Note that this sampling-like part of the computation is limited to small variable subsets due to the limit N. After completing the search, the resulting family of variable subsets needs to be turned into a partitioning of the variables. This is done using a greedy algorithm to compute a maximum set packing. In our experimental study, we exclusively consider MIASM, the stronger of the two merge strategies presented by Fan et al.

Torralba and Hoffmann (2015) use simulation relations for planning tasks to prune an A* search. A simulation relation is a dominance relation defined on the states of a planning problem.

Such simulation relations can be computed on an exact factored representation of the planning problem by computing simulation relations for each transition system of the factored transition system and then combining them into a single one. To obtain an exact factored representation of the planning problem, Torralba and Hoffmann start with the induced factored transition and use exact shrinking based on bisimulation and exact label reduction until some size limits are reached. Their experiments show that combining the (state) simulation relation with the concept of label dominance, which is an extension of the notion of label subsumption, even coarser dominance relations can be obtained. The combination is called label-dominance simulation.

In follow-up work, Torralba and Kissmann (2015) extend the use of simulation relations to prune operators from planning tasks and to prune transitions from merge-and-shrink heuristics. To this end, they use the notion of subsumed transitions, which is based on dominance-label simulation. Removing subsumed transitions can cause labels to become dead, and all operators that have been reduced to such dead labels through label reduction can safely be removed from the planning task in the sense that at least one optimal plan is preserved. Alternatively, pruning subsumed transitions can also be used during the merge-and-shrink computation (under certain conditions) and yield smaller transition systems, hence allowing more informed merge-and-shrink heuristics to be computed under the given time and memory constraints. Another application of label-dominance simulation is shrinking, but Torralba and Kissmann observe that if no subsumed transitions occur, simulation based shrinking behaves like bisimulation-based shrinking.

The most recent work on merge-and-shrink is due to Fan, Müller, and Holte (2017). They experimentally observe that merge-and-shrink suffers from diverse action costs in several non-unit-cost planning domains, compared to the unit-cost variant of these domains. The reason is that exact label reduction only allows combining labels of the same cost, which often hinders label reductions in non-unit-cost domains. To allow for more label reductions also on non-unit-cost domains, Fan et al. suggest a cost partitioning for label costs called delta cost partitioning.

Definition 5.1 (Delta Cost Partitioning, Fan et al., 2017)**.** *Let L be a set of labels and c a label cost function for L. Let $c_0 = 0$ and $0 < c_1 < c_2 < \cdots < c_n$ be the n different positive cost values to which the labels in L are mapped by c, and $\Delta i = c_i - c_{i-1}$ for $1 \leq i \leq n$.* Delta cost partitioning *divides the costs $c_1, c_2, \ldots, c_n$ among n delta cost functions $C_1, C_2, \ldots, C_n$ as follows. For $1 \leq i \leq n$, $C_i(\ell) := 0$ if $c(\ell) < c_{i-1}$, and $C_i(\ell) := \Delta_i$ otherwise, i.e. if $c(\ell) \geq c_i$.*

In words, a delta cost function C_i is associated with a value c_i of the image of the original cost function c. C_i maps each label ℓ to a cost of 0 if its original cost is lower than c_i and to the difference $c_i - c_{i-1}$ otherwise. For each delta cost function, Fan et al. compute a separate merge-and-shrink heuristic. Due to delta cost partitioning, they are additive, and since each delta cost function maps to only two values (0 or Δ_i), each single merge-and-shrink abstraction can profit from more exact label reductions.

5.3. Factored Symmetries of Factored Transition Systems

In this section, we describe the concept of factored symmetries and how to enhance merge strategies through using factored symmetries. Most parts of this section are based on our original work introducing factored symmetries (Sievers, Wehrle, Helmert, Shleyfman, & Katz, 2015).

The main motivation behind the investigation of symmetries for merge-and-shrink is to combine two successful techniques: merge-and-shrink abstractions and state-space pruning techniques based on symmetries. Current approaches find (structural) symmetries based on factored planning task representations called *problem description graphs*, or *PDGs* for short (Pochter et al., 2011; Shleyfman et al., 2015). We generalize this concept to *factored symmetries* based on factored transition systems, study the special cases of *local* and *atomic* symmetries and explore their relationship to bisimulation and label reduction. Finally, we devise a framework to enhance existing merge strategies to maximize the application of symmetry reduction.

5.3.1. Factored Symmetries

We begin by introducing factored symmetries, which are defined on factored transition systems F. To simplify notation, we assume throughout the chapter that states of different transition systems in F can always be distinguished, i.e. if S^i and S^j are state sets of different transition systems in F, then $S^i \cap S^j = \emptyset$. If σ is a function defined on states and labels, we extend it in the natural way to sets of states ($\sigma(X) = \{\sigma(x) \mid x \in X\}$), sets of sets of states ($\sigma(\mathcal{X}) = \{\sigma(X) \mid X \in \mathcal{X}\}$) and transitions ($\sigma(\langle s, \ell, s' \rangle) = \langle \sigma(s), \sigma(\ell), \sigma(s') \rangle$).

Definition 5.2. *Let $F = \{\Theta^1, \ldots, \Theta^n\}$ be a factored transition system, where $\Theta^i = \langle S^i, L, c, T^i, s_0^i, S_\star^i \rangle$ for all $1 \leq i \leq n$. A* factored symmetry *of F is a permutation σ of the set $\bigcup_{i=1}^n S^i \cup L$ that maps states to states and labels to labels such that*

1. $\sigma(\{S^1, \ldots, S^n\}) = \{S^1, \ldots, S^n\}$,
2. $\sigma(\bigcup_{i=1}^n S_\star^i) = \bigcup_{i=1}^n S_\star^i$,
3. $\sigma(\bigcup_{i=1}^n T^i) = \bigcup_{i=1}^n T^i$, *and*
4. $c(\sigma(\ell)) = c(\ell)$ *for all* $\ell \in L$.

The four conditions guarantee that a factored symmetry preserves the grouping of states into transition systems, the goal state property, the transition structure, and label costs. We illustrate the definition with the example in Figure 5.1, corresponding to a planning task with four variables a, b, c and d and uniform-cost operators $o_{x,y}$ for certain pairs of $x, y \in \{a, b, c, g\}$ that change x from 1 to 0 and y from 0 to 1. The mapping σ that cyclically rotates all facts and operators related to variables $\{a, b, c\}$ is a factored symmetry.

When applied to the atomic factors of a planning task, Definition 5.2 is equivalent to the definition of PDG symmetries or structural symmetries in earlier works. However, it is not limited to this case. On the opposite end of the spectrum, the case $F = \{\Theta(\Pi)\}$ captures a semantic notion of state space symmetry in a non-factored representation.

More generally, the definition can be applied to any intermediate result in the computation of a merge-and-shrink abstraction. A factored symmetry σ of F naturally induces a permutation $\sigma^\otimes$ on the states $S^\otimes$ of the product $\bigotimes F$, and Definition 5.2 guarantees that $h^*(\sigma^\otimes(s)) = h^*(s)$ for all $s \in S^\otimes$.

Like other notions of symmetry, factored symmetry is closed under composition (if σ and τ are factored symmetries, then so is $\sigma \circ \tau$) and hence induces a group structure. If Γ is a set of

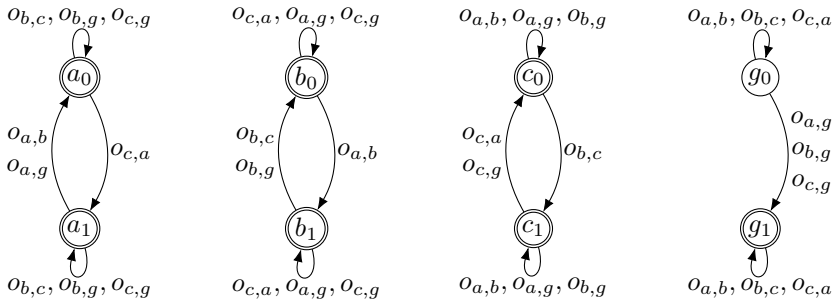


Figure 5.1.: Factored symmetry example: rotating $a \mapsto b \mapsto c \mapsto a$ in all abstract states (e.g., $\sigma(b_1) = c_1$) and labels (e.g., $\sigma(o_{a,b}) = o_{b,c}$) is a factored symmetry.

factored symmetries of F, then we say that two states s and s' of $\bigotimes F$ are *symmetric under* Γ, in symbols $s \sim^{\Gamma} s'$, if $\sigma^{\otimes}(s) = s'$ for some factored symmetry σ in the group generated by Γ. This is an equivalence relation, and states in the same equivalence class always share the same h^* value. The factored symmetry in Figure 5.1 induces an equivalence relation showing that e.g. all states containing g_0 and *exactly one* of $\{a_1, b_1, c_1\}$ are symmetric.

In the special case where Γ is the set of all factored symmetries of F, we simplify this notation to $s \sim s'$ and say that s and s' are *symmetric under factored symmetry*.

To compute factored symmetries, we can use PDGs as defined by Shleyfman et al. (2015), conceptually adapted to have the following nodes: variables (of the planning task) are replaced by factors of the factored transition system; values (of variables) are replaced by states of factors, treating goal states like goal values of variables; operators are replaced by labels, and their transitions are treated the same way. The nodes are then connected like in regular PDGs: states are connected to their transition systems, transitions are connected to their source and target states as well as to the inducing label. Using the PDG as input to any off-the-shelf graph automorphism tool, we can then compute factored symmetries as automorphisms of the PDG.

We remark that the optimized implementation (cf. Section 3.8.3) also has an impact on this computation of factored symmetries: similarly to considering label groups rather than labels when computing bisimulations, we only need to represent label groups instead of labels in the PDG, which considerably reduces the number of transition nodes. This reduces the computation time of graph automorphisms especially on factored transition systems that contain many factors or factors of large sizes.

5.3.2. Interaction with Merge-and-shrink Transformations

We want to combine reasoning about symmetries with the manipulation of transition systems in the merge-and-shrink framework. It is then natural to ask how factored symmetries interact with the transformations of the merge-and-shrink toolbox.

It is easy to see that the interaction between *shrinking* and factored symmetries is in general unpredictable. For example, we may have $s \sim s'$ for states s, s' of a factored transition system F, but $\tau(s) \not\sim \tau(s')$ for the corresponding abstracted states after applying a shrink transformation τ. This happens when a shrinking step "breaks" an existing symmetry, e.g. by combining a_0

and a_1 in Figure 5.1. The converse case is also possible: we may have $s \not\sim s'$ before shrinking, but $\tau(s) \sim \tau(s')$ after shrinking if the abstraction removes the obstacles to the symmetry of s and s'. This should come as little surprise because shrinking can transform a transition system in essentially arbitrary ways. We will discuss the special case of bisimulation-based shrinking later.

Perhaps somewhat surprisingly, *merging* can also affect the symmetry properties of F unpredictably. To see this, observe that the symmetry property between states $a_1b_0c_0g_0$ and $a_0b_0c_1g_0$ is lost when merging the two transition systems at the left of Figure 5.1 (which shows that symmetry can be lost by merging), and the same symmetry is recovered by then merging this product by the third transition system (showing that symmetries can be gained by merging).

Concerning *pruning*, the interaction with factored symmetries depends on which type of pruning is used. Since factored symmetries are goal-stable automorphisms, pruning irrelevant states cannot break factored symmetries (under the assumption that all irrelevant states and not only a subset of them are pruned): a factored symmetry cannot map an irrelevant to a relevant state, due to the path-preserving property of automorphisms (the relevant state has a path to some goal state while the irrelevant does not, and goal states are mapped to goal states only). Hence, if two irrelevant states are symmetric, they are still symmetric after being pruned because they then are identical ($\bot$ are mapped to $\bot$). However, pruning irrelevant states can result in finding factored symmetries that did not exist before: if a factored transition system does not exhibit a factored symmetry only because of two irrelevant states that have a single transition between them but that are otherwise "symmetric", then, after mapping these two states onto one isolated state, there is a factored symmetry of the transition system, that among others maps the isolated state onto itself.

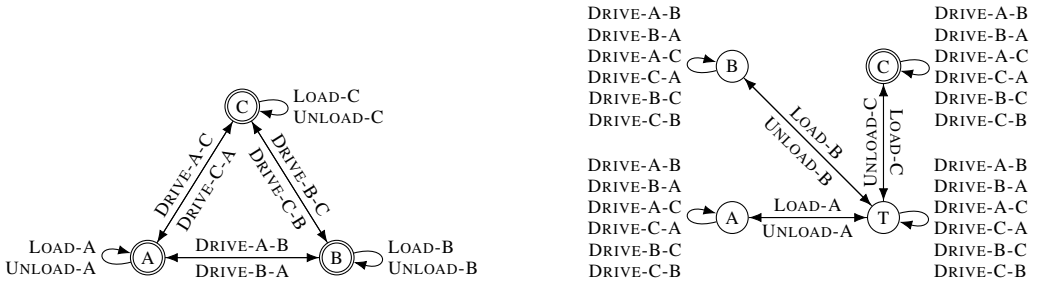
Pruning unreachable states (again assuming that all of them and not only a subset are pruned) can affect the symmetry properties of a factored transition system unpredictably because factored symmetries do not stabilize the initial state. Hence there exist factored symmetries that map an unreachable to a reachable state, and such a symmetry gets lost after pruning and thus mapping the unreachable state to an isolated state. The converse case is also possible for the same argumentation of why pruning irrelevant states can remove obstacles to a factored symmetry.

Finally, the interaction of *label reduction* and symmetry properties of factored transition systems is rather simple: reducing labels clearly cannot break existing symmetries (the structure of the transition systems does not change), but it can remove obstacles to factored symmetries. To see the latter, consider a factor with two states s, t, and two transitions $s \xrightarrow{\ell_1} t$, $t \xrightarrow{\ell_2} s$. Without label reduction, $s \not\sim t$, but after reducing labels ℓ_1 and ℓ_2 to a new label ℓ, $s \sim t$.

In summary, these considerations show that it can be beneficial to search for new symmetries after each transformation step in the construction of merge-and-shrink heuristics.

5.3.3. Local and Atomic Symmetries

Non-factored state spaces have the property that combining symmetric states (formally: abstracting by mapping each state to its orbit in the symmetry group) preserves optimal goal distances. It is thus natural to attempt using *factored symmetries* for information-preserving shrinking in the merge-and-shrink framework. The first obstacle to this is that shrinking happens at the level of



(a) Atomic factor for v_T.

(b) Atomic factor for v_P.

Figure 5.2.: Induced factored transition system of the example planning task of Figure 2.1.

individual transition systems $\Theta \in F$, while symmetries like the one in Figure 5.1 can critically rely on relationships between different transition systems. We thus now consider a subclass of symmetries that "stay within" the individual abstractions.

Definition 5.3. *A factored symmetry σ of a factored transition system F* stabilizes *$\Theta \in F$ if σ maps states of Θ to states of Θ. A factored symmetry is* local *if it stabilizes all $\Theta \in F$.*

Local symmetries are closed under composition and hence form a subgroup of the group of factored symmetries. Since local symmetries stay within each abstraction in F, they can be used to define an equivalence class on *abstract states*: for all $\Theta \in F$ and all states s, s' of Θ, we set $s \sim s'$ if there exists a local symmetry σ with $\sigma(s) = s'$.

By analogy to the use of symmetries in the global state space, it may appear natural to use this equivalence relation as a basis for shrinking by applying the state mapping based on the equivalence relation $\sim$ to *all* factors of F.[1] However, such a use of symmetries is *not* exact, i.e. can lead to a loss of precision in the resulting merge-and-shrink heuristic.

Proposition 5.1. *Let F and F' be factored transition systems and let σ be a local symmetry of F. Then a (shrink) transformation τ_{F} of F into factored transition system F' based on σ does not necessarily satisfy* $\mathbf{REF_T}$*, and hence is not necessarily exact.*

To prove the proposition, we give an example where shrinking based on local symmetries before merging results in an imperfect heuristic. We use our running example, the simple logistics planning task of Figure 2.1 in the background section. For convenience, we again show the atomic factors $\Theta(v_T)$ and $\Theta(v_P)$ of the induced factored transition system in parts (a) and (b) of Figure 5.2. Recall that all operator costs are 1 and the goal of the package is to be at C. In the context of factored symmetries, the initial state can be chosen arbitrarily.

There is a local symmetry σ that swaps the role of locations A and B everywhere (e.g., $\sigma(\mathrm{A}) = \mathrm{B}$ both for the values of variable v_T and v_P). A shrink transformation τ_{F} based on σ would thus

[1] This does not exactly match our definition of shrink transformations, cf. Definition 3.13 on page 48, which are defined on a state mapping applied to a single factor, keeping all other factors as they are. However, the definition can easily be extended to be based on a set of state mappings affecting several or even all factors of the given factored transition system. It is easy to see that such a shrink transformation still is a strict homomorphism, since it can be expressed as a sequence of shrink transformations affecting single factors as usually.

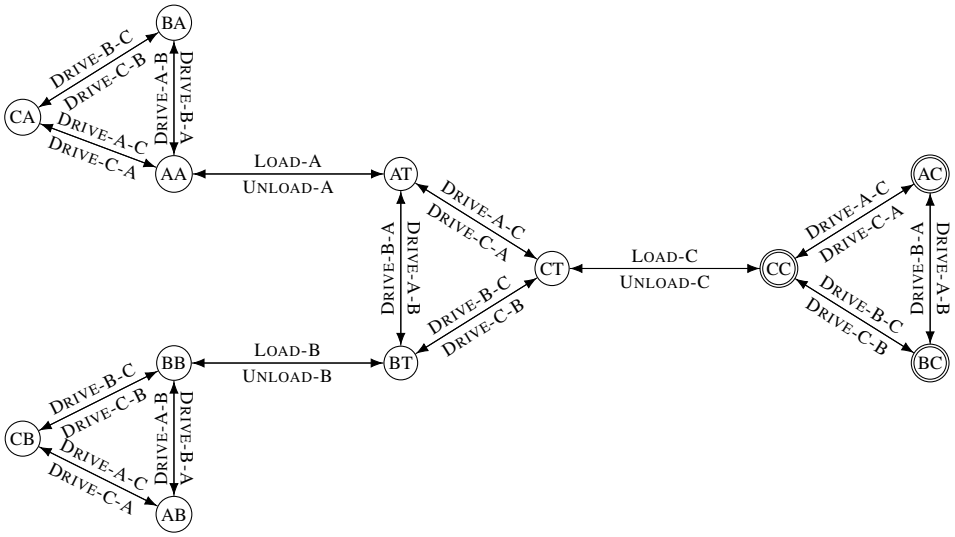


Figure 5.3.: Product of the induced factored transition system in Figure 5.2 (hence induced transition system of the example planning task of Figure 2.1).

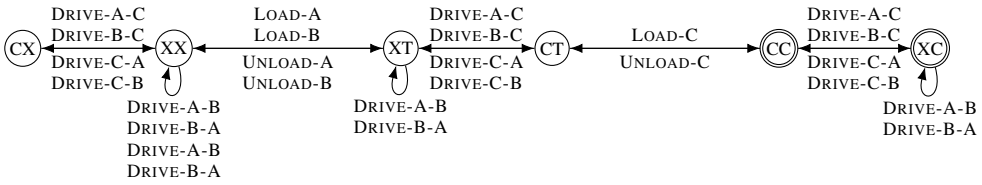


Figure 5.4.: Product of the induced factored transition system of Figure 5.2 after shrinking both factors according to local symmetry σ that permutes locations A and B, combining values A and B into X.

combine A and B to a common abstract state, say X ("truck at A or B" and "package at A or B").[2]

We now compare the product transition system $\bigotimes F$ which is the induced factored transition system of the planning task, also shown again in Figure 5.3 for convenience, to the product transition system $\bigotimes F'$ obtained by performing the transformation τ_{F} on F, shown in Figure 5.4. We see that the transformation causes a loss in precision: for example, it leads to the estimate $h^{\tau_{\mathrm{F}}}_{\bigotimes F}(\mathrm{BA}) = h^*(\sigma(\mathrm{BA})) = h^*(\mathrm{XX}) = 3$ instead of the correct $h^*_{\bigotimes F}(\mathrm{BA}) = 4$. Intuitively, even though locations A and B are *locally* symmetric in $\Theta(v_T)$ and $\Theta(v_P)$, they cannot simply be combined: this loses the distinction between states where the truck and package are at the same location and those where they are not.

More formally, the reason why the heuristic induced by a shrink transformation based on local symmetries is not exact is that it violates $\mathbf{REF_T}$, as stated by the proposition. To see

[2]Note that this transformation is almost the same as the one we used in the background section when illustrating the computation of FMs: compared to the example here, the product FM there also combines the states CC and XC into a single state, hence dropping the distinction of all locations for the truck if the package is at C.

this, consider e.g. the transition AA $\xrightarrow{\text{LOAD-A}}$ AT $\in F'$. There does not exist a corresponding transition for all preimages of AA in F: for example for the state BA for which the heuristic is not perfect anymore, there is no transition labeled LOAD-A (and hence no such transition that would lead to a preimage of AT as required by the property $\mathbf{REF_T}$).

This problem can be resolved by further restricting the notion of local symmetries to *atomic* symmetries.

Definition 5.4. *A factored symmetry σ of a factored transition system F* affects *$\Theta \in F$ if there exists a state s of Θ with $\sigma(s) \neq s$. A factored symmetry is* atomic *if it affects at most one transition system $\Theta \in F$.*

Clearly atomic symmetries are a subset of local symmetries. Unlike local symmetries, they do not form a group: the composition of atomic σ_1 and σ_2 is non-atomic if they affect different transition systems. However, for any *fixed* Θ, the atomic symmetries affecting at most Θ form a group.

Proposition 5.2. *Let F and F' be factored transition systems and let σ be an atomic symmetry of F. Then a shrink transformation τ_{F} of F into factored transition system F' based on σ is exact induced.*

We remark that σ induces an arbitrary state mapping for some $\Theta_{\text{shrink}} \in F$ and identity state mappings for all other $\Theta \in F$ with $\Theta \neq \Theta_{\text{shrink}}$. Hence τ_{F} based on σ is a shrink transformation according to our Definition 3.13, and as such is a strict homomorphism according to Theorem 3.13. Instead of directly proving the missing two properties $\mathbf{REF_T}$ and $\mathbf{REF_G}$ for the transformation to be exact induced, we postpone the proof of this proposition, as it follows from a more general result in the following section.

The fact that atomic symmetries allow for shrinking strategies that maintain perfection makes them particularly attractive, and we would like to exploit this result even for abstractions that are not atomic. The following result shows that we can achieve this by merging all transition systems affected by a given symmetry.

Proposition 5.3. *Let σ be a factored symmetry of a factored transition system F that affects (exactly) the transition systems $F^\sigma \subseteq F$. Let F' be the factored transition system obtained from F by merging all transition systems in F^σ. Then there exists an atomic symmetry σ' of F' that induces the same symmetry on $F^\otimes$ as σ.*

Proof. Let $F = \{\Theta_1, \ldots, \Theta_n\}$, and let $k \in \{1, \ldots, n\}$ such that σ affects exactly the transition systems Θ^i with $i \leq k$. We get $F' = \{\Theta', \Theta_{k+1}, \ldots, \Theta_n\}$ with $\Theta' = \bigotimes_{i=1}^{k} \Theta_i$. States of Θ' can be written as sets of the form $\{s_1, \ldots, s_k\}$, where each s_i is a state of Θ_i. (It is more common to use tuples instead of sets, but the two representations are equivalent because we require states of different transition systems to be disjoint. Using sets simplifies the definition of σ'.)

We define $\sigma'(\ell) = \sigma(\ell)$ for all labels ℓ, $\sigma'(s_j) = s_j$ for all states s_j of $\Theta_{k+1}, \ldots, \Theta_n$, and finally $\sigma'(\{s_1, \ldots, s_k\}) = \{\sigma(s_1), \ldots, \sigma(s_k)\}$. It is easy to verify that σ' satisfies all properties of symmetries (because σ does) and that it induces the same symmetry on $F^\otimes$ as σ. Also, it is clearly atomic, affecting only Θ'. □

5.3.4. Relationship to Bisimulation

Symmetry and bisimulation are similar concepts, as both identify (possibly smaller) structures with equivalent behavior. Hence, a natural question is to ask about their relationship. For non-factored transition systems, it is well-known that every symmetry induces a bisimulation (Clarke et al., 1999). It is also easy to see that the converse does not hold. The logistics task example shows that nontrivial factored symmetries, even local ones, do not necessarily induce nontrivial bisimulations on the individual transition systems. (There are no nontrivial bisimulations in the atomic factors of the example.)

We now show that for *fully label-reduced* factored transition systems (i.e. where no two labels are combinable for any transition system), *atomic* symmetries are captured by bisimulation.

Proposition 5.4. *Let F be a fully label-reduced factored transition system, and let $\Theta \in F$. Let $\sim$ be the equivalence relation on Θ induced by the atomic symmetries of F affecting at most Θ. Then $\sim$ is a bisimulation of Θ.*

Proof. Let S be the states of Θ. We show that $\sim \subseteq S \times S$ satisfies the two properties of bisimulations (cf. Definition 3.14 on page 52): (1) if $s \sim t$, then neither or both of s and t are goal states; (2) if $s \sim t$ and $s \xrightarrow{\ell} s' \in \Theta$, then $t \xrightarrow{\ell} t' \in \Theta$ for some t' with $s' \sim t'$.

Consider $s, t \in S$ with $s \sim t$. Then there exists a local symmetry σ of F affecting at most Θ such that $\sigma(s) = t$. From the goal-preserving property of symmetries, we get (1). To show (2), consider $\ell \in L$ and $s' \in S$ such that $s \xrightarrow{\ell} s' \in \Theta$. From the definition of symmetry, we get $\sigma(s) \xrightarrow{\sigma(\ell)} \sigma(s') \in \Theta$. Defining $t' = \sigma(s')$, we obtain $t \xrightarrow{\sigma(\ell)} t' \in \Theta$ and $s' \sim t'$. To complete the proof, we will show $\sigma(\ell) = \ell$ and hence $t \xrightarrow{\ell} t' \in \Theta$, which proves (2).

Consider any of the *other* transition systems $\hat{\Theta} \in F \setminus \{\Theta\}$. Because σ is an atomic symmetry affecting at most Θ, it maps each state of $\hat{\Theta}$ to itself. Hence, σ maps each $\hat{s} \xrightarrow{\ell} \hat{s}' \in \hat{\Theta}$ to $\sigma(\hat{s}) \xrightarrow{\sigma(\ell)} \sigma(\hat{s}') = \hat{s} \xrightarrow{\sigma(\ell)} \hat{s}' \in \hat{\Theta}$. In other words, every transition with label ℓ in $\hat{\Theta}$ has a parallel transition with label $\sigma(\ell)$. The converse also holds because σ^{-1} is also an atomic symmetry affecting at most Θ. This shows that ℓ and $\sigma(\ell)$ are locally equivalent in $\hat{\Theta}$.

This local equivalence holds for all $\hat{\Theta} \neq \Theta$, and hence ℓ and $\sigma(\ell)$ are Θ-combinable. Moreover, we have $cost(\sigma(\ell)) = cost(\ell)$ because σ is a factored symmetry. Therefore, if $\sigma(\ell) \neq \ell$, the two labels satisfy the two requirements for exact label reduction, which contradicts our requirement that F is fully label-reduced. Hence we must have $\sigma(\ell) = \ell$, concluding the proof. □

The proof for Proposition 5.2 that we postponed now follows as a corollary because shrinking based on bisimulation is exact induced (cf. Theorem 3.7 on page 53). More importantly, the result shows that existing bisimulation-based shrink strategies *automatically* capture redundancies exploitable by atomic symmetries when using full label reduction.

5.3.5. Framework for Enhancing Merge Strategies

In this section, we propose a general algorithm for including symmetry reasoning in the merge-and-shrink framework. We have seen that shrinking based on atomic symmetries is information-preserving, while shrinking based on other classes of factored symmetries is not. We exploit this information by tailoring the *merge strategy* to force the occurrence of atomic symmetries.

Specifically, if we detect a factored symmetry σ affecting k transition systems and then merge these transition systems, σ becomes an atomic symmetry only affecting the merged system.

We have also seen that state-of-the-art approaches using bisimulation-based shrinking and label reduction based on Θ-combinability automatically exploit atomic symmetries, so there is no need to adapt the shrink strategy. Finally, we have seen that new symmetries can arise at any time during the merge-and-shrink process, so it can pay off to search for new factored symmetry in every iteration of the merge-and-shrink loop.

Algorithm 5 The selection of a next pair with a symmetry-enhanced merge strategy.

Input: Factored transition system F and the set of associated FMs Σ, to-be-enhanced merge strategy MS.

Output: Pair of transition systems of F (and their associated FMs of Σ).

```
1:  function SELECT(F, Σ)
        ▷ Z: a set of transition systems to be merged such that their product exhibits an atomic
          factored symmetry.
2:      if |Z| < 2 then
3:          Compute a set Σ of non-atomic symmetries of F
4:          if Σ ≠ ∅ then
5:              Let Z := {Θ ∈ F | Θ is affected by one chosen σ ∈ Σ}
6:          end if
7:      end if
8:      if |Z| ≥ 2 then
9:          Choose Θ1, Θ2 ∈ Z
            ▷ Remove the factors that will be merged, keep reference to the to-be-computed
              product.
10:         Z ← Z \ {Θ1, Θ2} ∪ {Θ1 ⊗ Θ2}
11:     else
12:         Choose Θ1, Θ2 according to basic merging strategy MS
13:     end if
14:     return Θ1, Θ2 and associated FMs in Σ
15: end function
```

We explain how the approach to enhance merge strategies through symmetries works by explaining the SELECT function of the merge-and-shrink algorithm (cf. Algorithm 3 on page 70), which is the "representing" function of any merge strategy, in Algorithm 5. It augments an existing merge strategy MS as follows: initially, let $Z := \emptyset$. Whenever we are currently not pursuing any (previous) merge policy based on symmetries (line 2 triggers), we compute factored symmetries of F and, if any are found, store all factors affected by a chosen symmetry in Z (line 5). Hence, in any future call to SELECT, the next pair of transition systems to be merged is chosen either according to the set Z, if it contains at least two elements (line 8), or according to MS (line 12). In the former case, the set Z needs to be updated to contain the product after the merge-and-shrink algorithm has performed the merge transformation.

Two design choices remain: the choice of one non-atomic symmetry σ of the set Σ (line 5) and the decision which pair of transition systems to choose from Z (line 10). Regarding the first

choice, we consider selecting a symmetry which affects the least or the most factors of F. We call the former variant *smallest* and the latter variant *largest*.

Concerning the second choice, we settled on the following non-linear policy: we first merge all transition systems which are non-locally affected (i.e. mapped onto other abstractions) separately for each such cycle of mapped abstractions. After these merges, the induced symmetry is local, and we linearly merge all remaining factors affected by the symmetry. An alternative we will evaluate is to use the fallback merge strategy for merging all transition systems affected by the chosen symmetry. While this is simple for the score-based merge strategies, it also works with linear merge strategies using the technique that maps each variable to the transition system "representing" it, i.e. the transition system whose associated FM contains the variable in its associated variables. Computing a merge tree with MIASM for a factored transition system that is not the induced factored transition system (i.e. containing only atomic factors associated with the variables of the planning task) is not possible, hence we cannot use this alternative approach if MIASM is the fallback merge strategy. We call this variant FB for fallback in the experiments.

A third choice that is not directly visible from the algorithm is the resources we want to spend for computing symmetries. As a baseline, we consider computing factored symmetries only once, on the induced factored transition system. That is, we use structural symmetries for enhancing the merge strategy but do not attempt recomputing symmetries during later states of the merge-and-shrink computation. The second variant we consider is to use a budget (60s) for the total time required by symmetry computations. That means that we can search for new factored symmetries in every iteration of the merge-and-shrink algorithm as long as the time budget allows.

We remark that while this approach of enhancing merge strategies guarantees that after merging all factors affected by a chosen non-atomic symmetry, the resulting product system is the only factor affected by a now-atomic symmetry, this only works if no inexact transformations are applied to the factored transition systems in the meantime. For typical merge-and-shrink configurations, this requirement is fulfilled as long as no shrinking or pruning of unreachable states is performed. Unfortunately, for symmetries affecting a large number of factors or for large factors, we expect that shrinking will be necessary in many cases, and if using pruning, pruning opportunities will often arise. Hence, the positive impact of merging towards obtaining an atomic symmetry is potentially reduced. However, even if we have to shrink or prune before obtaining an atomic symmetry, we may not necessarily break the symmetry we are aiming for. In our experiments, we shed some light on how many times non-atomic symmetries can be turned into atomic ones successfully because no interfering transformations need to be applied.

5.4. More Merge Strategies of Different Types

This section describes the three non-linear merge strategies that we developed after the introduction of generalized label reduction (excluding the symmetry-enhancing framework which we already discussed in the previous section). Additionally, it introduces tie-breaking criteria for a simple type of merge strategies that we call score-based merge strategies. While we described most of the ideas presented here in our analysis of merge strategies (Sievers, Wehrle, & Helmert, 2016), we extend some of the ideas and provide more details and intuition here.

Before going into medias res, we discuss the different *types* of merge strategies we consider. Looking again at the merge strategies we have seen so far, i.e. the linear merge strategies and the two non-linear merge strategies by Fan et al. (2014), we see that all of them are what we call *precomputed merge strategies* because they compute the entire merge tree before the start of the merge-and-shrink algorithm, hence fixing a unique merge order up-front.[3] At the other extreme of the spectrum of merge strategies are those that do not store any information, but rather select the next pair of transition systems exclusively based on the current factored transition system maintained by the merge-and-shrink algorithm. We call these merge strategies *stateless merge strategies* because they do not need to memorize any information across different merge decisions.

In the next section, we present two so-called *score-based merge strategies*, which are stateless merge strategies that select a pair of transition systems based on ranking all candidate pairs by computing scores for each pair. We consider score-based and stateless merge strategies to be interchangeable, since we think that any stateless merge strategy can be cast as a score-based merge strategy that makes its decision based on a ranking of pairs of transition systems.

5.4.1. Two Score-based Merge Strategies

In their work introducing merge-and-shrink in the context of model checking, Dräger et al. (2006) also describe a "composition strategy" that selects which two processes to compose in each iteration of their algorithm to compute an abstract process. We adapt this composition strategy to planning, and name it *DFP merge strategy* after the original authors. We remark that DFP was the first non-linear merge strategy introduced for planning (Sievers et al., 2014), however it was never described in much detail in previous work on planning, and hence we do so in the following.

DFP is a score-based merge strategy that prefers selecting pairs of transition systems that have transitions close to goal states that synchronize in the product, i.e. transitions of both transition systems that are labeled by the same labels. The hope behind that approach is that the product transition system is an abstraction fine-grained in the goal region, which is arguably more important than regions far away from the goal, since the latter are less likely to be searched by forward searches like A*. In the following, we formally define the score that DFP computes for pairs of transition systems, but first, we need a few more definitions.

Definition 5.5 (Relevant Label)**.** *Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system. A label $\ell \in L$ is* irrelevant *in Θ if there exists exactly one transition $s \xrightarrow{\ell} t \in T$, and $s = t$. Label ℓ is* relevant *in Θ if it is not irrelevant in Θ.*

Labels that induce exactly self-looping transitions for all states of a transition systems are called irrelevant. The intuition is that they do not prevent any transitions from being synchronized in product systems, but they also do not add any useful information to the transition system. We write $L(\Theta)$ for the subset of relevant labels in a transition system Θ.

Definition 5.6 (Goal-relevant Transition System)**.** *A transition system Θ with states S and goal states $S_\star$ is* goal-relevant *if $S_\star \neq S$.*

[3] A merge tree uniquely defines a merge order if assuming a fixed policy such as always choosing the first two sibling leaf nodes according to a pre-order traversal.

Transition systems that do not only have goal states are called goal-relevant because only if there are non-goal states, heuristics induced by the transition system can be non-trivial. In the context of factored transition systems, if there are no goal-relevant factors, then also the product is not goal-relevant, hence only inducing trivial heuristics.

We now turn towards the metrics used by the DFP merge strategy.

Definition 5.7 (DFP Label Rank)**.** *Let $\Theta = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system and $\ell \in L$ be some label. The* DFP label rank *(label rank for short) of ℓ in Θ is defined as $rank(\ell, \Theta) = \min\{h^*(t) \mid t \in S, \exists s \xrightarrow{\ell} t \in T\}$.*

In words, the label rank of some label in a transition system is the minimum goal distance of any state that can be reached with a transition labeled with that label. In some sense, the rank quantifies how close a label can bring us to some goal state, from no specific state. The lower a label rank, the "better" it is with the intuition of DFP explained above. We remark that if a transition system is not goal-relevant, then all labels have the best rank of 0 in that transition system, since all states are goal states.

Definition 5.8 (DFP Score)**.** *Let Θ_1 and Θ_2 be two transition systems. The* DFP score *(score for short) of Θ_1 and Θ_2 is defined as $score(\Theta_1, \Theta_2) = \min\{\max\{rank(\ell, \Theta_1), rank(\ell, \Theta_2)\} \mid \ell \in (L(\Theta_1) \cap L(\Theta_2))\}$. If $(L(\Theta_1) \cap L(\Theta_2)) = \emptyset$, we define $score(\Theta_1, \Theta_2) = \infty$.*

The score of a pair of transition systems is the minimum, considering only labels that are relevant in both Θ_1 and Θ_2, over the maximum rank of the label in both transition systems. Intuitively, a label is only "good" if it has a low label rank in *both* transition systems (hence the maximization over the ranks) and if it is relevant in both transition systems (hence the score of ∞ if this is not the case). The reason for the latter is that if otherwise a label is irrelevant in both transition systems, it does not add non-self-looping transitions to the product system, and if it is relevant only in one of the transition systems, any path labeled in that transition system induces a path of the same cost in the product, hence not adding any information useful for heuristics.

Finally, given a factored transition system F, the DFP merge strategy is defined to select a pair of transition systems with minimum score among all pairs of transition systems in F where at least one component is goal-relevant. The restriction to goal-relevant transition systems is necessary to avoid choosing pairs where all common relevant labels have a rank of 0 but do not contribute any information to induced heuristics.

After its introduction, DFP quickly established as a state-of-the-art merge strategy, only challenged by MIASM. Since we consider score-based merge strategies to be easier to understand, used and hence analyzed than (complex) precomputed merge strategies like MIASM (also see the discussion in Section 5.4.3), we also devise a second score-based merge strategy. It is inspired by MIASM and serves as a score-based alternative to the precomputed original. *Score-based MIASM (sbMIASM)*[4] follows the very simple idea that also underlies the original MIASM strategy: it prefers merging pairs of transition systems such that their products have many unreachable or irrelevant states, which maximizes pruning opportunities. This can be achieved easily within the score-based merge strategies framework: (temporarily) compute the products

[4] We called it DYN-MIASM in our original paper (Sievers et al., 2016), as it is a "dynamic" merge strategy, i.e. a merge strategy we call stateless in this thesis.

of all pairs of transition systems, possibly shrink them like the merge-and-shrink computation would do, and compute the percentage of alive states in these products. This percentage serves as the score that should be minimized: the smaller the score, the more states can be pruned.

5.4.2. Tie-breaking-Criteria for Score-Based Merge Strategies

Looking at the definition of the DFP score, it is clear that the DFP merge strategy does not select a uniquely determined pair of transition systems, since there may be several pairs of a factored transition system that have the best score of 0. In particular, this happens very easily for typical planning benchmarks with few goal variables: in that case, there are lots of factors that are not goal-relevant and whose relevant labels all have a score of 0. That means that even pairs where at least one transition system is goal-relevant often have a score of 0 (maximizing the label ranks only helps if *all* common relevant labels have a non-zero rank in the goal-relevant transition system, since DFP minimizes over all common relevant labels). The same observation of possible ties also holds for sbMIASM, where several candidate pairs of transition systems can allow for the same amount of pruning in their products.

In the first implementation of DFP within Fast Downward, if several pairs of transition systems have the same score, the strategy simply chooses the "first" pair according to some order on the transition systems, basically dictated by the variable order of Fast Downward. To evaluate the choice of this tie-breaking on score-based merge strategies more generally, we consider several alternatives for fixing a total order on all transition systems, which includes both the n initial atomic factors and all $n - 1$ (intermediate) product transition systems of the merge-and-shrink computation. Such a total order on transition systems directly induces a total order on pairs of transition systems, which makes the selection of a score-based merge strategy uniquely defined if using the policy to select the first-ordered pair in case of ties.

We consider three parameters to determine the order of transition systems: whether to prefer atomic (*PA*) or to prefer composite (*PC*), i.e. non-atomic transition systems, the order in which atomic transition systems are considered, and the order in which composite transition systems are considered. For the order of atomic transition systems, the options include the variable orders RL, L, and a randomized order (RND), and for the order of composite transition systems, the options include *new to old (NTO)* and *old to new (OTN)*, preferring more recently merged transition systems and older transition systems, and also a randomized order (RND).[5]

5.4.3. Precomputed, Stateless, and Hybrid Merge Strategies

All the merge strategies we have discussed so far are linked to the causal graph of a planning task. This is obvious for the linear merge strategies, but it also holds for DFP and MIASM. With DFP, only transition systems that require non-trivial synchronization of transitions can be candidates for merging, where non-trivial synchronization means that both transition systems have a common label that is not irrelevant in both of them. For atomic factors, such common relevant labels only occur if the associated variables of the factors occur together in some operator of the planning task. More generally, for any two factors, there can only be non-trivial synchronization

[5] We did not describe this third option in our original work (Sievers et al., 2016), and all of the experiments there are based on using NTO. In the experiments of this thesis, we also evaluate the alternative of using OTN and RND.

if the associated variables of the FMs associated with the two factors (i.e. the "variables represented in the factors") are causally connected in the causal graph. The same reasoning applies to sbMIASM: only non-trivial synchronization can cause dead states in the product system, because if merging two causally unrelated factors, we obtain the full product, and hence no state is disconnected if there are not already disconnected states in the components of the product.

We now discuss in more generality how the two types of merge strategies we have seen so far fare in using information from the causal graph and other sources, such as the factored transition system during the merge-and-shrink computation. We also compare their potential for being combined or reused within other strategies.

An advantage of precomputed merge strategies is that they can capture the "big picture" of the planning task, such as causal dependencies. However, their disadvantage is their inflexibility: they are independent of the other merge-and-shrink transformations, which means that they cannot take into account the course of the merge-and-shrink computation, and they cannot easily be combined with other merge strategies, at least not without breaking their precomputed merge tree. For example, enhancing precomputed merge strategies with symmetries leaves the question of how to update the precomputed merge tree while merging according to symmetries. While this is still amenable for linear merge strategies since we can map variables of their underlying variable order to the factors that "represent" them, it is unclear how to best integrate MIASM with factored symmetries.

Score-based merge strategies are the opposite of precomputed merge strategies in some sense. Their disadvantage is that they are myopic in that they fail to "plan ahead" the current merge, and hence they cannot capture causal dependencies between more than two factors of the planning task. On the other hand, they can take into account the modifications done by other merge-and-shrink transformations and base their decision on the state of the factored transition system of the merge-and-shrink computation. Another advantage is that they are usually easy to understand and implement, since they are clearly defined in terms of the scores they compute. This contrasts precomputed merge strategies such as MIASM that tend to have rather complex precomputation algorithms. Additionally, score-based merge strategies can easily be combined with each other or reused within other merge strategies. For example, if after computing scores with a score-based merge strategy, ties need to be broken, we can use further score-based merge strategies to reduce the number of tied candidate pairs, and finally we can use the tie-breaking criteria we described in the previous section.

To get the best out of the two worlds that are precomputed and score-based merge strategies, we suggest a third type of merge strategies which we call *hybrid merge strategies*. All merge strategies that are not "simple" in the sense that they are either entirely precomputed or fully stateless can be considered hybrid merge strategies. Hybrid merge strategies potentially precompute some parts of the merge tree, leaving other merge decisions within subtrees to score-based merge strategies during the merge-and-shrink computation.

We have already seen a framework that turns simple, i.e. precomputed or score-based, merge strategies into hybrid ones: the symmetry-enhancing framework discussed in Section 5.3. Since symmetry-enhanced merge strategies need to memorize which transition systems are affected by a non-atomic factored symmetry across several merge decisions, they clearly are not stateless merge strategies. Because factored symmetries can only be computed for the current factored transition system, symmetry-enhanced merge strategies can also not be precomputed up-front.

5.4.4. A New Hybrid Merge Strategy

Driven by the previous discussion on types of merge strategies, we devise another hybrid merge strategy, which is again based on the causal graph. As a precomputation step, it computes the strongly connected components (SCCs) of the causal graph and decides on an order in which these SCCs should be considered. During the merge-and-shrink computation, the strategy then repeatedly merges all atomic factors for variables within an SCC, considering the SCCs in the specified order, which results in one product system for each SCC. From there on, the strategy merges these resulting products to form the final transition system.

This merge strategy, or rather this *framework* to enhance existing merge strategies which we call *SCC framework*, has several parameters: first, it requires to specify the already mentioned order in which SCCs should be considered for computing the product factors for each SCC (order of SCCs). Secondly, it needs a merge strategy that is able to decide on a merge order of the atomic factors for variables within SCCs (secondary merge strategy). Thirdly, it requires either another merge strategy that dictates the merge order of the product systems of SCCs, or alternatively, it again needs an order in which the SCCs are considered, which then can be used for linearly merging all product systems of SCCs (third parameter).

For the order of SCCs, we consider the following four variants: a topological and reverse topological sort of the SCCs, based on the derived directed graph where each SCC is a single "supervertex", and two orders in which SCCs are sorted by size, either decreasing or increasing, breaking ties by the topological order just described. For the merge strategy that decides on merging the atomic factors for variables within SCCs (secondary merge strategy), we can use any merge strategy that is able to select a next pair of transition systems out of a subset of the atomic factors of a planning task. This is of course easy for score-based merge strategies like DFP or sbMIASM, but also possible for linear merge strategies, since they define a variable order we can use directly. For other precomputed merge strategies like MIASM, it is in general not clear how to extract a subtree of the precomputed merge tree that contains only the necessary (atomic) factors, since they are not necessarily part of the same subtree.

For the third parameter, i.e. the order in which the product systems should be merged, we can again use the same different orders of SCCs to define a linear merge order on the product systems of SCCs, or alternatively, any merge strategy capable of deciding on a merge order given the product systems of SCCs. This is again trivially possible for all score-based merge strategies, but more complicated for precomputed merge strategies. We can use linear merge strategies using the technique described above that maps each variable of the underlying variable order to the factor "representing" that variable, but for the same reasons as above, we do not know how to use MIASM for that purpose.

In our original work introducing the merge strategy *SCC-DFP* (Sievers et al., 2016), we only reported results for the variant that uses the topological sort for the order of SCCS and DFP for the secondary merge strategy and the third parameter, hence not interpreting the strategy as a framework where any simple merge strategy could be plugged in. In the experimental study of this thesis, we also evaluate the alternative orders of SCCs, and we do not only use DFP as the secondary merge strategy and for the third parameter, but also consider sbMIASM and the linear merge strategies in the way described above.[6] For simplicity, we always use merge strategies

[6]Strictly speaking, the SCC merge strategies are not hybrid but can be entirely precomputed if using linear merge

Abbreviation	transformation strategy
CGGL	linear merge strategy: causal graph goal level (Helmert, Haslum, & Hoffmann, 2007)
F	shrink strategy: f-preserving (Helmert, Haslum, & Hoffmann, 2007)
B	shrink strategy: based on (approximating) bisimulation (Nissim, Hoffmann, & Helmert, 2011)
G	shrink strategy: based on greedy bisimulation (Nissim, Hoffmann, & Helmert, 2011)
GCGL	linear merge strategy: goal causal graph level (Nissim, Hoffmann, & Helmert, 2011)
L	linear merge strategy: (Fast Downward's variable order) level (Nissim, Hoffmann, & Helmert, 2011)
RL	linear merge strategy: (Fast Downward's variable order) reverse level (Nissim, Hoffmann, & Helmert, 2011)
DFP	non-linear merge strategy: due to Dräger, Finkbeiner, and Podelski (2006), adapted to planning by Sievers, Wehrle, and Helmert (2014)
MIASM	non-linear merge strategy: maximum intermediate abstraction size merge strategy (Fan, Müller, & Holte, 2014)
symm-X	non-linear merge strategy: merge strategy X enhanced with factored symmetries, called *symm* by Sievers, Wehrle, Helmert, Shleyfman, and Katz (2015)
sbMIASM	non-linear merge strategy: score-based MIASM, called DYN-MIASM by Sievers, Wehrle, and Helmert (2016)
SCC-X	non-linear merge strategy: individually merging SCCs of the CG, using X as secondary merge strategy (Sievers, Wehrle, & Helmert, 2016)

Table 5.1.: Abbreviations of transformation strategies with a brief summary and their source in the literature, in chronological order.

for the third parameter (and not any order of SCCs that induces a linear merge order), and we always use the same merge strategy as for the second parameter. We call the resulting merge strategy SCC-X if using merge strategy X as secondary merge strategy.

5.5. Conclusions

To summarize this chapter, we provide an overview of the transformation strategies we discussed in this chapter and which we will use in our experimental study. Table 5.1 lists the transformation strategies in chronological order, for each showing the abbreviation we use, a brief summary, and the source in the literature.

We discussed several new merge strategies and defined different types of merge strategies that have different strengths and weaknesses:

- Precomputed merge strategies fix a merge tree before the computation of the merge-and-shrink algorithm. Their advantage is being able to capture maximal causal dependencies of planning tasks, but they fail to take into account the impact of other transformation strategies which comes apparent only during the merge-and-shrink computation. Furthermore, they tend to be more complex to understand and to cause difficulties if combining them with other merge strategies. All linear merge strategies (CGGL, GCGL, L, RL), which can be understood as a variable order, and MIASM fall into this category.

- Stateless or score-based merge strategies are in some sense the opposite of precomputed merge strategies: they select the next pair of transition systems exclusively based on the current factored transition system and do not memorize any information across several merge decisions. They are myopic in the sense that the cannot plan ahead the current

strategies both as the secondary merge strategy and for the third parameter, because in that case all information is available from the beginning on.

merge decision, but they excel in taking into account the impact of other transformation strategies by greedily maximizing scores defined solely on the current factored transition system. Furthermore, they can usually be described mathematically and are simple to understand. They are also particularly flexible because they can easily be combined with each other. DFP and sbMIASM are representatives of this type of merge strategies.

- Hybrid merge strategies are those that are neither entirely precomputed nor solely score-based merge strategies. They can pick the best of both precomputed and score-based merge strategies, but they also have an disadvantage: they are usually similarly complex as precomputed merge strategies, and they also cannot easily be combined with other merge strategies. For example, it is unclear if there exists a meaningful way of combining a merge strategy with both factored symmetries and the SCC framework, which are the two ways of producing hybrid merge strategies we have seen.

In future work, besides investigating a possible combination of the frameworks for using factored symmetries and SCCs, we would also like to come up with a better integration of MIASM and factored symmetries. Similarly, we want to combine MIASM with the SCC framework. Given that score-based merge strategies are simple and flexible, yet powerful, we also think that there are many more criteria that could serve as a basis for even more informative score-based merge strategies, such as, e.g., looking at the number of transitions or states of factors, their g- and h-values, or the amount of combinable labels to maximize label reductions.

Furthermore, the approach that both the SCC framework and MIASM follow can be viewed as an even wider framework: in a precomputation phase, it partitions the variables of the planning task according to *some criteria*, in the spirit of how the SCC framework partitions variables based on SCCs and how MIASM partitions variables based on the expected ratio of dead states. Then it first merges all atomic factors for variables within a partition before merging the resulting products to form the final transition system, using any secondary merge strategy for these two subtasks.

6. Experimental Study

In this chapter, we present an experimental study of merge-and-shrink heuristics on classical planning benchmarks. Our main goal is to show the evolution of merge-and-shrink heuristics from before the contributions of this thesis to the state of the art. For this reason, we proceed chronologically and begin by reproducing and extending the results of our earlier papers. For this purpose, we use the previous, non-optimized implementation of merge-and-shrink based on generalized label reduction, and only afterwards evaluate the performance gains due to changing to the optimized implementation. From there on, we exclusively use the state-of-the-art optimized implementation and evaluate the most recently introduced merge-and-shrink techniques, following the chronological order of their presentation. Finally, we compare the state-of-the-art merge-and-shrink techniques against other abstraction heuristic based planners, including the state of the art planner.

This chapter is organized as follows. In Section 6.1, we describe the different implementations that we integrated into a common code base and the techniques available within them, in particular pointing out differences between the implementation used for this thesis and those used in previous work. We also explain the technical setup of the study and the abbreviations we use.

Section 6.2 begins our study with evaluating the impact of label reduction, and in particular that of generalized label reduction, on the performance of merge-and-shrink heuristics. It also includes an evaluation of the first two non-linear merge strategies DFP and MIASM. In Section 6.3, we report results for merge strategies enhanced by the framework for using factored symmetries. In Section 6.4, we evaluate the impact of the optimized implementation in, using all merge-and-shrink techniques available up to this point, i.e. available in the previous implementation of merge-and-shrink.

We then continue our study by evaluating further techniques on the optimized implementation. Section 6.5 investigates more variants of using factored symmetries and the alternative scenarios of using symmetries that are symmetry-based pruning and symmetric lookups. In a larger analysis of merge strategies in Section 6.6, we investigate the potential of current merge strategies by comparing against the set of all merge strategies on small planning tasks and large sets of randomly sampled merge strategies. We further evaluate tie-breaking strategies for DFP and sbMIASM, and discuss results of the SCC framework.

In Section 6.7, we evaluate the impact of pruning on the performance of merge-and-shrink heuristics. In Section 6.8, we proceed with providing an overview of state-of-the-art merge-and-shrink that includes new combinations of different merge strategies. Finally, in Section 6.9, we conclude the study with a comparison against other planners based on abstraction heuristics, including the state-of-the-art planner, and against the winner of the last International Planning Competition (IPC).

6.1. Setup

In this section, we first comment on the different code bases we use and the differences between older and newer implementations of some of the techniques. Furthermore, we describe the technical setup and explain how we illustrate the results of our study.

6.1.1. Implementation Differences and Integration into Fast Downward

We implemented all of the techniques in Fast Downward (Helmert, 2006). Recall that in the first part of our study (Sections 6.2–6.4), we want to reproduce results of techniques that are only available within older versions of Fast Downward, such as the previous label reduction or the non-optimized implementation of merge-and-shrink based on generalized label reduction. Since Fast Downward versions that are several years apart usually have huge differences in performance, we integrated the old implementations of merge-and-shrink into the most recent version of Fast Downward, attempting to keep the modifications required to these old implementations at a minimum. We think that this is the best way to allow for a fair comparison to state-of-the art techniques.

We use the same approach for integrating all techniques that are not part of public Fast Downward, such as the merge strategies MIASM and sbMIASM, the symmetry-enhancing framework, symmetry-based pruning techniques and symmetric-lookups. Hence *all* our own techniques we evaluate here are based on the same version of Fast Downward.[1] The source code of the resulting Fast Downward-based planner is publicly available.[2] To obtain the other planners evaluated in Section 6.9, please contact their authors. We also published all experimental data online.[3]

To compute structural symmetries and factored symmetries, we use the graph automorphism tool bliss (Junttila & Kaski, 2007), version 7.2, with the same implementation of problem description graphs as input in all implementations using factored symmetries. Our integration of bliss in Fast Downward handles failures of bliss computations due to reaching the memory limit or a time limit imposed to bliss. That means that even if the computation of bliss fails, Fast Downward continues with its regular computation.

We now comment on several specific implementation issues. In all implementations of the label reduction algorithm used for this work (including the non-optimized and the latest implementation), we fixed a bug that affects the results reported in all previous papers. Given an order of transition systems, if only the last one allowed for label reductions based on Θ-combinability, then this label reduction was not performed in the erroneous implementations, i.e. the fixed point iteration stopped one iteration early. If there have been previous label reductions within the same algorithm run, this problem did not persist.

When reporting results for DFP in the first part of the study, we use the implementation used since its introduction for planning (Sievers et al., 2014). In the second part, we also report different variants of DFP with different tie-breaking strategies, which we indicate accordingly. What we call DFP in all previous papers and the first part of our study matches the variant DFP with tie-breaking PC/RL/NTO in the second part of the study.

[1] Mercurial revision: 73041D26B55A
[2] https://doi.org/10.5281/zenodo.1163381
[3] https://doi.org/10.5281/zenodo.1164137

MIASM and the symmetry-enhancing framework are available both within the non-optimized implementation of merge-and-shrink based on generalized label reduction and the state-of-the-art implementation. However, their implementations exhibit some additional differences. Within the non-optimized implementation, we use the first implementation of MIASM, provided by its authors Fan et al., which they also used for their paper. Since their implementation stops computing a merge strategy if it only finds a trivial partitioning of the variables, i.e. if there is only one partition containing all variables or singleton partitions for every variable, we adapted it to fallback onto using DFP (in the explained old version, i.e. with tie-breaking PC/RL/NTO) whenever this happens. All state-of-the-art results for MIASM are based on a re-implementation, also provided by its authors, which we again adapted to use DFP as a fallback mechanism for a better comparison, instead of using the built-in fallback CGGL.[4] Since we are not familiar in detail with both implementations, we unfortunately cannot comment more on their differences.

In the symmetry-enhancing framework, an important difference between the non-optimized and optimized implementations of merge-and-shrink based on generalized label reduction is the way how variable orders are "broken" if enhancing linear merge strategies by merging according to symmetries. The old implementation considers the variable order underlying the linear merge strategy. Whenever two atomic transition systems Θ_1 and Θ_2 are merged where Θ_1 comes before Θ_2 in the variable order, it considers the product as a representative of Θ_1 and drops Θ_2. In the new implementation, we represent precomputed merge strategies such as linear ones as (merge) trees, which means that whenever two factors that are in different subtrees are merged, we cut one of the leaves and update the other one to represent the product. Since it is not possible for a given leaf to know whether it is ordered before or after another leaf without traversing the tree, we make that choice by randomization. In combination with the fact that we always traverse the tree in a fixed order to uniquely represent precomputed merge strategies, this means that symmetry-enhanced linear merge strategies can result in different merge orders compared to the previous implementation, which unfortunately makes some of the reported results with linear merge strategies less comparable.

When reporting results for sbMIASM, we use an improved implementation compared to that used in the original paper (Sievers et al., 2016). In particular, to compute the ratio of alive states to total states of all products in the old implementation, we cloned the component transition systems and FMs, possibly shrunk them and then pruned the product to obtain the result. Due to decoupling in the improved implementation, we now only clone the transition systems. Additionally and in contrast to the previous implementation, we only clone a transition system if we need to modify (i.e. shrink) it before merging. Furthermore, we do not need to actually prune the product as in the old implementation where pruning was not a logically separated step, but can deduce the ratio of alive to total states from computing distance information. Hence we save several copy operations (runtime) but also memory to store these temporary copies.

6.1.2. Technical Setup

We use the benchmarks of the (optimal, where applicable) sequential tracks of all IPCs up to 2014, a set comprised of 1667 planning tasks distributed across 57 domains, which are publicly

[4] This contrasts the results we originally reported (Sievers et al., 2016), where we also used the re-implementation but did not adapt it to use DFP as a fallback mechanism yet.

Attribute	explanation	winning value
Coverage	number of solved tasks	max
Exp Xth perc	Xth percentile of the number of expansions, rounded down to the next multiple of 1000, to reach the last f-layer, computed over all tasks where all algorithms have a value for expansions	min
Search time	geometric mean of the runtime of search in seconds, computed over commonly solved tasks	min
Total time	geometric mean of the total runtime in seconds, computed over commonly solved tasks	min
# constr	number of tasks for which the merge-and-shrink computation finishes	max
Constr time	arithmetic mean of the runtime in seconds required by the merge-and-shrink computation, computed over commonly solved tasks	min
Constr oom	number of tasks for which the merge-and-shrink computation exhausts the memory limit	min
Constr oot	number of tasks for which the merge-and-shrink computation exhausts the time limit	min
Perfect h	number of tasks for which the resulting heuristic is perfect	max
Linear tree	percentage of # constr for which the merge tree is linear	min
MITSS	arithmetic mean of the maximum intermediate size of transition systems, computed over commonly solved tasks	min

Table 6.1.: Attributes of planners and their abbreviation.

available.[5] This set contains duplicates because some of the domains and tasks of IPC 2008 have been reused in IPC 2011. We still stick to this set of benchmarks because it is an established test bed for comparing planners. All planning tasks are given as PDDL files which we translate into SAS$^+$ representations using the translator that is part of Fast Downward (Helmert, 2009).

We use downward-lab for conducting our experiments (Seipp, Pommerening, Sievers, & Helmert, 2017). Time is limited to 30 minutes and memory to 2 GiB per task. All experiments were run on machines with Intel Xeon E5-2660 CPUs running at 2.2 GHz.[6] To keep the impact of the number of concurrently running tasks and other factors of the compute cluster at a minimum, downward-lab randomizes the order in which tasks are started.

We use the shrink strategies available in Fast Downward, i.e. B, F, and G. When using the shrink strategies based on bisimulation, i.e. B and G, we allow the shrink strategy to attempt to compute a perfect bisimulation even if no shrinking is necessary (by setting the so-called size "threshold" parameter that triggers shrinking even if not required to 1), and we apply label reductions before shrinking, as bisimulation profits from label reduced transition systems. For the shrink strategy F, experiments have shown that there is no benefit of computing abstractions if no shrinking is necessary, hence we leave the threshold parameter inactive (i.e. set it to the same value as N). Furthermore, we perform label reduction after shrinking, since this is less expensive and f-based shrinking does not profit from label reduced transition systems. For all experiments with shrink strategies B and F, we use a size limit of $N = 50000$ for all transition systems at any point during the computation of merge-and-shrink. While this is an arbitrary choice and somewhat small, the same value was used in most recent papers and hence allows for a better comparison; other reasonable values which we do not test here use up to $N = 200000$ states. If using G, we follow the common practice of not limiting the size of transition systems by setting $N = \infty$.

[5] From the collection at https://bitbucket.org/aibasel/downward-benchmarks, mercurial revision 663D121BEC5B, we use the "optimal strips" benchmark suite.

[6] The compute cluster used a different OS (CentOS 7.3 instead of CentOS 6.5) compared to results reported in all of our previous work, which led to results that may differ slightly from results reported in these works, presumably due to different system libraries.

Table 6.1 lists the attributes that we use to describe aggregated results. The first column indicates the abbreviation as we use them in our tables. The second column explains the attribute and possibly mentions a function we use to aggregate values. If reporting results aggregated for a single domain, we either use the given aggregation function such as the geometric mean or a specific percentile, or the sum of the number of tasks as a default. To aggregate results across all domains, we use a two-level aggregation: we first compute the aggregated result for each domain individually and then use the same aggregation scheme to compute the overall result. This weights domains with an equal share, thus negating effects that could otherwise occur due to unequally sized domains. Finally, the third column indicates whether for the attribute, higher values are considered better or not. In all tables or blocks within tables below, we highlight the best performance for each attribute. If the tables contain several blocks, e.g. grouping results of different variants of a single merge-and-shrink strategy, then we usually aggregate and highlight values separately for each block. When reporting domain-wise results, we provide the number of tasks of a domain in parenthesis after its name.

6.2. The Impact of (Generalized) Label Reduction and DFP

In this section, we investigate the impact of label reduction, which includes a comparison of using no label reduction at all,[7] the previous label reduction, and the discussed variants of exact generalized label reduction in the non-optimized implementation. We also evaluate the first non-linear merge strategy DFP which we introduced together with generalized label reduction, as well as the non-linear merge strategy MIASM introduced shortly afterwards (Fan et al., 2014). As the previous label reduction was not implemented for non-linear merge strategies, we cannot report any results of the previous label reduction for DFP and MIASM.

We first discuss the numerous variants exclusively in terms of coverage over all domains, and only later go into more detail for a few selected representatives of the variants. Table 6.2 shows coverage for the different variants of using (or not using) label reduction, combined with all shrink strategies and all linear merge strategies, DFP, and MIASM. It shows a horizontal blocks for each of the shrink strategies, divided into columns according to the different label reduction variants, and uses a row for each merge strategy.

We begin with considering the results for the shrink strategy B, shown in the first block in columns 2–6. The first and most important observation is that label reduction is crucial for the efficient computation of merge-and-shrink heuristics: Comparing the results of using no label reduction (No LR) against using the previous label reduction (Old LR) or any of the variants based on generalized label reduction (2TS, ONCE, FP), the smallest difference in coverage is 38 for RL (no LR vs. old LR), a huge increase in optimal classical planning. The second observation concerns the comparison of the different label reduction techniques: for the three linear merge strategies CGGL, GCGL, and L, the previous label reduction and the cheap variant 2TS of generalized label reduction achieve the best coverage, however closely followed by the

[7] We use the non-optimized implementation of merge-and-shrink based on generalized label reduction to produce results for not using label reduction (no LR). Compared to using no label reduction in the implementation of merge-and-shrink based on the previous label reduction, this increases coverage of RL by 10 (i.e. the result of RL would be 654 with the previous implementation), but does not affect coverage of the other linear merge strategies.

	B					F					G				
	No	Old	Generalized LR			No	Old	Generalized LR			No	Old	Generalized LR		
	LR	LR	2TS	ONCE	FP	LR	LR	2TS	ONCE	FP	LR	LR	2TS	ONCE	FP
CGGL	628	**686**	685	680	682	480	**505**	503	502	502	542	**597**	**597**	568	565
GCGL	614	673	**674**	670	669	491	**513**	508	512	512	541	560	**563**	537	533
L	618	**676**	**676**	674	674	499	**519**	517	517	517	541	560	**563**	536	535
RL	664	702	702	**720**	**720**	469	481	483	**500**	**500**	498	**589**	586	555	552
DFP	654	-	719	**737**	736	495	-	514	**523**	522	499	-	**586**	555	552
MIASM	655	-	731	745	**746**	534	-	540	559	**562**	482	-	**557**	548	547

Table 6.2.: Coverage of different variants of (not) using label reduction: no label reduction (No LR), previous label reduction (Old LR), and the variants 2TS, ONCE, and FP of generalized label reduction (generalized LR), the latter two using a randomized order of transition systems. Columns are grouped by the shrink strategy used, and each row shows the results for a different merge strategy. Best coverage highlighted in bold within each group of shrink strategy and merge strategy.

more expensive variants that compute combinable labels for all factors. For RL and the non-linear merge strategies DFP and MIASM, ONCE and FP perform significantly better than 2TS (and than Old LR with merge strategy RL), i.e. for these merge strategies, reducing labels as much as possible pays off.

A closer look at the linear merge strategies CGGL, GCGL, and L shows that their variable orders are often "similar" in the sense that they often only differ in a few variables being ordered differently in the front part of the variable order, because also CGGL and GCGL use L as a tiebreaker. This has the effect that for large parts of the merge order (and for the remainder of the merge order after merging all differently ordered variables), exactly the same labels can be reduced for all three merge strategies. On the other hand, RL is the opposite variable order of L. Thus, with CGGL/GCGL/L, the variant 2TS that considers only the two transition systems merged next necessarily computes different label reductions as with RL. A possible reason why 2TS works better with CGGL/GCGL/L than with RL (and DFP/MIASM) is that 2TS misses combinable labels in early iterations of the merge-and-shrink computation that can be combined if using 2TS with RL, which has the effect that these labels can later on be combined in a different way. This means that in the case of 2TS, the order in which label reductions are performed matters, however not within an iteration of the fixed point algorithm but across several stages of label reductions in the merge-and-shrink algorithm.

While there is no obvious reason that the "full label reduction" methods ONCE and FP should perform better with RL or the non-linear merge strategies DFP and MIASM, we hold on to the observation that the impact of label reduction clearly depends on the chosen merge strategy. We also note that these three merge strategies achieve much higher coverage than the three linear ones CGGL/GCGL/L, out of which CGGL is the best performer, and in particular, the introduction of the first non-linear merge strategy clearly improves the performance of merge-and-shrink heuristics, which is only surpassed by the non-linear merge strategy MIASM (which uses DFP as a fallback mechanism).

Investigating the impact of label reduction on the non-linear merge strategies DFP and MI-

ASM, we observe that they benefit even more from label reduction than the linear merge strategies. One explanation for this observation is that non-linear merge strategies involve more complex products (merges) than linear ones, and hence benefit more from label reduction collapsing multiple parallel transitions into one. In linear merge strategies, at least one of the merged transition systems is always atomic, and atomic transition systems tend to have a comparatively low density of transitions. Another reason is that label reduction interacts favorably with the merge strategy DFP, which – unlike merge strategies previously considered in planning – takes the labels into account directly in order to decide which transition systems to merge next.

We now turn towards the results of the shrink strategy F, shown in the second block of Table 6.2 in columns 7–11. Similarly to the results for B, using any kind of label reduction increases coverage compared to not using label reduction, but the impact is smaller. The likely reason is that f-preserving shrinking, unlike bisimulation-based shrinking, does not profit from label reduction because it only considers states and their g- and h-values but not their transitions. We again also observe that with RL, DFP, and MIASM, using the more expensive label reduction variants ONCE and FP pay off, while the same is not true for CGGL, GCGL, and L, which perform best with the previous label reduction. Unlike with B, using F is stronger combined with GCGL and L than combined with RL, but the non-linear merge strategies DFP and MIASM are still the strongest performers.

Table 6.2 also shows the results of the shrink strategy G in the last block in columns 12–16. Here, we observe different trends than with G and F: for all merge strategies, the cheapest variant of generalized label reduction 2TS achieves the best coverage, followed by the previous label reduction where applicable (for RL, the order of Old LR and 2TS is swapped). While using any variant of label reduction is still strongly beneficial most of the time, with GCGL and L, it is even preferable to not reduce labels at all compared to using ONCE or FP. We think that these expensive-to-compute variants of generalized label reduction, compared to the relatively cheap-to-compute shrink strategy G, do not pay off in particular because greedy bisimulation often drastically shrinks transition systems so that they are of small size anyway. Considering the linear merge strategies, similarly to using B, we observe that RL profits most from label reduction, achieving the worst coverage without label reduction and the best with the previous label reduction. Independently of the used variant of label reduction, RL and CGGL perform better than L and GCGL.

Finally, comparing the three shrink strategies, it is clear that B is much stronger than F and G, regardless of the used merge strategy or label reduction technique. G achieves better coverage results than F (except with MIASM), mostly because the heuristic is very fast to compute and hence can be computed for more tasks. For the remainder of this chapter, we will mostly focus on using B as shrink strategy, following the practice used in most of recent work on merge-and-shrink. We sometimes briefly discuss the results if using F or G instead, but usually include the data only in the appendix to keep the presentation readable. Since we focus on B and to avoid reporting results for all linear merge strategies, we only report results for RL, the best linear merge strategy if using B, and CGGL, the best of the three similar remaining strategies. While RL and CGGL are also the best strategies if using G, leaving out L and GCGL may sometimes result in reporting slightly worse performance for F than what could be achieved using L instead.

For the best shrink strategy B, and also for the different type of shrink strategy that is F, using the full fixed point computation FP usually comes close to the best performance or delivers the

	CGGL			RL			DFP		MIASM	
	No	Old	FP	No	Old	FP	No	FP	No	FP
Coverage	628	**686**	682	664	702	**720**	654	**736**	655	**746**
Exp 50th perc	8063	**2915**	**2915**	11k	**1191**	1494	6058	**1074**	7705	**701**
Exp 75th perc	1302k	**1085k**	**1085k**	1162k	934k	**886k**	1058k	**591k**	558k	**457k**
Search time	0.31	0.24	**0.24**	0.40	**0.23**	0.24	0.33	**0.21**	0.28	**0.19**
Total time	2.95	**2.71**	3.23	3.22	**2.62**	2.74	3.46	**3.06**	8.48	**7.86**
# constr	1030	1307	**1338**	1142	1286	**1413**	1081	**1389**	1029	**1367**
Constr time	44.33	**37.73**	43.77	39.62	**33.52**	36.78	57.52	**45.27**	33.83	**30.59**
Constr oom	635	350	**184**	501	356	**112**	558	**134**	550	**149**
Constr oot	**2**	10	145	**24**	25	142	**28**	144	**88**	151
Perfect h	205	**257**	**257**	191	**265**	**265**	197	**266**	228	**307**

Table 6.3.: Detailed results of different variants of label reductions (No, Old, and generalized fixed point label reduction) for merge strategies CGGL, RL, DFP, and MIASM, all using shrink strategy B. Values aggregated and highlighted for each of the four merge strategies individually.

best performance among all variants. For the remainder of this study, we stick to using the full fixed point computation rather than ONCE or 2TS, hence using the same configuration as for all results reported in other papers using generalized label reduction to the best of our knowledge. Reducing labels as much as possible is not only important for allowing a better comparison to previously reported results, but also with respect to the result of Nissim et al. (2011) which states that for certain planning domains, using bisimulation-based shrinking with *fully* label reduced transition systems, we can obtain perfect heuristics in polynomial time.

Having settled on a subset of merge-and-shrink configurations to consider, we now report a few more detailed results to show the impact of label reduction and our non-linear merge strategy DFP. Table 6.3 shows more attributes than above Table 6.2, restricted to the four mentioned merge strategies, the shrink strategy B, and the variant of generalized label reduction that uses the fixed point computation.

Apart from the already discussed increase in coverage, we observe that expansions generally decrease for all configurations, and except for RL with the 50th percentile, generalized label reduction (FP) requires the fewest number of expansions. This decrease also transfers to search time, however total time does not always decrease significantly if using generalized label reduction but may even increase due to its expensive computation. Another indicator for the increases heuristic quality is the increased number of tasks for which the perfect heuristic can be computed. Still, the increase in heuristic quality due to label reduction in conjunction with bisimulation-based shrinking is not the only benefit. We also observe that label reduction (both the old and the generalized variant) is a crucial ingredient for the efficient computation of merge-and-shrink abstractions: the number of tasks for which the computation finishes (# constr) increases significantly already with the previous label reduction, and even more with generalized label reduction. This increase is achieved mainly due to a more compact representation of merge-and-shrink abstractions, as the computation fails less times due to reaching the memory limit (Constr oom), at the price of an increased number of cases where the computation reaches the time limit. However, on average, the construction time of merge-and-shrink abstractions (Constr time) even decreases with generalized label reduction, in spite of the expensive

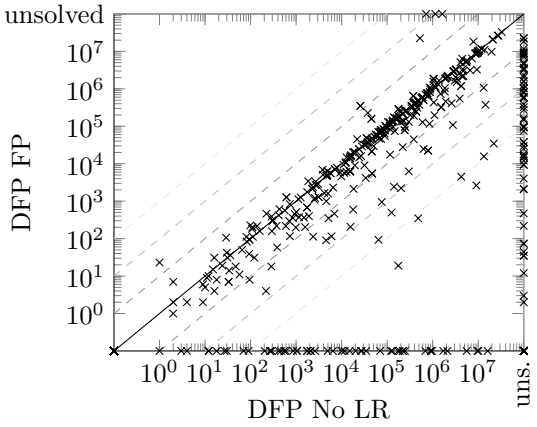


Figure 6.1.: Expansions of DFP with no label reduction (No LR) and with fixed point generalized label reduction (FP).

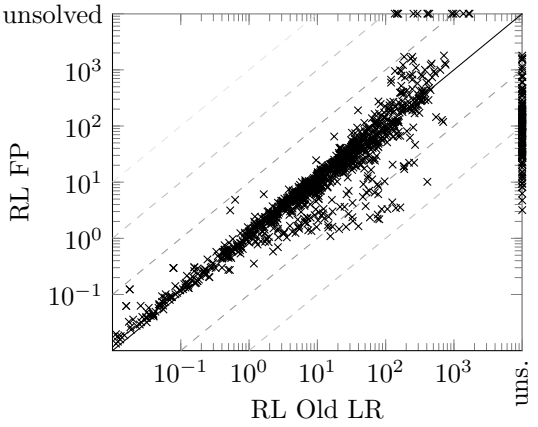


Figure 6.2.: Construction time of RL with old label reduction (Old LR) and with fixed point generalized label reduction (FP).

computation of label reduction.

We now show some domain-wise results for the best linear merge strategy RL and our non-linear merge strategy DFP. Table 6.4 shows domain-wise coverage for RL and DFP with the available variants of label reductions, aggregating domains with equal coverage of all variants in the second-to-last row. It confirms that label reduction is very useful across the board, over a wide range of domains. For the linear merge strategy RL, generalized label reduction increases coverage in 10 domains compared to the baseline where no labels are reduced, while decreasing coverage in only 2 domains by only 1 task. For the non-linear DFP merge strategy, label reduction increases coverage in 19 domains and decreases it in 1 domain by 1 task.

To provide another detailed view, Figure 6.1 shows the number of expansions of DFP, with and without label reduction. The figure plots the results without label reduction against the results with our new label reduction approach, over all instances in the benchmark suite. The figure clearly shows the significant impact that label reduction has on performance in many cases.

To compare the previous label reduction against generalized label reduction (only possible for the linear merge strategies), Figure 6.2 compares the time to construct the merge-and-shrink abstraction for the merge strategy RL. With generalized label reduction, even though it is much more expensive to compute than the previous one, the merge-and-shrink computation terminates faster and runs out of memory far less frequently.

Figure 6.3 compares expansions for the same configurations, showing that the heuristics are similarly informative in both cases. It is mainly the ability to complete the computation of the abstraction (see Figure 6.2) that makes the difference between the old and new label reduction here.

Finally, we compare DFP against the best linear merge strategy RL. Figure 6.4 compares the number of expanded states for both strategies using generalized label reduction. The comparison shows that while DFP achieves a higher coverage (cf. Table 6.3), the two merge strategies are

	RL			DFP	
	No LR	Old LR	FP	No LR	FP
airport (50)	**19**	**19**	18	18	18
blocks (35)	25	25	25	25	**26**
depot (22)	**7**	6	6	6	6
elevators-opt08-strips (30)	11	11	11	**16**	**16**
elevators-opt11-strips (20)	9	9	9	**13**	**13**
floortile-opt11-strips (20)	4	**5**	**5**	4	**5**
freecell (80)	14	11	19	9	**20**
grid (5)	**2**	**2**	**2**	1	**2**
gripper (20)	7	**20**	**20**	7	**20**
hiking-opt14-strips (20)	11	11	11	**13**	**13**
miconic (150)	58	**72**	**72**	58	**72**
mprime (35)	14	16	**23**	6	**23**
mystery (30)	11	10	**16**	8	**16**
nomystery-opt11-strips (20)	16	**18**	**18**	16	**18**
parcprinter-08-strips (30)	**16**	**16**	**16**	14	14
parcprinter-opt11-strips (20)	**12**	**12**	**12**	10	10
pipesworld-notankage (50)	**16**	**16**	**16**	14	**16**
pipesworld-tankage (50)	13	**15**	**15**	11	14
psr-small (50)	49	**50**	49	**50**	**50**
rovers (40)	7	**8**	**8**	7	**8**
satellite (35)	5	**6**	**6**	5	**6**
scanalyzer-08-strips (30)	12	12	12	12	**13**
scanalyzer-opt11-strips (20)	9	9	9	9	**10**
sokoban-opt08-strips (30)	24	24	24	**26**	25
sokoban-opt11-strips (20)	19	19	19	**20**	**20**
tetris-opt14-strips (17)	0	**2**	**2**	0	1
transport-opt14-strips (20)	6	6	6	**7**	**7**
trucks-strips (30)	6	**7**	6	6	6
woodworking-opt08-strips (30)	11	11	11	11	**13**
woodworking-opt11-strips (20)	6	6	6	6	**7**
zenotravel (20)	9	**12**	**12**	10	**12**
Sum (1050)	428	466	484	418	**500**
Remaining domains (617)	236	236	236	236	236
Sum (1667)	664	702	720	654	**736**

Table 6.4.: Domain-wise coverage of merge strategies RL and DFP with using no label reduction (No LR), previous label reduction (Old LR, only for RL), and generalized label reduction based on the fixed point computation (FP), using shrink strategy B.

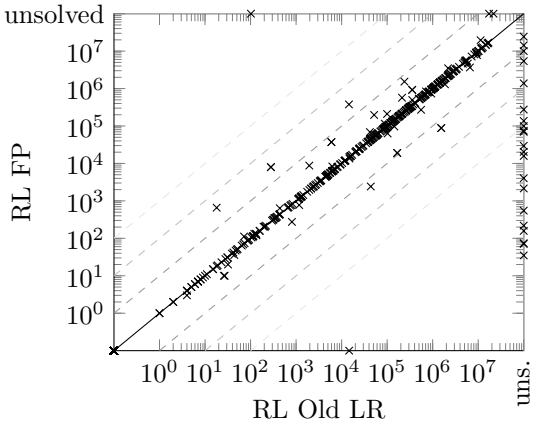


Figure 6.3.: Expansions of RL with old label reduction (Old LR) and with fixed point generalized label reduction (FP).

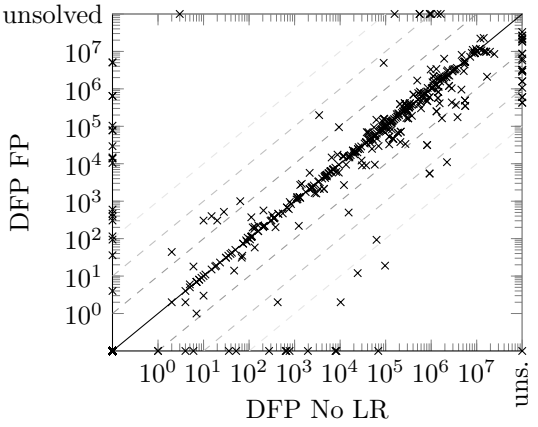


Figure 6.4.: Expansions of DFP and RL with fixed point generalized label reduction (FP).

quite complementary, with both strategies greatly outperforming each other on a significant number of instances, thus confirming the domain-wise coverage results shown in Table 6.4.

Concluding this part, we have shown that label reduction and in particular generalized label reduction has a strictly positive impact on the performance of merge-and-shrink heuristics. They do not only increase the efficiency of the merge-and-shrink computation a lot, but also allow shrink strategies based on bisimulation to reduce transition systems in a less lossy way, thus increasing the quality of the resulting heuristic. Furthermore, we also evaluated the first non-linear merge strategy DFP that we adapted to planning, as well as the follow-up non-linear merge strategy MIASM and showed that they both outperform existing linear merge strategies.

6.3. Factored Symmetries

In this section, we evaluate the symmetry-enhancing framework for merge strategies which we described in Section 5.3. Unless stated otherwise, all results are obtained using shrink strategy B and exact generalized label reduction using the fixed point algorithm. We mainly reproduce the results of our original work (Sievers, Wehrle, Helmert, Shleyfman, & Katz, 2015), but additionally also report results for the variant choosing a symmetry affecting the fewest transition systems (*smallest*). The additional approach that does not use the manually determined non-linear merge order for merging the affected transition systems but instead uses the fallback merge strategy, cannot easily be implemented in the non-optimized implementation we use in this section. We hence evaluate this alternative approach for merging according to symmetries in Section 6.5.

Furthermore, we also evaluate the two ways of limiting the resources of the symmetry computation described in Section 5.3. The variant which computes symmetries only once, i.e. computes symmetries of the induced factored transition system, is called *symm1*. The second variant, where we assign a total budget of 60s to all computations of symmetries of the merge-and-shrink

algorithm, is called *symm*. While the choice of 60s is arbitrary, we will see below that 60s is enough to compute symmetries many times within single merge-and-shrink computations.

In addition to the data we usually report (cf. Table 6.1), we also consider the following attributes of planner runs:

- Fallback: the percentage of # constr for which no symmetries are found for merging, i.e. the strategy behaves as the original variant.
- Bliss comp: the number of bliss attempts at computing symmetries.
- B time avg (am/gm): The arithmetic mean of the run times of bliss of a single merge-and-shrink computation, aggregated with the usual two-level process as either the arithmetic mean (am) or the geometric mean (gm).
- Bliss oom/Bliss oot: the number of computations of bliss which fail due to reaching the memory or time limit.
- Symm attempts: the summed number of times a non-atomic symmetry is selected for merging.
- Symm fail: out of the number of attempts of merging for symmetries, the number of times the merge-and-shrink algorithm prunes unreachable states or, shrinks any of the affected transition systems, thus potentially breaking the symmetry that the merge strategy currently aims for.

Table 6.5 shows results of the available merge strategies, enhanced with symmetries in the variants discussed above. We first compare coverage of all variants. We observe that independently of choosing *smallest* or *largest* and *symm1* or *symm*, merging according to symmetries improves coverage (except for DFP-*symm1*-*smallest*, where it is the same). This is true even if computing factored symmetries only on the induced factored transition system (*symm1*). The increase is very large for the linear merge strategies, e.g. CGGL improves from 682 to 729 for the best variant *symm-largest*, and RL from 720 to 740. For DFP and MIASM, the gains are less pronounced ($+8$ for DFP, $+7$ for MIASM[8]), but still non-negligible.

Comparing the variants *symm1* and *symm*, we see that coverage further increases, independently of choosing *smallest* or *largest* (except for MIASM), if computing symmetries in potentially many merge-and-shrink iterations (*symm* instead of only the first one (*symm1*). This increase is much stronger for *largest* (except RL) than for *smallest*. Comparing *symm1* and *symm* more generally, we expected that choosing a smaller symmetry leads to higher chance of merging all affected transition systems without intermediate shrinking or pruning of unreachable states, so that the symmetry is not broken and can be exploited for exact shrinking afterwards. However, choosing *smallest* tends to perform better than *largest* only with *symm1* (except CGGL), but for the generally stronger variant *symm*, *largest* tends to perform better than *smallest*.

[8] As explained in Section 5.4.3, combining precomputed merge strategies like MIASM with factored symmetries breaks the precomputed merge order of such merge strategies, thus also potentially breaking the original aim of MIASM, which is to minimize the size of intermediate transition systems.

	orig	*symm1*		*symm*		*orig*	*symm1*		*symm*	
		largest	*smallest*	*largest*	*smallest*		*largest*	*smallest*	*largest*	*smallest*
	CGGL					RL				
Coverage	682	724	712	**729**	723	720	732	734	**740**	737
Exp 50th perc	8920	**8359**	**8359**	**8359**	**8359**	**2132**	11k	5302	12k	10k
Exp 75th perc	1167k	951k	1035k	827k	**562k**	934k	739k	695k	570k	**564k**
Search time	0.30	0.24	0.26	0.24	**0.23**	0.22	0.24	**0.21**	0.24	0.23
Total time	3.69	**3.01**	3.13	5.39	5.11	3.40	3.30	**3.14**	6.25	5.86
# constr	1338	1364	1344	**1367**	1355	1413	1417	1407	**1426**	1416
Constr time	92.60	**75.90**	87.68	87.84	87.70	**97.64**	99.48	108.25	114.20	112.11
Constr oom	184	**147**	168	158	170	112	**93**	104	94	106
Constr oot	145	156	155	**142**	**142**	**142**	157	156	147	145
Perfect h	257	**284**	272	279	277	265	**274**	267	269	267
Linear tree	99.85	35.12	38.24	**29.26**	29.89	99.86	30.35	30.42	**28.75**	29.38
Fallback	-	**26.03**	26.26	27.36	27.68	-	**27.24**	27.58	28.12	28.46
Bliss comp	-	1361	1342	18584	**19266**	-	1414	1405	20324	**20965**
B time avg (am)	-	13.63	12.02	6.67	6.38	-	14.17	13.65	7.16	7.11
B time avg (gm)	-	0.42	0.37	0.96	0.91	-	0.46	0.44	1.00	0.93
Bliss oom	-	273	273	382	385	-	273	273	332	337
Bliss oot	-	**0**	**0**	438	454	-	**0**	**0**	458	468
Symm attempts	-	1008	991	2959	**3424**	-	1030	1019	3164	**3668**
Symm fail	-	**0**	**0**	1448	1426	-	**0**	**0**	1582	1577
	DFP					MIASM				
Coverage	736	736	743	**745**	742	746	747	**753**	751	752
Exp 50th perc	**2295**	12k	6934	12k	12k	**881**	3264	1883	12k	12k
Exp 75th perc	548k	943k	651k	513k	**503k**	528k	827k	**507k**	706k	706k
Search time	0.21	0.25	**0.21**	0.25	0.25	**0.19**	0.25	0.22	0.26	0.24
Total time	3.35	3.23	**3.17**	6.10	5.81	**8.21**	8.28	8.31	12.15	11.63
# constr	1389	1399	1389	**1405**	1396	1367	1376	1362	**1381**	1377
Constr time	93.13	**92.90**	97.02	105.04	105.32	83.52	**74.39**	90.80	86.74	84.11
Constr oom	134	**114**	124	**114**	127	149	**130**	143	132	137
Constr oot	**144**	154	154	148	**144**	**151**	161	162	154	153
Perfect h	266	**270**	267	265	265	**307**	285	289	284	284
Linear tree	88.98	33.60	47.01	**28.19**	28.87	58.30	16.35	22.39	14.19	**14.02**
Fallback	-	**26.66**	26.85	28.26	28.51	-	**26.53**	26.80	28.89	29.05
Bliss comp	-	1394	1384	17992	**18434**	-	1372	1356	19437	**20363**
B time avg (am)	-	14.15	14.06	7.13	7.08	-	13.28	12.28	7.58	7.32
B time avg (gm)	-	0.44	0.43	1.08	1.03	-	0.40	0.39	0.94	0.82
Bliss oom	-	273	273	341	343	-	156	155	221	221
Bliss oot	-	**0**	**0**	493	510	-	**0**	**0**	430	441
Symm attempts	-	1023	1013	2961	**3445**	-	1009	993	2900	**3412**
Symm fail	-	**0**	**0**	1451	1441	-	**0**	**0**	1445	1452

Table 6.5.: Results of unmodified merge strategies (*orig*) and combinations of the following symmetry-enhanced variants: only computing factored symmetries once (*symm1*) or as long as the time budget of 60s allows (*symm*), choosing a symmetry affecting the most (*largest*) or fewest (*smallest*) factors. All configurations with shrink strategy B. Values aggregated and highlighted for each of the four merge strategies individually.

A possible reason for this behavior can be seen by looking at the number of times a merge strategy attempts to merge according to a chosen symmetry (Symm attempts) and the number of times this symmetry will likely not be present after completing all merges due to intermediate shrinking or pruning of unreachable states (Symm fail). While for factored symmetries of the induced factored transition system (*symm1*), we can *always* merge all affected factors without breaking the symmetry, for *symm*, the same is not true: roughly one half of the attempts fails if using *largest*, and roughly 40% if using *smallest*. Hence choosing a large symmetry might pay off because it allows merging according to symmetries for more merge-and-shrink iterations (under the assumption that this is beneficial even with intermediate shrinking and pruning of unreachable states, which the data suggests).

We now investigate the reasons why the symmetry-enhanced merge strategies perform better than the original variants, but will also do so in more detail below for one of the merge strategies and one of its symmetry-enhanced variants. Looking at the aggregated expansions, we see that expansions decrease slightly comparing the original with all symmetry-enhanced variants, however not significantly (most of the times not an order of magnitude), which also transfers to a slight decrease in search time on average. However, this decrease in expansions and search cannot solely explain the sometimes large gains in coverage. The second reason is that symmetry-enhanced merge strategies improve the efficiency of the merge-and-shrink computation, as can be seen by the fact that most of the symmetry-enhanced variants can finish the merge-and-shrink computation for more tasks than the baseline, and in particular, *symm-largest* achieves the highest values for all variants. While the average construction time usually increases, especially with *symm*, the number of times the computation fails due to hitting the time limit (Constr oot) does not increase a lot. Furthermore, the number of times the computation fails due to hitting the memory limit (Constr oom) always decreases, thus explaining the increase in the number of tasks for which the heuristic can be constructed.

A further reason for the more efficient computation lies in the observation that enhancing merge strategies results in computing non-linear merge trees much more frequently compared to the baseline (Linear tree). Roughly 30% of the symmetry-enhanced linear merge strategies and DFP (which in its original form produces linear merge orders for nearly 90% of the tasks) are linear, and with MIASM, the symmetry-enhanced strategy produces as few as 14% linear merge trees compared to the original strategy which does so for more than half of the tasks. As discussed in Chapter 4, non-linear merge strategies are more powerful in theory, and our results here suggest that they can be computed more compactly than linear ones (less tasks for which the memory limit is reached). This efficiency improvement in particular explains the huge gains for the linear merge strategies. Finally, we note that symmetries arise in many domains and tasks: only in roughly 26%–27% of all tasks, the merge strategy is the same as the original one because no symmetries can be used for merging (Fallback).

We also investigate the efficiency of the computation of symmetries. We see that even if computing factored symmetries of the induced factored transition system (*symm1* variants), the graph automorphism tool bliss (without a time limit) exhausts memory in 273 (156 for MIASM; Bliss oom) out of 1356–1413 times in which bliss is tasked to compute such symmetries (Bliss comp). With the variant *symm*, we observe that Bliss is tasked to compute symmetries between 17976 and 20966 times in total across all merge-and-shrink iterations on all tasks, but fails between 332 and 386 (386 for MIASM) times due to reaching the memory limit and between

436 and 510 times due to exhausting the time budget. For example, for CGGL-*largest-symm*, this means that Bliss exhausts resources in 27.6% of the times. We also observe that for the *symm1* variants, there is no overhead incurred by computing symmetries (the total time tends to decrease). However with the *symm* variants, the total time increases significantly on average, which clearly is due to the computation of symmetries, because search time tends to slightly decrease and construction time, which includes the time to compute symmetries, increases.

These observations mean that the computation of factored symmetries does not come for free and can incur quite a substantial overhead. Still, we also see that on average, a single run of Bliss to compute factored symmetries of the induced factored transition system (*symm1*) takes 12.02–14.17 seconds in the arithmetic mean or 0.37–0.46 in the geometric mean. For the variant *symm*, the arithmetic mean of the average computation time decreases, but the geometric mean increases. While the latter is what we would expect, the reason for the decrease of the arithmetic mean is that for some domains, the induced factored transition system contains very large factors. For such induced factored transition systems, the merge-and-shrink computation usually considerably reduces the sizes of the factors within the first iteration, using label reduction and (exact) shrink transformations. This in turn reduces the computation time of bliss in the following iterations significantly. We further observed that towards the end of the merge-and-shrink algorithm, the average time to compute symmetries with bliss decreases again. It typically peaks towards the "middle" of the computation, where the factored transition system still contains many factors and many of these are of larger sizes in terms of both the numbers of transitions and states. These observations suggest that it may be worth investigating further variants of limiting bliss computations, e.g. by limiting each single call together with imposing a general budget. This would allow later iterations of the merge-and-shrink algorithm to compute factored symmetries again. We leave these considerations as future work.

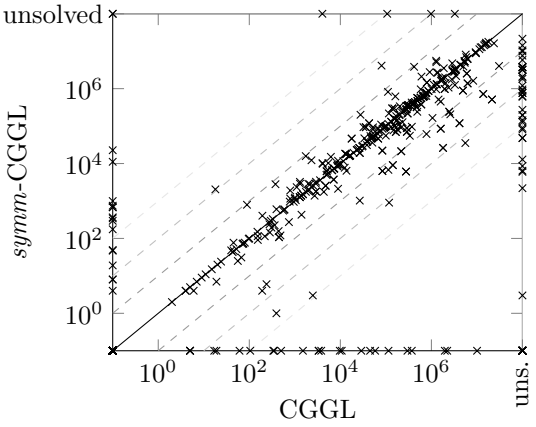
Based on the above analysis of the performance of the different variants of enhancing merge strategies through symmetries, we restrict all our further experiments to the variant *symm-largest* that achieves best or close-to-best coverage and call it *symm* instead of the longer *symm-largest*, writing *symm*-X to denote a symmetry-enhanced merge strategy X. This choice matches all results reported for symmetry-enhanced merge strategies in previous work.

In the following, we go into more detail for the CGGL merge strategy, the merge strategy which profits most from symmetries. We compare the baseline CGGL against the symmetry-enhanced variant *symm*-CGGL. To verify that the improvements obtained through symmetries affects a broad range of domains, Figure 6.5a shows domain-wise coverage. We see that coverage improves in 16 out of 57 domains, whereas it decreases in only 4 domains, and then only by 1 task. In the remaining 37 domains, both variants achieve equal coverage, displayed in an aggregated row.

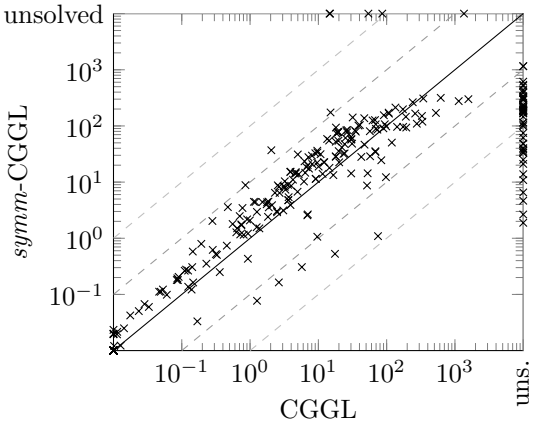
To consider the quality of the resulting heuristic, Figure 6.5b compares the number of expansions of CGGL base against *symm*-CGGL on all domains. Figure 6.6 shows the same data in a way that allows distinguishing the different domains, for clarity restricted to the domains of Table 6.5a where the two strategies do *not* achieve the same coverage. Finally, we also compare the runtime of both configurations on the full benchmark set (Figure 6.5c). While there is a larger number of cases where the symmetry-based configuration requires fewer expansions than the baseline, the clearest distinguishing characteristic is that there are far more tasks that the symmetry-based configuration solves but the baseline does not.

	CGGL		
	orig	*symm*	diff
airport (50)	11	**15**	**+4**
elevators-opt08-strips (30)	12	**13**	**+1**
elevators-opt11-strips (20)	10	**11**	**+1**
gripper (20)	7	**18**	**+11**
logistics98 (35)	**5**	4	−1
miconic (150)	72	**77**	**+5**
parking-opt11-strips (20)	0	**4**	**+4**
parking-opt14-strips (20)	0	**6**	**+6**
pipesworld-notankage (50)	12	**15**	**+3**
pipesworld-tankage (50)	9	**14**	**+5**
satellite (35)	**7**	6	−1
sokoban-opt08-strips (30)	26	**30**	**+4**
tetris-opt14-strips (17)	0	**1**	**+1**
tidybot-opt11-strips (20)	**1**	0	−1
trucks-strips (30)	7	**8**	**+1**
visitall-opt11-strips (20)	9	**10**	**+1**
visitall-opt14-strips (20)	4	**6**	**+2**
woodworking-opt08-strips (30)	11	**12**	**+1**
woodworking-opt11-strips (20)	6	**7**	**+1**
zenotravel (20)	**11**	10	−1
Sum (688)	220	**267**	**+47**
Remaining domains (979)	462	462	**±0**
Sum (1667)	682	**729**	**+47**

(a) Domain-wise coverage of CGGL and *symm*-CGGL with shrink strategy B. The last row aggregates domains with equal coverage. The number in parenthesis after each domain indicates the number of tasks of that domain.



(b) Expansions of CGGL and *symm*-CGGL.



(c) Total runtime of CGGL and *symm*-CGGL.

Figure 6.5.: Detailed comparison of CGGL and *symm*-CGGL.

	CGGL		
	orig	*symm*	diff
Coverage	682	**729**	**+37**
Constr oom	184	**158**	**−26**
Constr oot	145	**142**	**−3**
Search oom	648	**629**	**−19**
Search oot	**4**	**4**	**±0**

Table 6.6.: Comparison of reasons of failure to solve a task for CGGL *orig* and *symm*: merge-and-shrink computation reaching a limit (Constr oom/Constr oot) or search reaching a limit (Search oom/Search oot) (four tasks recognized as unsolvable omitted).

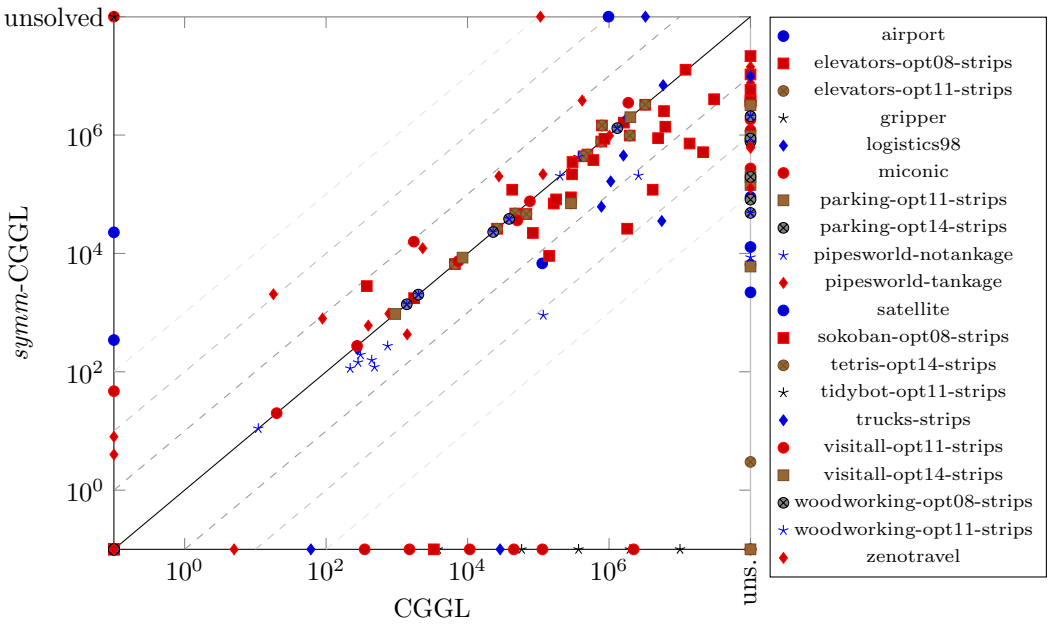


Figure 6.6.: Expansions of CGGL and *symm*-CGGL on domains where expansions are not equal.

To further investigate the reasons of failure of the two CGGL variants in addition to the data reported in Table 6.5 above, Table 6.6 also lists the number of tasks for which the search runs out of memory or time (Search oom/Search oot). We also again display coverage and the number of tasks for which the construction of the heuristic fails. As observed above already, using symmetries drastically decreases the number of tasks where the configuration runs out of memory while also not increasing the failures due to reaching the time limit at the same time. Looking at the reasons of failure of search, we observe a similar trend. These results confirm the previous observations: while the symmetry-enhanced symmetry requires fewer expansions more frequently than the other way round, the "other half" of the gains in coverage is due to the increased efficiency, which transfers to a higher amount of tasks for which the merge-and-shrink computation succeeds.

Finally, we briefly discuss results obtained using the other two shrink strategies, i.e. F and G. Table A.1 in the appendix on page 171 shows a pairwise comparison of the original (*orig*) and symmetry-enhanced (*symm*, in the configuration as used above) variant of each of the the four merge strategies we consider. We observe that with F, coverage only increases with the two linear merge strategies but decreases a lot with DFP. The latter is due to finishing the merge-and-shrink computation for fewer tasks than with the original merge strategy. Using G, the results are even more damning: coverage decreases significantly for all merge strategies, due to not finishing the merge-and-shrink computation for many tasks. For both shrink strategies, this is due to reaching the memory limit for more tasks. This can be explained by the fact that the goal of using factored symmetries for merging is to perform exact bisimulation-based shrinking, exploiting the fact that shrinking based on atomic factored symmetries is captured by non-approximate bisimulations. Neither F nor the greedy bisimulation-based shrinking variant

G capture these shrinking opportunities.

To summarize this part, we showed that using factored symmetries to enhance merge strategies is indeed beneficial. To some extent, the increased performance is due to an increased efficiency in the computation of merge-and-shrink abstractions, but for some variants (e.g. CGGL), the resulting heuristic is also of much higher quality.

6.4. Optimized Implementation

In the following, we compare the non-optimized implementation of merge-and-shrink based on generalized label reduction to the most recent, optimized implementation in Fast Downward. Recall that in contrast to the optimized implementation, in the non-optimized implementation, we do not use locally equivalent labels to group transitions and we do not enforce the valid-state invariant for transition systems. The latter means that pruning only happens before shrinking in contrast to always keeping all transitions pruned according to the prune strategy. Furthermore, transitions are sorted and duplicates are removed only before shrinking and merging but not after merging. Most importantly, for exact label reduction based on Θ-combinability, we have to compute the local equivalence relations on labels for each label reduction (caching local equivalence relations, whenever possible, across all reductions within the label reduction algorithm, but not over several merge-and-shrink iterations). Finally, the computation of product transition systems is more expensive due to not grouping labels in the non-optimized implementation, For more details, we refer to the discussion in Section 3.8.3.

In an ideal world, we would investigate some of the above differences of the implementation in isolation. However, since most of these changes were implemented together and depend on each other, we can only carry out a before-after comparison that includes all improvements we discussed. In the following, we refer to the non-optimized implementation as *base*, to the optimized one as *opt*, and we show the difference *diff* between opt and base and also list the number of tasks for which opt is better (+) and worse (-) than base. As always, unless stated otherwise, we use exact generalized label reduction based on the fixed point algorithm and the shrink strategy B, and all symmetry-enhanced merge strategies (*symm*) correspond to the variant *largest-symm* as discussed in the previous section. Also recall that for MIASM in the optimized implementation, we use the re-implementation provided by its authors, adapted to switch to DFP in case the merge strategy does not find a non-trivial partitioning of the state variables of the planning task, which makes the results of MIASM less comparable.

Table 6.7 shows the comparison for shrink strategy B and the merge strategies we considered so far. We begin with the non-symmetry enhanced strategies of the first two vertical blocks. We observe that the changes in the implementation are indeed improvements with respect to the efficiency of the merge-and-shrink computation. There is a huge decrease in the time required to compute the abstraction (Constr time), which is due to the much smaller amount of transitions required to be stored for transition systems that also transfers to running out of memory and time less frequently. Besides these savings in memory and runtime, with the new valid-state invariant, all dead states are always pruned immediately after each transformation. This effect is visible through the maximum size of intermediate transition systems (MITSS), which decreases for more tasks than for which it increases (except for MIASM, which is a re-implementation and

	base	opt	diff	+	-	base	opt	diff	+	-
	CGGL					RL				
Coverage	682	712	**30**	30	0	720	728	**8**	9	1
Search time	0.30	0.31	0.01	58	305	0.27	0.28	0.01	64	311
Total time	4.35	3.37	**−0.99**	531	75	3.73	2.92	**−0.81**	546	105
Exp 75th perc	**1170k**	**1170k**	0	12	32	**992k**	1353k	361	26	24
# constr	1338	1441	**103**	107	4	1413	1503	**90**	92	2
Constr time	117.16	49.18	**−67.98**	1183	151	95.10	57.34	**−37.76**	1219	192
Constr oom	184	95	**−89**	121	32	112	36	**−76**	96	20
Constr oot	145	131	**−14**	44	30	142	128	**−14**	29	15
MITSS	46802.63	46650.66	**−151.96**	160	94	46420.03	45756.11	**−663.92**	161	89
	DFP					MIASM				
Coverage	736	746	**10**	13	3	746	773	**27**	40	13
Search time	0.29	0.28	**−0.01**	284	114	0.23	0.20	**−0.03**	203	182
Total time	4.42	3.23	**−1.20**	613	53	10.25	3.53	**−6.72**	606	113
Exp 75th perc	1047k	**1040k**	**−7**	40	40	710k	**696k**	**−14**	132	146
# constr	1389	1491	**102**	104	2	1367	1470	**103**	105	2
Constr time	109.68	57.78	**−51.91**	1212	175	105.74	66.75	**−38.99**	819	546
Constr oom	134	42	**−92**	114	22	149	104	**−45**	106	61
Constr oot	144	134	**−10**	31	21	151	93	**−58**	91	33
MITSS	46948.16	45004.23	**−1943.93**	227	140	44440.46	45417.77	977.31	380	384
	symm-CGGL					*symm*-RL				
Coverage	729	741	**12**	20	8	740	743	**3**	10	7
Search time	0.29	0.32	0.02	59	328	0.30	0.32	0.02	75	318
Total time	8.09	5.53	**−2.56**	638	55	7.51	4.66	**−2.85**	661	42
Exp 75th perc	**980k**	1012k	32	35	85	**947k**	1010k	63	64	85
# constr	1367	1468	**101**	105	4	1426	1504	**78**	80	2
Constr time	121.14	86.20	**−34.95**	1260	103	115.12	74.23	**−40.88**	1333	90
Constr oom	158	72	**−86**	114	28	94	36	**−58**	78	20
Constr oot	142	127	**−15**	35	20	147	127	**−20**	32	12
MITSS	45263.84	44970.77	**−293.07**	343	229	46195.63	44546.67	**−1648.96**	444	244
	symm-DFP					*symm*-MIASM				
Coverage	745	752	**7**	10	3	751	771	**20**	31	11
Search time	0.31	0.32	0.00	160	252	0.28	0.24	**−0.04**	157	251
Total time	8.22	5.45	**−2.77**	681	33	14.97	5.56	**−9.41**	653	73
Exp 75th perc	**970k**	971k	1	54	62	**709k**	**709k**	0	118	108
# constr	1405	1496	**91**	94	3	1381	1472	**91**	95	4
Constr time	125.48	81.06	**−44.42**	1284	117	111.52	75.70	**−35.81**	1032	345
Constr oom	114	44	**−70**	93	23	132	102	**−30**	93	63
Constr oot	148	127	**−21**	34	13	154	93	**−61**	95	34
MITSS	46170.15	44452.02	**−1718.13**	312	227	43607.21	44605.48	998.27	401	364

Table 6.7.: Comparison of the non-optimized implementation of merge-and-shrink based on generalized label reduction to the optimized one, for several merge strategies and shrink strategy B. Values aggregated and highlighted for each merge strategy individually.

					symm			
	CGGL	RL	DFP	MIASM	CGGL	RL	DFP	MIASM
Coverage	712	728	746	**773**	741	743	752	771

Table 6.8.: Coverage of the four considered merge strategies and their symmetry-enhanced counterparts in the state-of-the-art implementation, using shrink strategy B.

the comparison must be taken with care). All of these reasons also transfer to a high increase of the number of tasks for which the computation succeeds, which in turn results in better coverage for all configurations. It is also worth noting that neither the number expansions nor the search time decreases on average, but only total time does. Hence the optimizations are indeed only optimizations that affect the efficiency of the computation of the merge-and-shrink algorithm, but mostly not the heuristic quality. We remark that MIASM is much stronger here than reported in previous work (Sievers et al., 2016) because it now uses a fallback mechanism that switches to DFP, as the first implementation of MIASM did.

For the symmetry-enhanced strategies, shown in the last two vertical blocks, the picture is similar: we observe improved performance in terms of construction time and coverage for all strategies, however in a slightly less notable amount. The obtained improvements are smaller for symmetry-enhanced merge strategies than for the original ones because enhancing merge strategies through factored symmetries is, to some extent, an efficiency improvement, and so is using the optimized implementation. Hence combining the two ways of improving the efficiency is not fully orthogonal, however still beneficial.

We now briefly discuss the effect of the optimized implementation on the shrink strategies F and G. Table A.3 on page 172 in the appendix shows the results. For F, we observe the same trends as when using B, i.e. the performance of all merge strategies increases. However the improvements are less significant for all merge strategies except DFP, which profits a lot from the efficiency improvements. For MIASM, coverage even decreases, however this is likely due to the different implementation used in the optimized code base. We also again observe that the symmetry-enhanced variants do not work as well with F as with B. With G, the changes in performance caused by the optimized implementation are even smaller, however still beneficial.

To summarize this part, it is safe to say that the optimizations of the implementation based on generalized label reduction are indeed optimizations in the sense that the merge-and-shrink algorithm can be computed more efficiently.

As a point of reference for the remaining parts of our study, Table 6.8 shows coverage of the merge strategies we investigate for the optimized implementation, evaluated using shrink strategy B. The data is the same as the data shown in column opt of Table 6.7 that compares the non-optimized to the optimized implementation.

6.5. More Variants of Using Symmetries

In this section, we first discuss the new alternative way of enhancing merge strategies with factored symmetries, called *FB*, that we did not evaluate on the non-optimized implementation, since it is not implemented there. FB merges the transition systems affected by the chosen

	CGGL			RL			DFP			MIASM		
	orig	*symm*	FB	*orig*	*symm*	FB	*orig*	*symm*	FB	*orig*	*symm*	FB
Coverage	712	**741**	732	728	**743**	733	746	752	**759**	**773**	771	746
E 50th perc	20k	**20k**	20k	**2132**	9120	3037	**2343**	12k	6528	**1478**	9120	3471
E 75th perc	1168k	**897k**	922k	942k	**639k**	762k	611k	**504k**	**504k**	**491k**	616k	552k
Search time	0.38	**0.31**	0.32	**0.21**	0.24	0.23	**0.20**	0.26	0.23	**0.15**	0.20	0.20
Total time	**4.15**	5.02	4.88	**2.98**	4.43	4.43	**3.40**	5.58	5.75	**2.76**	4.48	6.46
# constr	1441	1468	**1482**	1503	**1504**	1502	1491	**1496**	1488	1470	**1472**	1367
Constr time	**94.24**	100.02	98.29	**85.72**	105.48	106.94	**96.74**	119.98	127.98	**51.46**	58.09	223.35
Constr oom	95	72	**54**	**36**	**36**	37	**42**	44	46	104	102	**43**
Constr oot	131	**127**	131	128	**127**	128	134	**127**	133	**93**	**93**	257
Perfect h	257	**274**	271	265	266	**270**	268	264	**278**	324	306	**334**
Linear tree	100.00	**19.96**	31.51	100.00	**16.56**	29.76	87.79	**18.98**	62.37	73.47	**12.36**	40.60

Table 6.9.: Comparison of original merge strategies (*orig*) against the previous symmetry-enhanced variant (*symm*) and the alternative approach that uses the fallback merge strategy to merge transition systems affected by the chosen symmetry (FB). All algorithms use shrink strategy B. Values aggregated and highlighted for each merge strategy individually.

symmetry in the order computed by the fallback merge strategy that is being enhanced. In the second part of this section, we evaluate alternative ways of exploiting information from symmetries, which are using structural symmetries for pruning the A* search and using symmetric lookups over merge-and-shrink heuristics. In particular, we also combine the three alternatives in all possible ways. This evaluation reproduces some results of the study performed in our work on using symmetries for abstraction heuristics (Sievers, Wehrle, Helmert, & Katz, 2015).

Table 6.9 evaluates the variant FB of symmetry-enhanced merge strategies, comparing it both to the original merge strategy (*orig*) and the previous way of enhancing merge strategies through symmetries (*symm*). Like *symm*, FB chooses symmetries affecting the most factors and uses a time budget of 60s for computations of symmetries.

The general observation is that the results are mixed, i.e. differ from merge strategy to merge strategy. Considering coverage, CGGL, RL, and MIASM are better with *symm* than with FB, only DFP improves over *symm* with FB. Also concerning heuristic quality, i.e. number of expansions and search time, there are no clear trends. With *symm*, the merge-and-shrink computation finishes for more tasks than for FB and results in non-linear merge trees more frequently. To summarize, the merge order for merging the transition systems affected by the chosen symmetry clearly affects the overall performance. However no clear trends allows preferring one method over the other. For this reason, we stick with the previous variant *symm* in the remainder of the study.

We now proceed with evaluating the merge strategy DFP with alternative approaches of using symmetries. In particular, we evaluate symmetric lookups which we introduced to planning in our original paper (Sievers, Wehrle, Helmert, & Katz, 2015), and orbit space search, a representative of symmetry-based pruning algorithms due to Domshlak et al. (2015). See Section 2.4 for a description of both techniques.

To use symmetric lookups and orbit space search in combination with merge-and-shrink heuristics, we need to make sure that the heuristics yield admissible values for all symmetric

	orig	IP	SLone	SLsub5	SLsub10	SL
Coverage	746	745	748	753	**754**	753
Exp 50th perc	11k	14k	11k	8773	**8562**	8773
Exp 75th perc	786k	803k	618k	715k	**582k**	715k
Search time	**0.26**	0.32	0.45	0.52	0.61	0.52
Total time	**3.79**	4.94	5.55	5.93	6.33	5.95

Table 6.10.: Results of original DFP (*orig*) compared to the variant that does not prune unreachable states (IP) and variants of using symmetric lookups over a set of symmetric states of different sizes (SLone/SLsub5/SLsub10/SLall), using shrink strategy B.

states. While this might seem obvious at first glance, admissibility is no longer guaranteed when pruning unreachable states because the symmetries we use do not stabilize the initial state. Hence a non-dead-end state could have a symmetric state which is unreachable and thus pruned. To address this issue, we simply disable this pruning within the computation of merge-and-shrink in all experiments that combine merge-and-shrink heuristics with symmetric lookups or orbit space search.[9]

Following the experimental study of our original paper, we first evaluate several variants of computing a set S of symmetric states of a state s that should be evaluated by the symmetric lookups heuristic h_{SL}. Let Γ be the set of symmetries of the planning task. We always include s itself into S. The most simple variant of computing a single symmetric state is to perform a short "random walk" (here: 5 steps) by repeatedly applying random symmetries from Γ starting from s and adding the resulting state to S. We call this variant *SLone*. For all other variants of computing more symmetric states, we systematically sample symmetric states of the orbit of s by performing a breadth-first search starting from s using the symmetry generators from Γ. We evaluate the variants that generate 5 (*SLsub5*), 10 (*SLsub10*), and all (*SLall*) symmetric states.

Table 6.10 shows the original strategy DFP (*orig*), the variant that does not prune unreachable states (for naming consistency with Section 6.7 called *IP* for pruning irrelevant states), and the discussed variants of using symmetric lookups over the merge-and-shrink heuristic computed with DFP without pruning unreachable states. Besides the observation that not pruning unreachable states apparently is not very harmful to the computation of merge-and-shrink abstractions with DFP (we investigate the impact of pruning in Section 6.7), computing symmetric lookups over merge-and-shrink heuristics slightly decreases expansions for all variants we consider. This decrease in expansions is expected, since we improve the quality of the heuristic by maximizing heuristic values over more states. On the other hand, computing symmetric states for each heuristic evaluation is expensive, as can be seen by the significantly increased search time (and hence total time, since the computation of the merge-and-shrink abstraction is the same in all variants). However, since coverage also increases for all variants, the increase in heuristic quality outweighs the increase in search time.

Comparing the different variants of computing symmetric states, we observe that computing a single symmetric state does not improve performance a lot, but considering *some* and even *all* symmetric states achieves the highest coverage and largest reduction in expansions. Figures 6.7

[9] This contrasts the approach in our original work where with symmetric lookups, we only ignored symmetric dead-ends but did not disable pruning of unreachable states entirely.

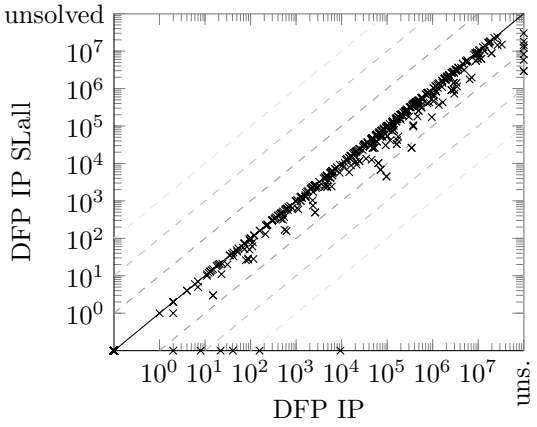


Figure 6.7.: Expansions of DFP without and with symmetric lookups (SLall), both without pruning unreachable states (IP).

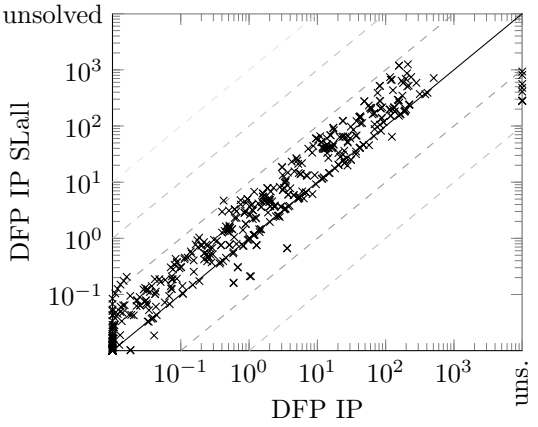


Figure 6.8.: Search time of DFP without and with symmetric lookups (SLall), both without pruning unreachable states (IP).

and 6.8 provide a detailed comparison for the variant SLall against the baseline (without pruning unreachable states), confirming that expansions are strictly decreased and search time increases for most tasks. While SLsub10, for this comparison, achieves slightly better results than SLall, we stick to the variant SLall, simply called "SL" from now on, for the remaining experiments in this section, because we are interested to see how far the expensive increase in heuristic quality fares.

In our original paper, we also evaluated a technique called *bidirectional pathmax* that originated in the heuristic search community (Felner et al., 2011). Informally speaking, BPMX "repairs" inconsistent jumps in the heuristic values for s and successor state $s(o)$ (and vice versa) by adapting the values accordingly. Since the implementation used for our original paper is not compatible with the most recent version of Fast Downward, and because in our original paper, we found the technique to not be beneficial for planning, we drop its evaluation here.

We now continue with evaluating all symmetry-based techniques, i.e. orbit space search (OSS) and symmetry-enhanced merge strategies (*symm*) in addition to the symmetric lookups heuristic (SL), both alone and in all combinations. We extend the evaluation of our original paper to include further merge strategies and do not restrict the results to using DFP.

Table 6.11 shows results, vertically grouping results for CGGL, RL, DFP, and MIASM. Considering coverage, we observe that all single techniques improve compared to the baseline merge strategy.[10] For combined techniques, however, only the combination OSS+*symm* always improves over the best coverage of its components, but all other combinations only sometimes do, such as SL+*symm* for CGGL and RL, but not for DFP and MIASM, OSS+SL for all merge strategies but MIASM, and OSS+SL+*symm* is never better than OSS+*symm*. Concerning expansions, the picture is less clear. Generally, symmetry-based pruning (OSS) improves performance

[10]These results have to be taken with a grain of salt, though, because for merge strategies other than DFP, there is a considerable decrease in coverage due to not pruning unreachable states.

	IP	*symm*	SL	OSS	OSS+*symm*	SL+*symm*	OSS+SL	OSS+SL+*symm*
					CGGL			
Coverage	703	735	710	791	**804**	737	792	802
Exp 50th perc	32k	30k	25k	8244	8994	15k	7872	**5523**
Exp 75th perc	1274k	934k	1270k	583k	414k	819k	467k	**413k**
Search time	0.43	0.34	0.72	0.32	**0.28**	0.61	0.44	0.40
					RL			
Coverage	727	738	728	795	**802**	739	795	800
Exp 50th perc	2771	15k	2709	**2373**	7928	13k	**2373**	5202
Exp 75th perc	945k	717k	917k	350k	314k	603k	327k	**301k**
Search time	0.27	0.29	0.46	**0.22**	0.24	0.50	0.31	0.34
					DFP			
Coverage	745	747	753	809	**813**	751	810	**813**
Exp 50th perc	6233	15k	4795	4597	7928	13k	**3214**	5210
Exp 75th perc	712k	717k	596k	284k	290k	706k	**264k**	283k
Search time	0.25	0.28	0.40	**0.21**	0.23	0.50	0.28	0.33
					MIASM			
Coverage	749	750	756	815	**819**	752	813	815
Exp 50th perc	7545	13k	4795	2310	6895	9635	**1771**	5202
Exp 75th perc	482k	648k	424k	264k	290k	636k	**200k**	283k
Search time	0.23	0.27	0.37	**0.19**	0.23	0.50	0.26	0.33

Table 6.11.: For CGGL, RL, DFP, and MIASM, comparison of different symmetry-based techniques, in isolation and all combinations: original merge strategy without pruning unreachable states (IP), symmetry-enhanced merge strategy (*symm*), orbit space search (OSS), symmetric lookups (SL). All variants use shrink strategy B.

						symm				
	CGGL	DFP	L	MIASM	RL	CGGL	DFP	L	MIASM	RL
Coverage	710	745	704	**757**	725	747	**752**	742	749	749

Table 6.12.: Coverage of the considered merge strategies using shrink strategy B on the Fast Downward version used in the original work (Sievers, Wehrle, & Helmert, 2016).

in terms of coverage, expansions, and search time by far the most, and symmetric lookups (SL) seem to be the least useful if combining it with the other techniques. Still, using symmetric lookups are beneficial if used as a single technique for enhancing merge-and-shrink heuristics.

To summarize this part, we showed that the framework for enhancing merge strategies can still be altered to obtain further improvements, e.g. by changing the merge order when merging according to information from factored symmetries. Furthermore, we compared using factored symmetries to symmetry-based pruning and symmetric lookups. While symmetry-based pruning is much stronger than the other two techniques, its performance can be further improved by combining it with merge-and-shrink heuristics that use symmetry-enhanced merge strategies.

6.6. An Analysis of Merge Strategies

In this section, we analyze the untapped potential of merge strategies by investigating all possible merge strategies on small planning tasks and large sample sets of random merge strategies on the entire benchmark set. Further, we discuss the impact of tie-breaking on the score-based merge strategy DFP and then evaluate our score-based variant of MIASM, sbMIASM, using the same tie-breaking criteria. Finally, we also evaluate the SCC framework for merge strategies, using DFP and sbMIASM with the same tie-breaking strategies as secondary merge strategies.

The evaluation of strategies in this section follows the study of our original paper (Sievers et al., 2016), and some of the text in this section, in particular in Section 6.6.1, is borrowed from this paper.

6.6.1. Considering All and Random Merge Strategies

Many of the results reported for merge-and-shrink (and generally in the planning literature) are of the form "strategy X solves more tasks than Y" or "strategy X requires fewer expansions than Y". While certainly useful, this tells us little about the quality of X and Y in absolute terms. Is X strong and Y poor? Is Y strong, but X even stronger? Can we do better than X? In this small part of our experimental study, we aim at analyzing the potential of (non-linear) merge strategies and to see how existing merge strategies fare with respect to that potential. To do so, we consider all possible merge strategies on small planning tasks and large sets of randomly sampled merge strategies on all planning benchmarks.

Since generating the data of all merge strategies for small planning tasks and large sets of random merge strategies for *all* planning tasks requires millions of planner runs and takes more than a month even on a large compute cluster, we could not regenerate the data for the latest Fast Downward version and instead reuse the data of our original work. While this is certainly

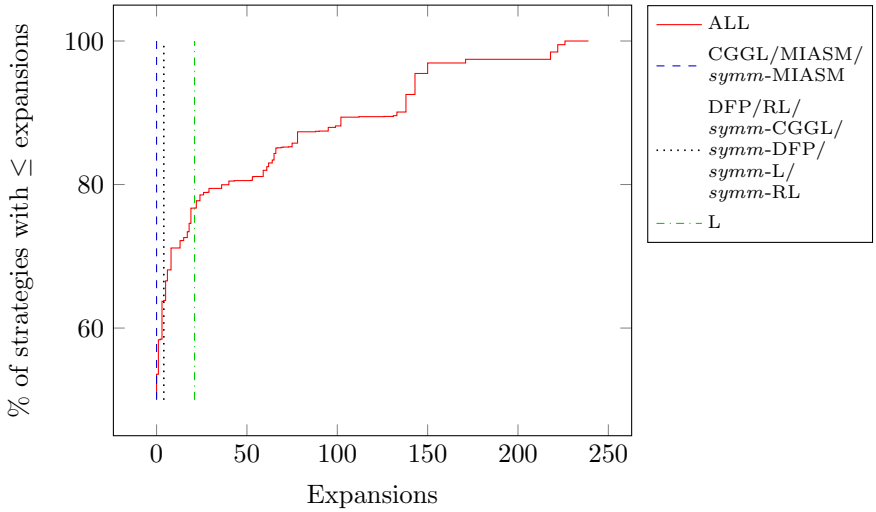


Figure 6.9.: Expansions for Zenotravel #5 with all 1587600 possible merge strategies.

not ideal, we hope that given that we consider all or large sets of random merge strategies, the qualitative results remain the same even if the absolute numbers could be slightly better with the most recent version of Fast Downward. For this reason, Table 6.12 shows overall coverage (number of tasks solved) of the four merge strategies we consider and L (since it is included in the original data) as well as their symmetry-enhanced variants, however generated with the Fast Downward version used in the original work. Differences to the values shown in Table 6.8 are due to different Fast Downward versions, due to not using DFP as a fallback mechanism in MIASM and due other small differences in the merge-and-shrink implementations.

In our first experiment, we consider small planning tasks where we can compare the strategies from the literature to *all possible* merge strategies. This is only feasible for tasks with up to 8 state variables (which already give rise to 1587600 merge strategies). Most such tasks are so simple that all merge strategies result in a perfect heuristic, but there are exceptions. Here, we report results for *Zenotravel #5*.

Figure 6.9 shows the quality of all merge strategies for this instance as a cumulative distribution function. For example, a data point at $(62, 83.0\%)$ means that 83% of all merge strategies require 62 or fewer expansions to reach the final f-layer of an A* search. The curve starts at $(0, 50.2\%)$, showing that 50.2% of *all* merge strategies reach the final f-layer immediately. Given this, the results for the merge strategies from the literature may appear somewhat disappointing: only CGGL, MIASM and *symm*-MIASM have 0 expansions until the final f-layer; DFP, RL and all other symmetry-enhanced variants require 4, and L requires 21.

For larger planning tasks, evaluating all merge strategies is infeasible. However, it is still possible to assess how strong a given merge strategy is in absolute terms by *sampling* a large subset of random merge strategies. We conducted experiments with 1000 merge strategies per task on the entire benchmark set. This showed that the existing merge strategies are quite well suited for many planning domains: The expected coverage of random merge strategies is 680.107, worse than any strategy from the literature we consider. Moreover, we found 72 tasks in 19 domains

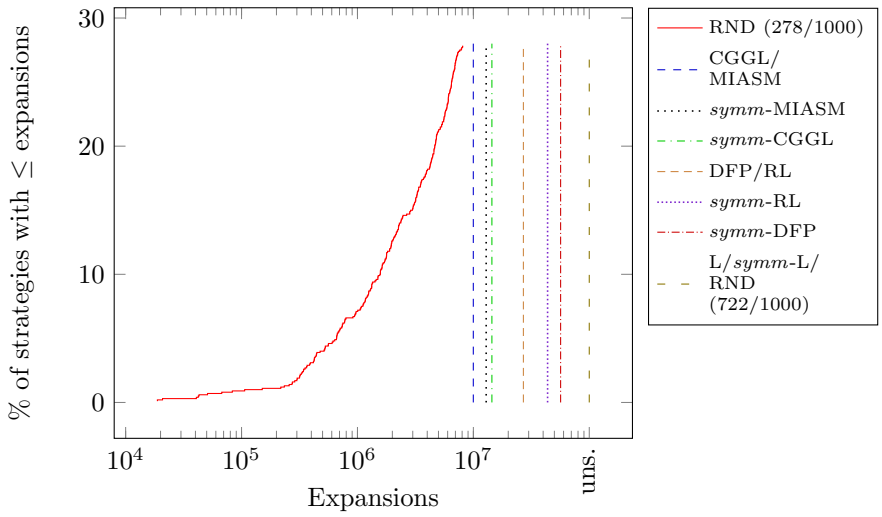


Figure 6.10.: Expansions for NoMystery-2011 #9 with 1000 random merge strategies and those from the literature.

which were solved by at least one merge strategy from the literature but by *none* of the 1000 random merge strategies.

However, interestingly, we also found 21 tasks in 9 domains solved by at least one random merge strategy but by no strategy from the literature. For example, in the NoMystery-2011 domain, only 18 tasks are solved by existing strategies, but all 20 tasks are solved by some random strategy. Moreover, solving these tasks is not a rare occurrence, with most of them solved by all 1000 random strategies and the hardest one solved by 26.4%. Another domain with clear room for improvement is Elevators-2008, where only 18 tasks are solved by any of the existing strategies, but 22 tasks are solved by some random strategy.

In these two domains, we performed additional experiments with the merge strategies from the literature with no time limit and a 64 GiB memory limit in order to determine how much better the good random strategies (with the regular limits of 2 GiB and 30m) are compared to the state of the art. Figures 6.10 and 6.11 show the results (again as cumulative distribution function of expansions) for two of these tasks.[11] For example, we see that in NoMystery-2011 #9, the best existing merge strategies require roughly 1000 times as many expansions as the best random ones. The results look similar for other instances of these two domains (not shown). In particular, the best random merge strategy *always* performs strictly better on all non-trivial tasks which are not perfectly solved by any merge strategy.

The experiment also showed that in 8 of the 11 tasks that the RL strategy can solve in Elevators-2008, its heuristic quality is equal to the *worst* of the 1000 random merge strategies. Similarly damning, the L strategy is consistently as bad as the worst 5% of random strategies in the NoMystery-2011 domain.

[11] The y-axis only goes up to 28% (19%) because only 278 (192) out of 1000 random merge strategies solved these instances. However, also note that *none* of the considered merge strategies from the literature (except *symm*-DFP in Elevators-2008 #7) can solve these instances within the regular time and memory limits.

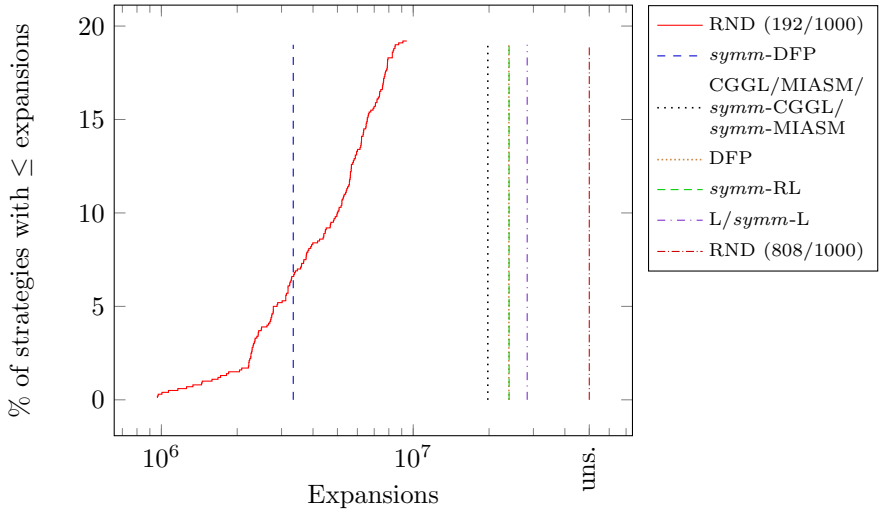


Figure 6.11.: Expansions for Elevators-2008 #7 with 1000 random merge strategies and those from the literature.

To summarize this first part of our analysis, our experiments show that current merge strategies clearly have an untapped potential of being significantly improved, since some these merge strategies have a notable gap of performance compared to large sets of random merge strategies. However, it remains unclear *how* to improve existing merge strategies or how to design new ones to fully use their potential.

6.6.2. The Impact of Tie-breaking on DFP

Next, we take a closer look at the merge strategy DFP. Recall that DFP is score-based and that it prefers to merge transition systems if there exist labels that are relevant for both transition systems and that occur in transitions close to the abstract goal states. Since we observed that in practice, DFP evaluates many candidate pairs of transition systems with the same best score, we investigate several tie-breaking strategies for DFP. In particular, we determine a total order on transition systems which induces a total order of merge candidate pairs. The first choice is to prefer composite (*PC*) or atomic transition systems (*PA*). For the atomic transition systems, we consider the three orders reverse level (RL), level (L), and random (RND). For the composite transition systems, we consider the three orders new to old (NTO), old to new (OTN), and random (RND). We remark that all results reported in the original paper use NTO for the third choice. Finally, we also consider a fully randomized order (Random). Again note that the combination PC/RL/NTO corresponds to the implementation of DFP as reported in the literature. For more details, we refer to Section 5.4 on page 108.

Table 6.13 shows the result of an experiment with all tie-breaking strategies. Considering coverage, we observe a huge variability in performance, ranging from 696–760 solved tasks, from *worse* than the worst merge strategies (combined with the best shrink strategy B) in the literature (coverage of 712 with CGGL in Table 6.8) to *better* than the best merge strategy,

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	**746**	**746**	**746**	729	731	731	696	696	696	707
Exp 50th perc	**1267**	**1267**	**1267**	5725	5725	5725	19k	19k	19k	14k
Exp 75th perc	**539k**	**539k**	**539k**	786k	786k	786k	1025k	1025k	1025k	1163k
Search time	**0.18**	0.18	0.18	0.24	0.24	0.24	0.30	0.29	0.30	0.36
# constr	1491	1491	**1492**	1435	1438	1437	1479	1479	1480	1477
Constr time	61.86	61.97	61.97	86.97	86.37	86.63	73.69	73.91	74.08	**55.72**
Constr oom	**42**	**42**	**42**	93	93	93	52	52	52	57
Constr oot	134	134	**133**	139	136	137	136	136	135	**133**
Perfect h	268	268	268	**273**	272	271	228	229	228	221
Linear tree	87.79	87.79	87.80	88.78	88.80	88.80	90.20	90.26	90.27	**12.46**
	Prefer atomic (PA)									
Coverage	727	711	720	**760**	735	736	733	708	723	
Exp 50th perc	11k	9332	5407	**1690**	5178	4010	5613	11k	8096	
Exp 75th perc	719k	936k	884k	**364k**	718k	656k	826k	871k	862k	
Search time	0.27	0.27	0.26	**0.17**	0.22	0.23	0.26	0.28	0.26	
# constr	1477	1477	1480	**1489**	1480	1479	1478	1469	1474	
Constr time	89.15	79.91	83.15	88.36	80.76	83.47	85.10	**79.30**	81.70	
Constr oom	57	61	55	**49**	57	59	57	65	59	
Constr oot	133	**129**	132	**129**	130	**129**	132	133	134	
Perfect h	239	223	228	**251**	224	233	228	202	216	
Linear tree	8.73	8.73	8.72	8.87	8.92	8.92	**8.66**	8.71	8.68	

Table 6.13.: DFP with different tie-breaking strategies, using shrink strategy B.

excluding MIASM, which also uses DFP as a fallback mechanism.

We first consider the upper half of the table where we fix the first parameter to preferring composite transition systems. We observe that the order of atomic transition systems has a huge impact on performance: RL achieves higher coverage than L which in turn is better than RND, which achieves a lower coverage even than the fully randomized tie-breaking strategy Random. On the other hand, the third parameter, i.e. the order of composite transition systems (NTO, OTN, or RND), does not influence performance a lot: coverage of these variants is nearly the same for a fixed order of atomic transition systems. This can be explained by looking at the percentage of linear merge strategies among these variants, which is very high (more than 88%) for all of them. This means that after the initial merge (which necessarily merges two atomic transition systems), PC tends to repeatedly include the only existing composite in the next merge (hence there is no need to order composite transition systems), which leads to a linear merge strategy.

Considering the other attributes shown in the table, they mostly fit the observation with respect to coverage: within each vertical block of the three variants RL, L, and RND, performance is very similar, and across the three blocks, it follows the comparison of coverage. For example, the variant RL for ordering atomic transition systems, which achieves the highest coverage, also requires the fewest expansions, shortest runtime for search and for heuristic construction (except for Random, which has the lowest construction time), which it also completes more often than the other variants, and it runs out of memory and time during the merge-and-shrink computation less frequently than the other variants. The variant L solves significantly fewer tasks, mostly

because it finishes the merge-and-shrink computation less frequently, and the randomized order produces heuristics of worse quality.

We now consider the variant PA shown in the bottom half of the table. Here, both the second and third parameter, i.e. the order of atomic and composite transition systems, have a large influence on performance. For the order of atomic transition systems, L achieves the best performance in terms of all attributes, except construction time, independently of the other parameters, whereas the performance of RL is only on the level of the performance of the randomized order RND. Concerning the order of composite transition systems for a given order of atomic transition systems, NTO always performs significantly better than the alternatives OTN and RND. Another notable difference to the results of the upper half of the table is that all merge strategies compute non-linear merge trees very frequently (less than 9%). While this does not necessarily lead to higher coverage, the best variant of all variants we consider here is the combination Prefer Atomic/L/NTO which computes a high percentage of non-linear merge trees.

We also briefly discuss results for shrink strategies F and G, shown in Table A.4 on page 173 in the appendix. Also with F, tie-breaking has a notable effect on coverage which ranges from 515–596, and we observe mostly similar trends as with B: preferring composite transition systems results in mostly linear merge strategies and a removal of the influence of the order of composite transition systems. Unlike when using B, preferring atomic transition systems clearly dominates preferring composite transition systems, and also the fully randomized tie-breaking variant is among the top-performing variants. With G, conforming with our previous observations for this shrink strategy, the computed heuristics are of the same or similar quality for all variants (mostly the same amount of expansions), and the major difference appears being the efficiency of the computation of the heuristic. While most variants achieve a coverage ranging in 553–560, the variants Prefer composite/L and the variant PA/L/OTN achieve a much higher coverage of 605 and 597, exclusively due to finishing the computation of the heuristic for many more tasks.

To summarize, these results clearly indicate that there is more to the DFP merge strategy than initially meets the eye and that it is very susceptible to tie-breaking strategies. As the strategy is a simple score-based merge strategy, one way to alleviate the problem of tie-breaking is to combine the strategy with other simple merge strategies or to integrate it into more complex ones. To better use the potential of DFP directly by e.g. improving its evaluation function, a better understanding of the approach is needed.

For the remainder of our study, we will evaluate both the "previous" variant of DFP that corresponds to PC/RL/NTO and the best variant Prefer atomic/L/NTO.

6.6.3. The Score-based MIASM strategy

We continue with an evaluation of our score-based variant of the original MIASM strategy, using the same tie-breaking strategies for the evaluation. Table 6.14 shows the results in the same way as Table 6.13 for DFP.

The results in the table show similar trends as those for DFP, but linear strategies with PC are somewhat less common (45%–53%) because there are more cases where merging is driven by differences in scores rather than tie-breaking. This is also reflected in the smaller overall variance of coverage, which ranges from 721–755, and the fact that the order of composite transition systems also plays a role if preferring composite transition system. Unlike for DFP, where

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	**750**	748	745	726	734	730	722	727	722	739
Exp 50th perc	**449**	572	460	3809	1980	1799	1199	1214	485	1502
Exp 75th perc	443k	443k	**396k**	537k	518k	502k	562k	562k	562k	543k
Search time	**0.17**	0.18	0.18	0.21	0.20	0.20	0.18	0.18	0.19	0.18
Total time	5.50	5.28	5.45	6.83	6.56	6.75	6.67	6.41	6.44	**4.67**
# constr	1324	1331	1337	1299	1304	1304	1319	1326	1315	**1388**
Constr time	173.79	165.78	175.22	219.32	204.88	213.95	212.14	200.54	204.90	**121.17**
Constr oom	**42**	**42**	**42**	44	44	43	44	44	44	43
Constr oot	301	294	288	324	319	320	304	297	308	**236**
Perfect h	332	332	**334**	316	316	316	309	311	313	297
Linear tree	44.94	44.70	44.58	52.50	52.30	52.30	51.33	51.06	51.48	**14.84**
	Prefer atomic (PA)									
Coverage	747	**755**	752	750	748	742	737	745	741	
Exp 50th perc	1932	1396	**1056**	2065	5914	5367	2828	5481	4214	
Exp 75th perc	556k	714k	583k	589k	**521k**	666k	892k	574k	688k	
Search time	**0.18**	0.20	0.18	0.20	0.21	0.23	0.23	0.21	0.21	
Total time	5.49	**4.71**	4.94	5.61	4.93	5.49	5.82	4.72	5.27	
# constr	1384	1403	1408	1374	1384	1379	1389	**1409**	1405	
Constr time	168.71	**135.34**	141.81	164.49	141.92	155.61	169.50	143.75	149.31	
Constr oom	45	51	43	43	**42**	44	43	43	44	
Constr oot	238	**213**	216	250	241	244	235	215	218	
Perfect h	307	310	303	**314**	311	306	301	286	300	
Linear tree	10.12	**9.98**	10.01	10.19	10.12	10.15	10.22	10.08	10.11	

Table 6.14.: sbMIASM with different tie-breaking strategies, using shrink strategy B.

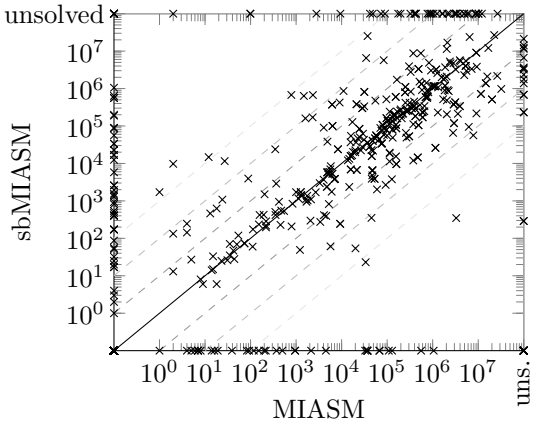


Figure 6.12.: Expansions of MIASM (with DFP PC/RL/NTO as fallback) and sbMIASM (PA/RL/OTN).

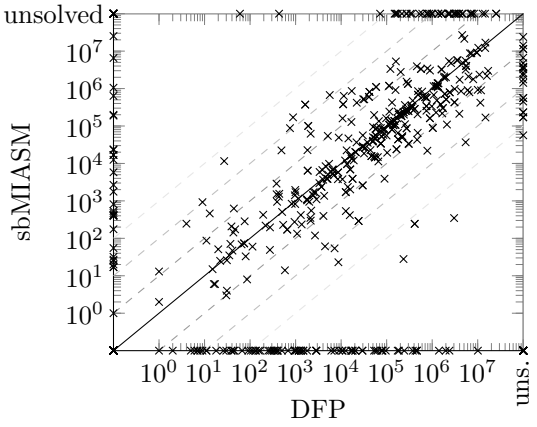


Figure 6.13.: Expansions of DFP (PA/L/NTO) and sbMIASM (PA/RL/OTN).

NTO always dominated the other orders, we do not observe that any of the orders of composite transition systems NTO, OTN, or RND would always perform better than the others. While the order RL for atomic transition systems is the strongest if preferring composite transition systems, there is no clear best single option if preferring atomic transition systems.

Another notable difference to the results of DFP is the time required to compute the merge-and-shrink abstractions, which is very high with sbMIASM, since it requires (tentatively) performing all possible merges in order to compute the scores. Still, the results show that this effort is usually not prohibitive. One reason could be the higher quality of heuristics, as indicated by the smaller amount of expansions required compared to DFP and the much higher number of tasks for which perfect heuristics can be computed.

We also briefly discuss results for shrink strategies F and G, shown in Table A.5 on page 174 in the appendix. With F, the impact of tie-breaking is much smaller than with B: coverage ranges from 569–586, and there is not always a clearly visible correspondence between those variants with highest coverage and those with lowest expansions, total or construction time. Like when using B, preferring composite transition systems results in a smaller fraction of non-linear merge strategies. With G, we again observe that all variants compute heuristics of the same or similar quality (same amount of expansions), but also coverage has a very small range of 558–571, unlike for DFP. The percentage of linear merge trees is very small even when preferring composite transition systems, and is the smallest of all merge-and-shrink variants we considered so far for the variant preferring atomic transition systems.

Focusing again on the shrink strategy B, we compare the best variant Prefer atomic/RL/OTN against the original MIASM strategy and other merge strategies: sbMIASM solves 755 tasks, which is still behind MIASM that solves 773 tasks (cf. Table 6.8), but better than any other non-symmetry-enhanced merge strategy, including DFP in its previously known form (corresponding to PC/RL/NTO). Figures 6.12 and 6.13 compare expansions of MIASM[12] against sbMIASM in

[12]MIASM uses previous DFP, i.e. with tie-breaking Prefer composite/RL/NTO as fallback. In Section 6.8, we

		RL			L			RND			Random	Linear
		NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND		
top	DFP Pref comp	776	777	**778**	752	753	753	734	734	734	733	CGGL
	DFP Pref atom	751	730	736	**764**	748	744	741	734	741		744
	sbMIASM Pref comp	**765**	756	750	739	736	734	746	741	742	761	RL
	sbMIASM Pref atom	767	**769**	766	766	763	757	757	762	764		762
rtop	DFP Pref comp	**778**	774	777	753	748	751	736	731	731	732	CGGL
	DFP Pref atom	753	727	733	**766**	745	746	743	731	734		744
	sbMIASM Pref comp	**766**	755	753	742	733	736	747	739	738	757	RL
	sbMIASM Pref atom	**769**	767	768	768	761	759	758	760	759		762
inc	DFP Pref comp	**780**	774	777	753	748	752	736	731	731	732	CGGL
	DFP Pref atom	755	727	739	**766**	743	744	745	731	734		746
	sbMIASM Pref comp	**766**	754	752	741	733	736	746	738	737	756	RL
	sbMIASM Pref atom	**770**	767	767	768	761	759	758	760	759		761
dec	DFP Pref comp	**778**	774	774	754	747	749	736	730	732	728	CGGL
	DFP Pref atom	754	728	733	**766**	743	740	745	731	739		744
	sbMIASM Pref comp	**766**	758	752	741	736	735	746	741	741	759	RL
	sbMIASM Pref atom	**769**	**769**	766	768	764	758	759	763	762		761

Table 6.15.: Coverage of SCC-X for merge strategies DFP and sbMIASM in all tie-breaking variants, and merge strategies CGGL and RL. All variants use shrink strategy B.

its best variant, and DFP in its best variant against sbMIASM in its best variant. We see that sbMIASM is orthogonal to both state-of-the-art merge strategies. Given these observations, given that sbMIASM is much simpler to implement, and given that it is even more expensive to compute than MIASM, we think that simple score-based merge strategies offer much potential for further research.

For the remainder of our study, when writing sbMIASM, we always refer to the tie-breaking variant PA/RL/OTN that leads to the best performance of sbMIASM.

6.6.4. The SCC Framework

Finally, we evaluate the SCC framework for merge strategies. In our original paper, we only used the topological sort of SCCs and the merge strategy DFP to decide on the merge order both when merging atomic transition systems corresponding to an SCC and for merging the resulting product systems. Here, we also use CGGL, RL, and sbMIASM as secondary merge strategies and consider the four orders of SCCs discussed in Section 5.4: topological (top), reverse of topological (rtop), and decreasing (dec) or increasing (inc) in the size of the SCCs, breaking ties with the topological order.

Table 6.15 shows coverage of the four different orders of SCCs in vertical blocks as indicated in the first column, using the mentioned merge strategies, with all tie-breaking variants for DFP and sbMIASM. The linear merge strategies are shown separately in the right-most column. Best results of each row are highlighted in bold.

We first compare the impact of the order of SCCs, hence comparing the four vertical blocks, leaving all other aspects invariant (i.e. comparing the third entry of the first row of the first

evaluate MIASM with different fallback strategies.

block to the third entry of the first row of the second block). This comparison reveals no large differences in coverage of merge strategies between all four variants, the largest difference being 4 (e.g. top vs inc for DFP PC/RL/NTO, 776 vs 780). This result is not very surprising, since after merging all atomic factors of all SCCs, the resulting factored transition system contains the same elements, independently of the order in which the SCCs are considered. The only influence the order of SCCs has is the order of the resulting product factors in the intermediate factored transition system, which can play a role for future decisions of merge or label reduction strategies.

Next, we compare the results to plain CGGL and RL (cf. Table 6.8), DFP (cf. Table 6.13), and sbMIASM (cf. Table 6.14): independently of the order of SCCs (vertical blocks), using the SCC framework always improves coverage compared to using the plain strategies. This important observation shows that the hybrid approach of the strategy – combining a priori information from the causal graph and using simple merge strategies for the remaining merge decisions – effectively pays off.

In the following, we stick to the topological order of SCCs, which seems to be the most natural choice, and which achieves the highest overall coverage summed within each of the four alternatives (30060 compared to 30030, 30033, and 30039 for rtop, inc, and dec). We observe that with DFP and sbMIASM, the different tie-breaking strategies have similar impact as with the original merge strategies. For example, using DFP with preferring composite transition systems, the order RL of atomic transition systems is the best choice, and if preferring atomic transition systems, L is the best. However, the best performance of SCC-DFP is achieved with the combination PC/RL/RND (closely followed by the variants that substitute NTO or OTN for RND), which does not match the best variant of DFP if used alone, which is DFP Prefer atomic/L/NTO. While using the linear merge strategy CGGL as secondary merge strategy in the SCC framework cannot compete with high coverage values of DFP and sbMIASM, using RL comes a bit closer, solving 762 tasks compared to the best result of 780. For the interested reader, Table A.6 on page 175 in the appendix contains more detailed results that allow a direct comparison to plain CGGL, RL, DFP, sbMIASM also with respect to other attributes than coverage.

In absolute terms, the SCC framework sets a new record of coverage (780) compared to any of the previous merge-and-shrink combinations, where the best competitor is MIASM that solves 773 tasks (cf. Table 6.8). Figure 6.14 compares the expansions of MIASM (still with the variant DFP PC/RL/NTO as fallback strategy) against SCC-DFP in the same tie-breaking configuration (although this variant only solves 776 tasks). We observe that both merge strategies are orthogonal, requiring orders of magnitude more (or fewer) expansions for many tasks. Figure 6.15 compares the expansions of sbMIASM (755 coverage) against SCC-sbMIASM (769), with the tie-breaking PA/RL/OTN that achieves the best performance both with original sbMIASM and SCC-sbMIASM. In this comparison, the two merge strategies are more similar to each other, requiring the same amount of expansions (points on the diagonal) for the majority of tasks, however the SCC-enhanced variant solves more tasks.

Finally, we briefly comment on results for SCC-enhanced merge strategies (topological order of SCCs) with shrink strategies F and G, shown in Tables A.7 and A.8 on pages 176 and 177 in the appendix. With F, we observe similar variations in performance of SCC-DFP as with B: coverage ranges from 520–592, however only the entirely randomized tie-breaking variant Random and the variant PA/L/RND achieve the highest coverage values, mainly due to finishing

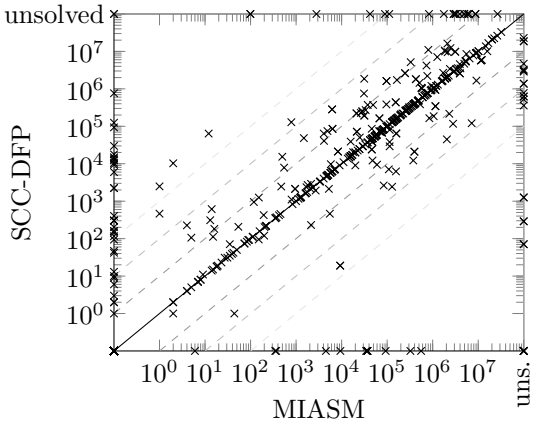


Figure 6.14.: Expansions of MIASM (DFP PC/ RL/NTO) and SCC-DFP (RL/ NTO).

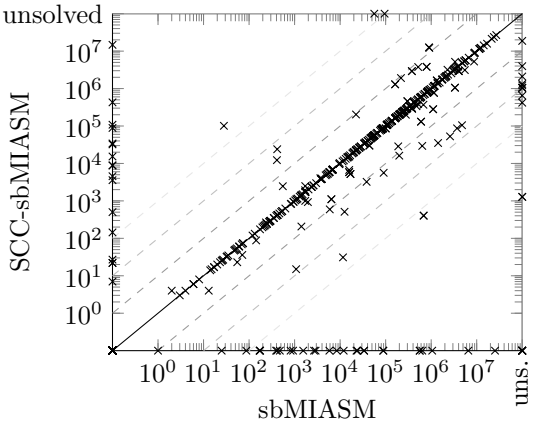


Figure 6.15.: Expansions of sbMIASM and SCC-sbMIASM, both with PA/ RL/OTN.

the merge-and-shrink computation for more tasks. With SCC-sbMIASM, the variation is very small, but still non-negligible. With G, the results of SCC-DFP are very similar to those of DFP: only few variants achieve a high coverage and the remaining ones are all on a similar but lower level. For SCC-sbMIASM, the variance of performance is again much smaller, to the point that no clear recommendation on which tie-breaking is best can be made.

To conclude, the SCC framework for merge strategies, which we consider producing hybrid merge strategies because it integrates a "global analysis" (partitioning variables of the planning task) of precomputed merge strategies with simple (score-based) merge strategies, produces the strongest known merge-and-shrink heuristics. The merge strategies it computes are still susceptible to tie-breaking like the original merge strategies used as secondary merge strategies, which, on the negative side, means that these merge strategies are fragile. On the positive side, this means that there is still considerable room for stronger merge-and-shrink heuristics and that investigating better merge strategies appears to be a fruitful direction for future research.

6.7. The Impact of Pruning

In this section, we evaluate the impact that pruning has on the computation of merge-and-shrink heuristics. While pruning has always been present in merge-and-shrink implementations to the extent that transition systems have been fully pruned, we described pruning as a configurable transformation of the merge-and-shrink framework and also implemented it as a separate, configurable step. As discussed in Section 3.5, we consider two different techniques for pruning: the *original* variant which has previously been used, where dead states and their transitions are entirely removed from transition systems rather than mapping the dead states to a special state. The alternative approach formalizes pruning as *abstraction*, thus mapping dead states to special states but keeping the transitions. Furthermore, we evaluate four different variants of pruning strategies: full pruning (*FP*), where all dead states are pruned, pruning of only irrelevant states

(*IP*), pruning of only unreachable states (*UP*), and no pruning at all (*NP*).

Table 6.16 shows results for the merge strategies CGGL, RL, DFP (tie-breaking configuration PA/L/NTO), MIASM, sbMIASM (PA/RL/OTN), and SCC-DFP (PC/RL/NTO). The first impression is that FP in the previous implementation (*original*) is indeed the best variant of all. It achieves the highest coverage among all variants (except for merge strategy RL), finishes the merge-and-shrink computation for the highest number of tasks, requires the shortest runtime (total time), and computes perfect heuristics for the highest number of tasks. Also for the other attributes, FP usually achieves the best result (bold values in the first data column). This comes as expected, since pruning in general increases the efficiency of the computation due to reduced memory and runtime requirements. In some cases, this can even transfer to an improved heuristic because the savings in sizes of the transition systems allows for fewer lossy shrink transformations. Furthermore, pruning by removing states has a potential positive side-effect on all transformation strategies that consider states and/or transitions, such as bisimulation-based shrinking or the DFP merge strategy.

However, there is more to pruning than initially meets the eye. For example, with RL, full pruning (FP) is better if used with *abstraction* than with *original* pruning. Furthermore, with *original* pruning, pruning of unreachable states appears to be a lot more important than pruning of irrelevant states: for all six merge strategies, the variant UP achieves coverage much closer to full pruning (FP) compared to only pruning irrelevant states (IP), which lags behind significantly. The reason is that the total runtime is higher on average with IP, and the resulting heuristics are perfect for significantly fewer tasks. With *abstraction* pruning, the effect is still present (except for DFP), however less pointed. A possible reason is the interaction of pruning with bisimulation: since bisimulation considers the *outgoing* transitions of states (cf. Definition 3.14) and states must have the "same behavior" with respect to their successor states and hence ultimately some goal states, irrelevant states are less of a concern than unreachable states. All irrelevant states have the same h-value of ∞ since they cannot lead to goal states, which makes them candidates for being bisimilar in the first place. For unreachable states, the same is not true: they can have different h-values, in which case they cannot be bisimilar. Hence, completely removing unreachable states allows finding more bisimulations. This observation also explains why pruning as abstractions, where unreachable states are condensed into a single state, is also less effective in combination with bisimulation-based shrinking.

The results for F and G, shown in the appendix in Tables A.9 and A.10 on pages 178 and 179, also confirm this hypothesis: with F, pruning only irrelevant states, i.e. not pruning unreachable states, is better than the other way around for half of the six merge strategies, whereas with G, like with B, UP strictly dominates IP, usually by large margins, and comes close to the performance of FP (except for SCC-DFP). Other than that, the results for F generally also have a clear tendency that *original* full pruning (FP) is the best choice, with a few minor exceptions such as sbMIASM, where pruning as abstraction solves slightly more tasks. Similarly with G, *original* full pruning (FP) is the best performer except for MIASM, where *abstraction* pruning of reachable states (UP) solves three tasks more.

Concluding this part, we saw that pruning is indeed an important ingredient to the efficient computation of merge-and-shrink heuristics, increasing the number of successful computations of merge-and-shrink heuristics which also transfers to higher coverage. Furthermore, we observed that with bisimulation-based shrinking B, pruning of unreachable states is particularly

	original				*abstraction*			*original*				*abstraction*		
	FP	IP	UP	NP	FP	IP	UP	FP	IP	UP	NP	FP	IP	UP
	CGGL							RL						
Coverage	**712**	703	711	705	710	704	711	728	727	**731**	728	**731**	728	**731**
E 50th perc	**15k**	28k	**15k**	25k	**15k**	25k	**15k**	**5353**	9911	**5353**	9911	**5353**	9911	**5353**
E 75th perc	**1197k**	1304k	**1197k**	1304k	**1197k**	1304k	**1197k**	**1246k**	**1246k**	**1246k**	**1246k**	**1246k**	**1246k**	**1246k**
Search time	**0.36**	0.42	0.36	0.42	0.36	0.42	0.36	**0.28**	0.34	0.29	0.34	0.29	0.34	0.29
Total time	**3.98**	5.51	4.27	5.66	4.56	5.86	4.75	**3.48**	4.52	3.74	4.76	3.90	4.65	4.14
# constr	**1441**	1438	1439	1439	1438	1440	1438	**1503**	1495	1502	1497	1494	1495	1493
Constr time	**81.17**	83.00	83.03	81.52	92.18	86.80	89.33	**87.95**	89.62	88.93	89.48	92.93	90.37	90.92
Constr oom	95	98	94	97	**92**	94	95	**36**	43	37	43	44	43	44
Constr oot	**131**	**131**	134	**131**	137	133	134	128	129	128	**127**	129	129	130
Perfect h	**257**	209	**257**	207	**257**	207	**257**	**265**	226	260	225	261	226	260
	DFP PA/L/NTO							MIASM						
Coverage	**760**	747	759	744	744	747	746	**773**	749	767	749	767	751	764
E 50th perc	**7182**	12k	**7182**	12k	12k	12k	12k	**9038**	14k	**9038**	14k	9073	14k	9073
E 75th perc	**1178k**	**1178k**	**1178k**	**1178k**	1192k	**1178k**	1192k	768k	857k	768k	857k	**751k**	857k	768k
Search time	**0.28**	0.33	0.28	0.33	0.30	0.33	0.30	**0.23**	0.30	0.24	0.30	0.23	0.30	0.24
Total time	**3.20**	3.92	3.32	3.83	3.69	3.98	3.76	**4.34**	5.94	4.66	6.25	4.94	6.18	5.20
# constr	**1489**	1486	1482	1482	1479	1486	1477	**1470**	1441	1465	1439	1451	1441	1450
Constr time	89.29	88.84	89.68	**87.96**	98.35	89.76	95.46	**96.10**	97.78	97.88	97.52	108.14	101.85	106.68
Constr oom	**49**	52	55	57	56	52	57	**104**	130	108	134	119	131	121
Constr oot	129	129	130	**128**	132	129	133	**93**	96	94	94	97	95	96
Perfect h	**251**	214	250	212	240	213	240	**324**	256	317	255	319	256	316
	sbMIASM PA/RL/OTN							SCC-DFP PC/RL/NTO						
Coverage	**776**	752	773	753	760	755	757	**755**	713	731	707	749	711	727
E 50th perc	7051	9278	7051	9278	**4445**	9278	**4445**	**1502**	4470	**1502**	7025	1603	7025	1547
E 75th perc	**826k**	851k	832k	854k	879k	851k	887k	671k	779k	**452k**	732k	695k	779k	695k
Search time	**0.26**	0.31	0.26	0.31	0.27	0.31	0.27	**0.19**	0.27	0.21	0.29	0.21	0.28	0.22
Total time	**3.31**	4.37	3.54	4.71	3.98	4.58	4.17	**4.69**	12.98	6.19	15.18	6.12	13.45	7.40
# constr	**1491**	1465	1487	1466	1484	1466	1481	**1403**	1306	1353	1280	1311	1298	1306
Constr time	**86.25**	89.11	91.29	89.77	97.09	90.50	95.85	**123.70**	242.81	142.17	279.62	196.13	273.71	209.13
Constr oom	**40**	63	43	63	43	63	46	51	**37**	46	38	66	**37**	60
Constr oot	**136**	139	137	138	140	138	140	**213**	324	268	349	290	332	301
Perfect h	**310**	248	308	247	306	248	300	**310**	228	283	221	308	224	287

Table 6.16.: Comparison of different pruning mechanisms: pruning via throwing away states (*original*) and pruning via abstracting (*abstraction*), full pruning (FP), pruning only irrelevant states (IP), pruning only unreachable states (UP), and no pruning (NOP). All algorithms use shrink strategy B. Values aggregated and highlighted for each merge strategy individually.

important. For the remaining parts of this study, we continue using *original* FP for all variants we consider.

6.8. Overview of the State-of-the-Art

In this section, we summarize the previous parts of our study and give an overview of the different merge strategies we considered so far, including a few new combinations. In particular, Table 6.17 shows results for the following merge strategies:

- CGGL/RL: the vanilla linear merge strategies.
- DFP1/DFP2: DFP1 corresponds to the previous notion of DFP, i.e. the tie-breaking configuration PC/RL/NTO, whereas DFP2 is the best variant of DFP with tie-breaking configuration PA/L/NTO.
- sbMIASM: the best configuration with tie-breaking PA/L/OTN.
- MIASM1/MIASM2/MIASM3: the original MIASM strategy with different fallback merge strategies DFP1, DFP2, and sbMIASM as above. MIASM1 corresponds to the previous notion of MIASM and the other two variants have not been tested previously.
- SCC: the best variant ordering SCCs in increasing size and using DFP2 as a secondary merge strategy.
- *symm*: symmetry-enhanced variants of all above merge strategies (except for SCC, for which no direct integration is possible), including the new combinations with sbMIASM and different tie-breaking variants of DFP and MIASM.

While we already discussed most of the results shown in the table, it serves as a point of reference for future work aiming at improving merge-and-shrink heuristics, and also for the comparison against other abstraction heuristics in the final part of our study. We still discuss several observations, mainly regarding new combinations of merge strategies. In the appendix, Table A.11 on page 181 also shows the results for F and G which we do not discuss further here.

Using MIASM with DFP2 instead of DFP1 improves performance compared to DFP2 ($+14$ coverage), however the improvement is much smaller compared to the improvement obtained with MIASM1 over DFP1 ($+27$). Generally, the performance of MIASM with different variants of DFP as fallback achieves similar performance. Combining MIASM with sbMIASM (MIASM3) is not better than only using sbMIASM (-7 coverage), although MIASM3 can compute the heuristic for more tasks and is faster to compute. Among all merge strategies, the SCC variant using DFP1 as secondary merge strategy is still the state-of-the-art merge strategy.

Comparing the plain merge strategies with their symmetry-enhanced variants, we observe that using DFP2, coverage decreases compared to plain DFP2 (-7), while it increases with the previous configuration of DFP, i.e. DFP1. For the combination with MIASM, we have the opposite behavior: while with the previous variant MIASM1, using symmetries is not beneficial (-2 coverage), it increases coverage by 2 if using MIASM2. Furthermore, also for the merge strategy sbMIASM, using symmetries is beneficial, even though the combination is most expensive to

	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3	SCC
Coverage	712	728	746	760	755	773	774	748	**780**
Exp 50th perc	4437	1148	1754	2141	1245	421	1300	**333**	1754
Exp 75th perc	846k	690k	394k	**255k**	326k	370k	283k	547k	440k
Search time	0.26	0.18	0.16	0.17	0.15	**0.12**	0.13	0.16	0.15
Total time	3.28	2.30	2.40	2.05	3.46	2.63	2.42	3.40	**2.05**
# constr	1441	**1503**	1491	1489	1403	1470	1471	1427	1490
Constr time	64.62	**39.28**	49.14	43.99	162.54	63.12	61.25	127.14	44.16
Constr oom	95	**36**	42	49	51	104	104	105	40
Constr oot	131	128	134	129	213	93	**92**	135	137
Perfect h	257	265	268	251	310	**324**	315	309	310
Linear tree	100.00	100.00	87.79	**8.87**	9.98	73.47	25.97	26.98	60.87
					symm				
	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3	
Coverage	741	743	752	753	757	771	**775**	753	
Exp 50th perc	12k	12k	12k	12k	**1686**	4593	12k	**1686**	
Exp 75th perc	886k	847k	589k	589k	**450k**	706k	**450k**	536k	
Search time	0.27	0.26	0.26	0.26	**0.19**	0.21	0.21	0.21	
Total time	5.54	**4.65**	5.42	5.42	5.26	5.63	5.27	5.93	
# constr	1468	**1504**	1496	1497	1418	1472	1468	1429	
Constr time	103.79	**91.20**	102.46	103.52	189.50	103.16	101.13	158.13	
Constr oom	72	**36**	44	44	46	102	110	102	
Constr oot	127	127	127	126	203	93	**89**	136	
Perfect h	274	266	264	264	**308**	306	306	**308**	
Linear tree	19.96	16.56	18.98	18.97	**2.40**	12.36	5.72	5.46	

Table 6.17.: Overview of state-of-the-art merge-and-shrink configurations using shrink strategy B. DFP1: PC/RL/NTO, DFP2: PA/L/NTO, sbMIASM: PA/RL/OTN, MIASM1: with fallback DFP1, MIASM2: with fallback DFP2, MIASM3: with fallback sbMIASM, SCC: order SCCs in increasing size, with secondary strategy DFP1.

compute in terms of construction time: coverage increases by 2 and the already low percentage of linear merge strategies of sbMIASM becomes even lower, achieving the best result of 2.4% among all merge strategies considered here. Finally, also for the combination of MIASM with sbMIASM (MIASM3), using symmetries is beneficial ($+5$ coverage), however coverage is still below that of plain sbMIASM (-2).

Concerning the best non-symmetry-enhanced merge strategy SCC-DFP1, as explained earlier, we cannot use it directly within the symmetry-enhancing framework, since the SCC framework requires simple merge strategies for deciding on the merge order of a subset of factors. In future work, we would like to restrict the computation of factored symmetries to a subset of factors, thus allowing their use within the SCC framework.

To conclude this part of the study, we compare the state-of-the-art of merge-and-shrink to the state before the addition of generalized label reduction, and thus before the contributions made in this thesis. The best coverage of any of the linear merge strategies with previous label reduction is 702 (cf. Table 6.3), compared to the best coverage of 780 of our merge strategy SCC-DFP, which is an increase of 78, a difference to be considered very large given that planning tasks of the benchmarks tend to scale exponentially in difficulty. As we have seen throughout our study, this large increase of coverage is due to several techniques: first, the addition of generalized label reduction increased the efficiency of the computation of merge-and-shrink heuristics. Secondly, it allowed for conceptual efficiency improvements in the implementation of merge-and-shrink within Fast Downward. Finally, using new non-linear merge strategies and improving existing merge strategies through using the symmetry-enhancing or the SCC framework pushed the performance of merge-and-shrink heuristics even further.

6.9. Comparison to State-of-the-Art Planners

In this final part of our study, we compare the best merge-and-shrink strategies against other state-of-the-art planning techniques. In particular, we compare against a single PDB heuristic that, starting with all goal variables, includes causally relevant variables to the already selected variables to be part of the pattern until reaching a size limit of 1 million abstract states in the PDB. Furthermore, we include results for iPDB, using a time limit of 900s as suggested by Scherrer, Pommerening, and Wehrle (2015). For both PDB-based techniques, we use the implementation within Fast Downward, which we describe in our paper (Sievers et al., 2012). To also evaluate Cartesian abstractions, the third class of abstractions besides PDBs and merge-and-shrink abstractions, we use a state-of-the-art configuration, called $h^{\text{SCP}}_{\text{hybrid-opt}}$, that computes diverse saturated cost partitioning heuristics (Seipp et al., 2017b; Seipp, 2017) over the combination of PDBs computed by the hill climbing of iPDB, systematic PDBs (Pommerening et al., 2013), and Cartesian abstractions (Seipp & Helmert, 2013) of the goal and landmark task decompositions (Seipp & Helmert, 2014).

We also compare against the winner of the last IPC, SymBA$^*_2$ (Torralba, Alcázar, Borrajo, Kissmann, & Edelkamp, 2014; Torralba et al., 2016), a planner based on symbolic search. It performs several symbolic bidirectional A* searches on different search spaces, starting in the original state space. Whenever search is deemed to become too hard, the planner switches to a previously started bidirectional symbolic search in an abstract state space or starts a new

	RL	DFP2	sbM	M2	SCC	PDB	iPDB	$h^{\text{SCP}}_{\text{hybrid-opt}}$			
Coverage	728	760	755	774	780	697	815	**1043**			
Exp 50th perc	679	1300	808	738	425	5196	1947	**216**			
Exp 75th perc	690k	255k	285k	255k	394k	963k	187k	**59k**			
Search time	0.15	0.15	0.11	0.11	0.13	0.27	0.14	**0.10**			
Total time	2.22	1.96	3.16	2.23	1.96	**0.99**	1.94	223.30			
	h2										
	RL	DFP2	sbM	M2	SCC	PDB	iPDB	$h^{\text{SCP}}_{\text{hybrid-opt}}$	SPM&S	SymBA$^*_1$	SymBA$^*_2$
Coverage	784	822	800	702	817	732	833	**1078**	933	1013	1011
Total time	1.25	1.18	1.96	1.95	1.21	0.84	1.26	222.93	1.99	**0.48**	0.48

Table 6.18.: Comparison of state-of-the-art merge-and-shrink configurations as in Table 6.17 and competitors: single PDB heuristic with 1 million states (PDB), iPDB with $900s$ time limit, $h^{\text{SCP}}_{\text{hybrid-opt}}$, and the IPC 2014 planners SPM&S, SymBA$^*_1$, and SymBA$^*_2$. Results obtained without (upper block) and with (lower block) using the h^2 mutex preprocessor.

such search, using the current frontiers of the original state space search as perimeters to derive (symbolic) abstraction heuristics. SymBA$^*_2$ uses merge-and-shrink abstractions with B and $N = 10000$ and some linear variable order (BDDs can only represent linear merge-and-shrink heuristics), and three different PDBs obtained through selecting variables in different (linear) orders. The variant SymBA$^*_1$ is the same planner but does not use merge-and-shrink abstractions.

Another planner that also participated in IPC 2014 is SPM&S (Torralba et al., 2013; Torralba, Alcázar, López, et al., 2014), a planner using explicit A* search but with symbolic (abstraction) heuristics, enhanced through using a perimeter. In particular, to derive symbolic merge-and-shrink heuristics, the planner performs a symbolic backward search, interleaving expanding further sets of states during the search with relaxing the current frontiers through using merge-and-shrink abstractions (using the same setting as in SymBA$^*_2$ above). Similarly, it generates symbolic PDB heuristics using four different (linear) variable orders.

All three of the planners are mostly about leveraging the strengths of symbolic search rather than using good strategies to compute informed abstraction heuristics.[13] Quantifying the impact of symbolic (perimeter) merge-and-shrink abstractions within these planners is additionally hard because they also use PDBs. Since all three planners also use the h^2 mutex preprocessor of Alcázar and Torralba (2015), we also provide results for our merge-and-shrink variants as well as PDB, iPDB, and $h^{\text{SCP}}_{\text{hybrid-opt}}$ using planning tasks preprocessed with that preprocessor.

Consider Table 6.18. In the upper part, it shows coverage, expansions, search time, and total time for all planners not using the h^2 mutex preprocessor. We see that single merge-and-shrink abstraction heuristics are stronger than single PDB heuristics, however the canonical PDB heuristic over pattern collections (iPDB) already solves more tasks than any variant of merge-and-shrink. Using saturated cost partitionings over diverse Cartesian abstractions and PDBs ($h^{\text{SCP}}_{\text{hybrid-opt}}$) outperforms the other variants by a large margin. The number of expansions and search time correlate with the results for coverage: only for total time, we observe that $h^{\text{SCP}}_{\text{hybrid-opt}}$

[13] Personal communication of Torralba.

is more expensive to compute (it spends $200s$ to find good saturated cost partitionings), and that PDBs are fastest to use (a simple table lookup) compared to the other approaches.

Consider now the results in the lower part. (Since the symbolic search based planners do not report a number of expansions or search time in a comparable manner to the explicit search planners, we omit these attributes.) We observe that all merge-and-shrink variants profit from using the h^2 mutex preprocessor, with the exception of MIASM, which uses information on mutexes to seed its computation of variable partitionings and thus suffers from a reduced amount of left-over mutexes in the SAS$^+$ representation. Furthermore, some of the merge-and-shrink strategies profit more than others. For example, DFP2 solves 62 tasks more and becomes the best performer, while the previously best variant SCC-DFP1 only increases coverage by 37. iPDB also profits from mutexes, but the gap between merge-and-shrink and iPDB is reduced from 35 down to 11 due to using the h^2 mutex preprocessor. The planners involving symbolic search or symbolic abstraction heuristics, shown in the last three columns, perform better than the explicit merge-and-shrink and PDB-based heuristics, but their coverage is still below that of $h^{\text{SCP}}_{\text{hybrid-opt}}$.

While these results clearly show that single merge-and-shrink heuristics cannot compete with state-of-the-art cost partitioning techniques, we think that it is possible that using cost partitioning could also improve the performance of several merge-and-shrink abstractions. A difficulty that merge-and-shrink abstractions face is that they cannot easily be refined like Cartesian abstractions, which makes it harder to guide their computation towards producing abstractions that are refined in certain regions of the state space. To achieve this for merge-and-shrink abstractions, one solution would be to have shrink strategies perform more selective shrink transformations, possibly seeded by additional parameters to guide them towards achieving a certain goal. An alternative way of tailoring merge-and-shrink abstractions towards being fine-granulated in specific regions of the state space is to use partial and non-orthogonal abstractions, i.e. computing merge-and-shrink for (overlapping) subsets of variables of planning tasks. Then a suitably chosen cost partitioning could take care of admissibly combining the abstractions.

7. Conclusion

Merge-and-shrink abstractions were already an established approach for optimally solving classical planning tasks previous to the contributions made in this thesis. In particular, they were among the few approaches able to compute perfect heuristics in polynomial time for several planning domains. This was achieved by using shrinking based on bisimulation and fully label-reduced transition systems. However, the previous theory of label reduction had several limitations, with the most important consequence being that all practical merge-and-shrink algorithms used linear merge strategies. By introducing generalized label reduction, we removed these restrictions: it now is a purely semantic transformation that is safe to be applied at every intermediate step of the merge-and-shrink computation. Furthermore, it is more powerful, allowing the reduction of more labels, while at the same time being much simpler to understand and reason about. Finally, generalized label reduction laid the foundation for practical merge-and-shrink algorithms using non-linear merge strategies. In our experimental study, we confirmed the importance of (generalized) label reduction and also showed that the first non-linear merge strategy we devised outperforms all previous linear merge-and-shrink heuristics.

Our second theoretical contribution is the formal description of the entire merge-and-shrink framework in terms of transformations of (factored) transition systems. Besides the name-giving merging and shrinking, we showed that also pruning and label reductions (based on generalized label reduction) can be described in terms of the transformation framework. We studied all four types of transformations in terms of desirable formal properties of transformations and showed that heuristics induced by transformations of certain properties are admissible and consistent, or even perfect. Finally, we also described how to efficiently implement the merge-and-shrink framework based on generalized label reduction, using memory-efficient representations of label groups and their transitions.

Our final theoretical contribution concerns the expressive power of merge-and-shrink. We proved that non-linear factored mappings are strictly more powerful than linear ones by showing that there exist problem families that can be represented compactly with general factored mappings but not with linear ones. We also gave a precise bound that quantifies the necessary blowup incurred by conversions from general factored mappings to linear ones.

On the side of transformation strategies, we also made several contributions. Firstly, we transferred the notion of structural symmetries to factored transition systems, introducing factored symmetries. We showed how factored symmetries interact with the merge-and-shrink transformations and proved that under certain conditions, shrinking based on factored symmetries is captured by bisimulation-based shrinking and thus an exact transformation. Based on these results, we devised a framework to enhance existing merge strategies by preferably merging symmetric factors, thus increasing the amount of exact shrinking opportunities based on bisimulation. Our experimental results confirmed that such enhanced merge strategies are more efficiently computable and also result in more informed merge-and-shrink heuristics. Further-

more, we showed that the resulting heuristics can be combined with symmetry-based pruning to further increase coverage compared to both techniques used alone.

Secondly, we performed an experimental analysis of existing merge strategies by comparing them against all merge strategies on small planning tasks and large sets of randomly sampled merge strategies on all planning benchmarks, revealing an untapped potential of many existing merge strategies. Based on these observations, we further investigated the simple score-based merge strategy DFP and found that it is highly susceptible to tie-breaking. As an alternative to the successful but complicated precomputed merge strategy MIASM, we presented a simple score-based variant whose performance comes close to that of DFP, although being very expensive to compute. Since precomputed and simple score-based merge strategies both have advantages and disadvantages, we devised a framework for creating hybrid merge strategies that combine the best of both worlds. This framework is based on the strongly connected components (SCCs) of the causal graph. As a precomputation step, it partitions the atomic factors according to the SCCs and only then uses the given merge strategy for merging the atomic factors within the partitions and finally the resulting products. Our experiments confirmed that SCC-based merge strategies indeed improve over the original merge strategies, and that certain combinations of SCC and DFP set a new high water mark for the performance of merge-and-shrink heuristics.

On the practical sides of contributions, we performed a large experimental study, showing the evolution of merge-and-shrink heuristics from before the addition of generalized label reduction until the most recent state-of-the-art techniques. Besides evaluating the above-mentioned techniques, we also confirmed experimentally that our optimized implementation indeed improves the efficiency of computing merge-and-shrink abstractions significantly. All of our practical contributions led to improving merge-and-shrink heuristics further, yielding a large increase in coverage compared to the previous state of the art. Finally, we also provided a comparison against planners making use of symbolic search or symbolic abstraction heuristics, and against other planners based on explicit search and abstraction heuristics, including the state-of-the-art planner using saturated cost partitionings over several diverse abstractions such as PDBs and Cartesian abstractions. While even the best merge-and-shrink heuristics cannot compete with the best planners, we think that there is still further room for improvements, which we discuss next.

7.1. Future Work

Based on our study which suggests an untapped potential of existing merge strategies, one obvious direction for future work is to further investigate merge strategies. More precisely, we already suggested to further improve the performance of symmetry-enhanced merge strategies by considering different merge orders when merging according to factored symmetries. Additionally, limiting each computation of symmetries separately could allow searching for symmetries in more merge-and-shrink iterations. More generally, we think that better combinations of existing merge strategies are possible, such as combining factored symmetries with MIASM or other precomputed merge strategies, and the combination of the frameworks for using factored symmetries and SCCs. Finally, the SCC framework could be extended to support searching for good variable partitionings in general instead of only using SCCs. Such a framework would then

also cover MIASM, allowing an easier combination of MIASM with other merge strategies.

While exact label reductions are obviously preferable to non-exact ones, recent work suggests that requiring labels to have the same costs often represents an obstacle to reducing many labels (Fan et al., 2017). Fan et al. use delta cost partitionings to leverage more label reductions, but an alternative could be to use non-exact label reductions by ignoring label costs. Also, independently of non-unit-cost domains, using non-exact label reductions could help in planning domains with many labels and few opportunities for exact label reductions. Similarly to how the parameter N controls the size of transition systems, a parameter for the maximum amount of transitions could trigger inexact label reductions that reduce labels even if they are not combinable to reduce the number of distinct transitions.

Our comparison to other planners showed that symbolic search, even with linear merge-and-shrink abstractions, is very powerful, and a natural question to ask is how to use non-linear merge strategies to devise symbolic merge-and-shrink abstractions. A promising alternative to BDDs (which are polynomially equivalent to linear factored mappings) are sentential decision diagrams (Darwiche, 2011), which also replace variable orders by variable trees, however using a different generalization than non-linear factored mappings compared to linear ones. More generally, our results showing that non-linear factored mappings are more powerful than linear ones also indicate that it may be worth questioning the ubiquity of BDD representations for symbolic search in automated planning and other areas.

Finally, the comparison against the state-of-the-art planner which uses sophisticated ways of computing (saturated) cost partitionings also suggests that using costs partitionings over several abstractions results in much stronger heuristics than using single heuristics. While in theory there is no need to use several merge-and-shrink abstractions and combine them additively because single merge-and-shrink abstractions can encode arbitrary abstractions, with the advent of general cost partitionings, this has changed. Using partial and non-orthogonal merge-and-shrink abstractions, i.e. several abstractions computed for overlapping subsets of the variables, could help in devising diverse abstractions that can then be combined by cost partitioning.

Bibliography

Alcázar, V. & Torralba, Á. (2015). A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In R. Brafman, C. Domshlak, P. Haslum, & S. Zilberstein (Eds.), *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)* (pp. 2–6). AAAI Press.

Bäckström, C. & Jonsson, P. (2012). Abstracting Abstraction in Search with Applications in Planning. In G. Brewka, T. Eiter, & S. A. McIlraith (Eds.), *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)* (pp. 446–456). AAAI Press.

Bäckström, C. & Jonsson, P. (2013). Bridging the Gap Between Refinement and Heuristics in Abstraction. In F. Rossi (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)* (pp. 2261–2267). AAAI Press.

Bäckström, C. & Klein, I. (1991). Planning in polynomial time: the SAS-PUBS class. *Computational Intelligence*, *7*(3), 181–197.

Bäckström, C. & Nebel, B. (1995). Complexity Results for SAS^+ Planning. *Computational Intelligence*, *11*(4), 625–655.

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1993). Algebraic Decision Diagrams and Their Applications. In M. R. Lightner & J. A. G. Jess (Eds.), *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)* (pp. 188–191).

Bonet, B. & Castillo, J. (2011). A Complete Algorithm for Generating Landmarks. In F. Bacchus, C. Domshlak, S. Edelkamp, & M. Helmert (Eds.), *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)* (pp. 315–318). AAAI Press.

Bonet, B. & Geffner, H. (2001). Planning as Heuristic Search. *Artificial Intelligence*, *129*(1), 5–33.

Bonet, B. & Helmert, M. (2010). Strengthening Landmark Heuristics via Hitting Sets. In H. Coelho, R. Studer, & M. Wooldridge (Eds.), *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)* (pp. 329–334). IOS Press.

Bryant, R. E. (1985). Symbolic Manipulation of Boolean Functions Using a Graphical Representation. In H. Ofek & L. A. O'Neill (Eds.), *Proceedings of the 22nd ACM/IEEE Conference on Design Automation (DAC 1985)* (pp. 688–694).

Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Cattell, K., Dinneen, M. J., & Fellows, M. R. (1996). A simple linear-time algorithm for finding path-decompositions of small width. *Information Processing Letters*, *57*, 197–203.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. The MIT Press.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press.

Culberson, J. C. & Schaeffer, J. (1998). Pattern Databases. *Computational Intelligence*, *14*(3), 318–334.

Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In T. Walsh (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)* (pp. 819–826). AAAI Press.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*, 269–271.

Domshlak, C., Hoffmann, J., & Katz, M. (2015). Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, *221*, 73–114.

Domshlak, C., Katz, M., & Shleyfman, A. (2012). Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In L. McCluskey, B. Williams, J. R. Silva, & B. Bonet (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Domshlak, C., Katz, M., & Shleyfman, A. (2015). *Symmetry Breaking in Deterministic Planning as Forward Search: Orbit Space Search Algorithm* (tech. rep. No. IS/IE-2015-03). Technion, Haifa.

Dräger, K., Finkbeiner, B., & Podelski, A. (2006). Directed Model Checking with Distance-Preserving Abstractions. In A. Valmari (Ed.), *Proceedings of the 13th International SPIN Workshop (SPIN 2006)* (Vol. 3925, pp. 19–34). Lecture Notes in Computer Science. Springer-Verlag.

Dräger, K., Finkbeiner, B., & Podelski, A. (2009). Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer*, *11*(1), 27–37.

Edelkamp, S. (2001). Planning with Pattern Databases. In A. Cesta & D. Borrajo (Eds.), *Proceedings of the Sixth European Conference on Planning (ECP 2001)* (pp. 84–90). AAAI Press.

Edelkamp, S. (2006). Automated Creation of Pattern Database Search Heuristics. In *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)* (pp. 35–50).

Edelkamp, S. & Hoffmann, J. (2004). *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition* (tech. rep. No. 195). Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Edelkamp, S., Kissmann, P., & Torralba, Á. (2012). Symbolic A* Search with Pattern Databases and the Merge-and-Shrink Abstraction. In L. De Raedt, C. Bessiere, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, & P. Lucas (Eds.), *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)* (pp. 306–311). IOS Press.

Fan, G., Müller, M., & Holte, R. (2014). Non-Linear Merging Strategies for Merge-and-Shrink Based on Variable Interactions. In S. Edelkamp & R. Barták (Eds.), *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)* (pp. 53–61). AAAI Press.

Fan, G., Müller, M., & Holte, R. (2017). Additive Merge-and-Shrink Heuristics for Diverse Action Costs. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)* (pp. 4287–4293). AAAI Press.

Felner, A., Korf, R. E., Meshulam, R., & Holte, R. (2007). Compressed Pattern Databases. *Journal of Artificial Intelligence Research*, *30*, 213–247.

Felner, A., Korf, R., & Hanan, S. (2004). Additive Pattern Database Heuristics. *Journal of Artificial Intelligence Research*, *22*, 279–318.

Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., & Zhang, Z. (2011). Inconsistent Heuristics in Theory and Practice. *Artificial Intelligence*, *175*, 1570–1603.

Felner, A., Zahavi, U., Schaeffer, J., & Holte, R. C. (2005). Dual Lookups in Pattern Databases. In L. P. Kaelbling & A. Saffiotti (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)* (pp. 103–108). Professional Book Center.

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, *2*, 189–208.

Fox, M. & Long, D. (1999). The Detection and Exploitation of Symmetry in Planning Problems. In T. Dean (Ed.), *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)* (pp. 956–961). Morgan Kaufmann.

Fox, M. & Long, D. (2002). Extending the Exploitation of Symmetries in Planning. In M. Ghallab, J. Hertzberg, & P. Traverso (Eds.), *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)* (pp. 83–91). AAAI Press.

Fox, M. & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Gnad, D. & Hoffmann, J. (2015). Beating LM-Cut with h$^{\max}$ (Sometimes): Fork-Decoupled State Space Search. In R. Brafman, C. Domshlak, P. Haslum, & S. Zilberstein (Eds.),

Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015) (pp. 88–96). AAAI Press.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Haslum, P. (2009). $h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation from $h^{\max}$ to h^m. In A. Gerevini, A. Howe, A. Cesta, & I. Refanidis (Eds.), *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)* (pp. 354–357). AAAI Press.

Haslum, P., Botea, A., Helmert, M., Bonet, B., & Koenig, S. (2007). Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)* (pp. 1007–1012). AAAI Press.

Haslum, P. & Geffner, H. (2000). Admissible Heuristics for Optimal Planning. In S. Chien, S. Kambhampati, & C. A. Knoblock (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)* (pp. 140–149). AAAI Press.

Helmert, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M. (2009). Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, *173*, 503–535.

Helmert, M. & Domshlak, C. (2009). Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In A. Gerevini, A. Howe, A. Cesta, & I. Refanidis (Eds.), *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)* (pp. 162–169). AAAI Press.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible Abstraction Heuristics for Optimal Sequential Planning. In M. Boddy, M. Fox, & S. Thiébaux (Eds.), *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)* (pp. 176–183). AAAI Press.

Helmert, M., Haslum, P., & Hoffmann, J. (2008). Explicit-State Abstraction: A New Method for Generating Heuristic Functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)* (pp. 1547–1550). AAAI Press.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM*, *61*(3), 16:1–63.

Helmert, M., Röger, G., & Sievers, S. (2015). On the Expressive Power of Non-Linear Merge-and-Shrink Representations. In R. Brafman, C. Domshlak, P. Haslum, & S. Zilberstein

(Eds.), *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)* (pp. 106–114). AAAI Press.

Hoffmann, J. (2005). Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *Journal of Artificial Intelligence Research*, *24*, 685–758.

Hoffmann, J., Kissmann, P., & Torralba, Á. (2014). "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In T. Schaub, G. Friedrich, & B. O'Sullivan (Eds.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)* (pp. 441–446). IOS Press.

Hoffmann, J. & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Holte, R. C., Felner, A., Newton, J., Meshulam, R., & Furcy, D. (2006). Maximizing over Multiple Pattern Databases Speeds up Heuristic Search. *Artificial Intelligence*, *170*(16–17), 1123–1136.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation* (2nd). Addison-Wesley.

Horton, R. E. (1945). Erosional development of streams and their drainage basins; hydrophysical approach to quantitative morphology. *Bulletin of the Geological Society of America*, *56*, 275–370.

Jonsson, P. & Bäckström, C. (1998). State-Variable Planning under Structural Restrictions: Algorithms and Complexity. *Artificial Intelligence*, *100*(1–2), 125–176.

Junttila, T. & Kaski, P. (2007). Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)* (pp. 135–149). SIAM.

Katz, M. & Domshlak, C. (2007). Structural Patterns of Tractable Sequentially-Optimal Planning. In M. Boddy, M. Fox, & S. Thiébaux (Eds.), *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)* (pp. 200–207). AAAI Press.

Katz, M. & Domshlak, C. (2008). Optimal Additive Composition of Abstraction-based Admissible Heuristics. In J. Rintanen, B. Nebel, J. C. Beck, & E. Hansen (Eds.), *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)* (pp. 174–181). AAAI Press.

Katz, M. & Domshlak, C. (2010a). Implicit Abstraction Heuristics. *Journal of Artificial Intelligence Research*, *39*, 51–126.

Katz, M. & Domshlak, C. (2010b). Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, *174*(12–13), 767–798.

Katz, M., Hoffmann, J., & Helmert, M. (2012). How to Relax a Bisimulation? In L. McCluskey, B. Williams, J. R. Silva, & B. Bonet (Eds.), *Proceedings of the Twenty-Second Interna-*

tional Conference on Automated Planning and Scheduling (ICAPS 2012) (pp. 101–109). AAAI Press.

Keyder, E., Hoffmann, J., & Haslum, P. (2014). Improving Delete Relaxation Heuristics Through Explicitly Represented Conjunctions. *Journal of Artificial Intelligence Research*, *50*, 487–533.

Keyder, E., Richter, S., & Helmert, M. (2010). Sound and Complete Landmarks for And/Or Graphs. In H. Coelho, R. Studer, & M. Wooldridge (Eds.), *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)* (pp. 335–340). IOS Press.

Kinnersley, N. G. (1992). The vertex separation number of a graph equals its path-width. *Information Processing Letters*, *42*, 345–350.

Kissmann, P. & Edelkamp, S. (2011). Improving Cost-Optimal Domain-Independent Symbolic Planning. In W. Burgard & D. Roth (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)* (pp. 992–997). AAAI Press.

Knoblock, C. A. (1994). Automatically Generating Abstractions for Planning. *Artificial Intelligence*, *68*(2), 243–302.

Kocsis, L. & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)* (Vol. 4212, pp. 282–293). Lecture Notes in Computer Science. Springer-Verlag.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., ... Wilkins, D. (1998). *PDDL – The Planning Domain Definition Language – Version 1.2* (tech. rep. No. CVC TR-98-003/DCS TR-1165). Yale Center for Computational Vision and Control. Yale University.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Milner, R. (1990). Operational and Algebraic Semantics of Concurrent Processes. In J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics* (pp. 1201–1242). Elsevier and MIT Press.

Nissim, R., Hoffmann, J., & Helmert, M. (2011). Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning. In T. Walsh (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)* (pp. 1983–1990). AAAI Press.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pochter, N., Zohar, A., & Rosenschein, J. S. (2011). Exploiting Problem Symmetries in State-Based Planners. In W. Burgard & D. Roth (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)* (pp. 1004–1009). AAAI Press.

Pommerening, F., Helmert, M., & Bonet, B. (2017). Abstraction Heuristics, Cost Partitioning and Network Flows. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)* (pp. 228–232). AAAI Press.

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015). From Non-Negative to General Operator Cost Partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)* (pp. 3335–3341). AAAI Press.

Pommerening, F., Röger, G., & Helmert, M. (2013). Getting the Most Out of Pattern Databases for Classical Planning. In F. Rossi (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)* (pp. 2357–2364). AAAI Press.

Pommerening, F., Röger, G., Helmert, M., & Bonet, B. (2014). LP-based Heuristics for Cost-optimal Planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)* (pp. 226–234). AAAI Press.

Richter, S. & Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, *39*, 127–177.

Riddle, P., Douglas, J., Barley, M., & Franco, S. (2016). Improving Performance by Reformulating PDDL into a Bagged Representation. In *ICAPS 2016 Workshop on Heuristics and Search for Domain-independent Planning* (pp. 28–36).

Rintanen, J. (2012). Planning as Satisfiability: Heuristics. *Artificial Intelligence*, *193*, 45–86.

Rintanen, J., Heljanko, K., & Niemelä, I. (2006). Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, *170*(12–13), 1031–1080.

Scherrer, S., Pommerening, F., & Wehrle, M. (2015). Improved Pattern Selection for PDB Heuristics in Classical Planning (Extended Abstract). In L. Lelis & R. Stern (Eds.), *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)* (pp. 216–217). AAAI Press.

Seipp, J. (2017). Better Orders for Saturated Cost Partitioning in Optimal Classical Planning. In A. Fukunaga & A. Kishimoto (Eds.), *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)* (pp. 149–153). AAAI Press.

Seipp, J. & Helmert, M. (2013). Counterexample-guided Cartesian Abstraction Refinement. In D. Borrajo, S. Kambhampati, A. Oddi, & S. Fratini (Eds.), *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)* (pp. 347–351). AAAI Press.

Seipp, J. & Helmert, M. (2014). Diverse and Additive Cartesian Abstraction Heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)* (pp. 289–297). AAAI Press.

Seipp, J., Keller, T., & Helmert, M. (2017a). A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)* (pp. 259–268). AAAI Press.

Seipp, J., Keller, T., & Helmert, M. (2017b). Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)* (pp. 3651–3657). AAAI Press.

Seipp, J., Pommerening, F., & Helmert, M. (2015). New Optimization Functions for Potential Heuristics. In R. Brafman, C. Domshlak, P. Haslum, & S. Zilberstein (Eds.), *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)* (pp. 193–201). AAAI Press.

Seipp, J., Pommerening, F., Sievers, S., & Helmert, M. (2017). downward-lab 2.0. https://doi.org/10.5281/zenodo.399255. doi:10.5281/zenodo.399255

Seipp, J., Sievers, S., Helmert, M., & Hutter, F. (2015). Automatic Configuration of Sequential Planning Portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)* (pp. 3364–3370). AAAI Press.

Shleyfman, A., Katz, M., Helmert, M., Sievers, S., & Wehrle, M. (2015). Heuristics and Symmetries in Classical Planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)* (pp. 3371–3377). AAAI Press.

Sieling, D. & Wegener, I. (1993). NC-Algorithms for Operations on Binary Decision Diagrams. *Parallel Processing Letters*, *3*(1), 3–12.

Sievers, S., Ortlieb, M., & Helmert, M. (2012). Efficient Implementation of Pattern Database Heuristics for Classical Planning. In D. Borrajo, A. Felner, R. Korf, M. Likhachev, C. Linares López, W. Ruml, & N. Sturtevant (Eds.), *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)* (pp. 105–111). AAAI Press.

Sievers, S., Röger, G., Wehrle, M., & Katz, M. (2017). Structural Symmetries of the Lifted Representation of Classical Planning Tasks. In *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning* (pp. 67–74).

Sievers, S., Wehrle, M., & Helmert, M. (2014). Generalized Label Reduction for Merge-and-Shrink Heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)* (pp. 2358–2366). AAAI Press.

Sievers, S., Wehrle, M., & Helmert, M. (2016). An Analysis of Merge Strategies for Merge-and-Shrink Heuristics. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)* (pp. 294–298). AAAI Press.

Sievers, S., Wehrle, M., Helmert, M., & Katz, M. (2015). An Empirical Case Study on Symmetry Handling in Cost-Optimal Planning as Heuristic Search. In S. Hölldobler, M. Krötzsch, R. Peñaloza-Nyssen, & S. Rudolph (Eds.), *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)* (Vol. 9324, pp. 151–165). Lecture Notes in Artificial Intelligence. Springer-Verlag.

Sievers, S., Wehrle, M., Helmert, M., & Katz, M. (2017). Strengthening Canonical Pattern Databases with Structural Symmetries. In A. Fukunaga & A. Kishimoto (Eds.), *Proceed-*

ings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017) (pp. 91–99). AAAI Press.

Sievers, S., Wehrle, M., Helmert, M., Shleyfman, A., & Katz, M. (2015). Factored Symmetries for Merge-and-Shrink Abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)* (pp. 3378–3385). AAAI Press.

Torralba, Á. (2015). *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning* (Doctoral dissertation, Universidad Carlos III de Madrid).

Torralba, Á., Alcázar, V., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). SymBA*: A Symbolic Bidirectional A* Planner. In *Eighth International Planning Competition (IPC-8): planner abstracts* (pp. 105–109).

Torralba, Á., Alcázar, V., Kissmann, P., & Edelkamp, S. (2017). Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, *242*, 52–79.

Torralba, Á., Alcázar, V., López, C. L., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). SPM&S Planner: Symbolic Perimeter Merge-and-Shrink. In *Eighth International Planning Competition (IPC-8): planner abstracts* (pp. 101–104).

Torralba, Á. & Hoffmann, J. (2015). Simulation-Based Admissible Dominance Pruning. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)* (pp. 1689–1695). AAAI Press.

Torralba, Á. & Kissmann, P. (2015). Focusing on What Really Matters: Irrelevance Pruning in Merge-and-Shrink. In L. Lelis & R. Stern (Eds.), *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)* (pp. 122–130). AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2013). Symbolic Merge-and-Shrink for Cost-Optimal Planning. In F. Rossi (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)* (pp. 2394–2400). AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2016). Abstraction Heuristics for Symbolic Bidirectional Search. In S. Kambhampati (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)* (pp. 3272–3278). AAAI Press.

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

Wehrle, M. & Helmert, M. (2012). About Partial Order Reduction in Planning and Computer Aided Verification. In L. McCluskey, B. Williams, J. R. Silva, & B. Bonet (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)* (pp. 297–305). AAAI Press.

Wehrle, M. & Helmert, M. (2014). Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)* (pp. 323–331). AAAI Press.

Wehrle, M., Helmert, M., Alkhazraji, Y., & Mattmüller, R. (2013). The Relative Pruning Power of Strong Stubborn Sets and Expansion Core. In D. Borrajo, S. Kambhampati, A. Oddi, & S. Fratini (Eds.), *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)* (pp. 251–259). AAAI Press.

Wolfe, J. & Russell, S. J. (2011). Bounded Intention Planning. In T. Walsh (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)* (pp. 2039–2045). AAAI Press.

Yang, F., Culberson, J., Holte, R., Zahavi, U., & Felner, A. (2008). A General Theory of Additive State Space Abstractions. *Journal of Artificial Intelligence Research*, *32*, 631–662.

A. Appendix: Additional Tables

Section 6.3: Factored Symmetries

	CGGL		RL		DFP		MIASM	
	orig	*symm*	*orig*	*symm*	*orig*	*symm*	*orig*	*symm*
Coverage	502	**504**	500	**502**	**522**	505	**562**	559
Exp 50th perc	**0**	**0**	825	**305**	**443**	1522	**0**	275
Exp 75th perc	741k	**659k**	776k	**499k**	726k	**625k**	**376k**	457k
Search time	0.12	**0.12**	0.13	**0.12**	**0.15**	0.16	**0.10**	0.14
Total time	**2.00**	6.60	**1.94**	6.27	**2.17**	6.45	**5.84**	12.57
# constr	979	**1000**	1055	**1074**	**1095**	1057	**1121**	1115
Constr time	**87.21**	114.97	**85.24**	105.24	**64.15**	95.97	**54.46**	82.20
Constr oom	566	**544**	479	**458**	**434**	470	**399**	405
Constr oot	**122**	123	**133**	135	**138**	140	**147**	**147**
Perfect h	**280**	274	242	**245**	**262**	252	**321**	280
Linear tree	99.80	**32.50**	99.81	**32.96**	86.21	**31.50**	58.52	**16.59**

Table A.1.: Pairwise comparison of plain merge strategies (*orig*) to their symmetry-enhanced counterparts (*symm*), using shrink strategy F.

	CGGL		RL		DFP		MIASM	
	orig	*symm*	*orig*	*symm*	*orig*	*symm*	*orig*	*symm*
Coverage	**565**	515	**552**	517	**552**	519	**547**	524
Exp 50th perc	**2617**	**2617**	**4177**	**4177**	**4177**	**4177**	**2617**	**2617**
Exp 75th perc	**969k**	**969k**	**969k**	**969k**	**969k**	**969k**	**969k**	**969k**
Search time	0.18	**0.18**	0.21	**0.21**	0.22	**0.21**	0.18	**0.17**
Total time	**1.44**	2.52	**1.51**	2.62	**1.60**	2.73	**7.00**	8.50
# constr	**1069**	1030	**1206**	1091	**1184**	1069	**1097**	1026
Constr time	**73.93**	82.76	**99.07**	111.98	**98.03**	110.70	**57.75**	69.44
Constr oom	**461**	503	**310**	425	**335**	447	**416**	487
Constr oot	137	**134**	**151**	**151**	**148**	151	**154**	**154**
Perfect h	**76**	61	**71**	61	**71**	61	**71**	61
Linear tree	99.81	**23.40**	99.83	**24.66**	89.27	**25.54**	55.42	**10.04**

Table A.2.: Pairwise comparison of plain merge strategies (*orig*) to their symmetry-enhanced counterparts (*symm*), using shrink strategy G.

Section 6.4: Optimized Implementation

	base	opt	diff	+	-	base	opt	diff	+	-
	CGGL					RL				
Coverage	502	520	**18**	18	0	500	507	**7**	8	1
Search time	0.15	0.15	0.00	44	154	0.17	0.17	0.00	52	179
Total time	2.35	2.24	**−0.11**	221	201	2.44	2.42	**−0.03**	179	243
Exp 75th perc	**741k**	741k	0	67	73	801k	**801k**	0	96	75
# constr	979	1095	**116**	119	3	1055	1144	**89**	92	3
	DFP					MIASM				
Coverage	522	576	**54**	76	22	562	586	**24**	35	11
Search time	0.15	0.08	**−0.07**	151	74	0.14	0.13	**−0.01**	117	110
Total time	2.36	2.15	**−0.21**	124	310	6.84	2.14	**−4.70**	485	51
Exp 75th perc	726k	**300k**	**−426**	144	65	1106k	**792k**	**−314**	84	99
# constr	1095	1067	−28	111	139	1121	1240	**119**	131	12
	symm-CGGL					*symm*-RL				
Coverage	504	522	**18**	25	7	502	512	**10**	15	5
Search time	0.14	0.15	0.01	35	163	0.16	0.16	0.01	56	176
Total time	5.90	4.89	**−1.01**	346	127	6.39	5.08	**−1.31**	362	112
Exp 75th perc	**745k**	827k	82	73	65	522k	**507k**	**−15**	85	101
# constr	1000	1123	**123**	136	13	1074	1166	**92**	104	12
	symm-DFP					*symm*-MIASM				
Coverage	505	514	**9**	14	5	559	549	−10	15	25
Search time	0.17	0.18	0.00	107	120	0.14	0.14	**−0.00**	93	145
Total time	6.68	5.52	**−1.15**	409	67	12.06	5.43	**−6.64**	446	73
Exp 75th perc	**724k**	918k	194	86	90	828k	**772k**	**−56**	106	95
# constr	1057	1152	**95**	99	4	1115	1183	**68**	95	27
	CGGL					RL				
Coverage	565	579	**14**	14	0	552	554	**2**	2	0
Search time	0.17	0.17	0.01	25	280	0.23	0.23	0.01	12	323
Total time	1.98	1.67	**−0.30**	362	119	1.80	1.61	**−0.19**	305	158
# constr	1069	1084	**15**	20	5	1206	1209	**3**	10	7
	DFP					MIASM				
Coverage	552	554	**2**	2	0	547	548	**1**	10	9
Search time	0.23	0.23	0.00	122	213	0.21	0.21	0.00	101	224
Total time	1.89	1.67	**−0.22**	346	121	7.55	2.07	**−5.47**	465	56
# constr	1184	1213	**29**	35	6	1097	1149	**52**	65	13
	symm-CGGL					*symm*-RL				
Coverage	515	531	**16**	16	0	517	520	**3**	7	4
Search time	0.19	0.19	0.00	22	285	0.19	0.19	0.01	29	281
Total time	2.67	1.73	**−0.93**	414	66	2.39	1.57	**−0.81**	411	66
# constr	1030	1032	**2**	25	23	1091	1052	−39	24	63
	symm-DFP					*symm*-MIASM				
Coverage	519	525	**6**	9	3	524	524	0	6	6
Search time	0.19	0.19	0.00	60	253	0.18	0.18	0.00	79	234
Total time	2.50	1.62	**−0.87**	427	54	8.65	2.26	**−6.39**	459	43
# constr	1069	1059	−10	32	42	1026	1048	**22**	39	17

Table A.3.: Comparison of the non-optimized implementation against the optimized one, for shrink strategies F (upper half) and G (lower half).

Section 6.6.2: DFP Tie-breaking

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	539	537	540	531	528	533	517	515	516	**575**
Exp 50th perc	703	703	709	**3**	**3**	12	545	545	541	1012
Exp 75th perc	394k	394k	394k	346k	346k	**346k**	406k	406k	406k	436k
Search time	0.12	0.12	0.12	0.09	**0.09**	0.09	0.12	0.12	0.12	0.13
Total time	1.50	1.49	1.50	1.35	1.35	1.35	1.45	1.44	1.45	**1.18**
# constr	1196	**1197**	1189	1095	1095	1099	1068	1068	1075	1190
Constr time	39.20	39.13	39.05	41.36	41.57	41.44	49.28	49.02	49.46	**37.18**
Constr oom	**331**	**331**	338	437	438	434	453	453	445	338
Constr oot	140	139	140	135	**134**	**134**	146	146	147	139
Perfect h	263	263	263	**276**	274	273	255	255	255	238
Linear tree	92.89	92.90	93.36	84.57	84.57	84.71	92.23	92.23	92.09	**15.46**
	Prefer atomic (PA)									
Coverage	549	573	558	572	569	**596**	549	572	565	
Exp 50th perc	766	939	1370	**63**	660	421	523	717	503	
Exp 75th perc	604k	624k	572k	**375k**	812k	486k	459k	539k	634k	
Search time	0.12	0.15	0.15	**0.11**	0.13	0.12	0.12	0.14	0.14	
Total time	1.05	0.96	1.12	1.20	**0.94**	1.13	1.26	1.00	1.09	
# constr	1187	1245	1226	1191	1233	**1283**	1155	1198	1197	
Constr time	27.47	22.88	24.55	32.09	23.24	24.17	32.31	**22.84**	25.75	
Constr oom	340	288	304	345	302	**252**	375	333	331	
Constr oot	140	134	137	**131**	132	132	137	136	139	
Perfect h	248	226	236	**259**	227	237	235	216	234	
Linear tree	10.78	10.28	10.44	10.92	10.54	10.21	10.56	**10.18**	10.19	
	Prefer composite (PC)									
Coverage	554	554	554	**605**	**605**	**605**	560	560	560	557
Exp 50th perc	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**
Exp 75th perc	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**
Search time	0.22	0.22	0.22	**0.22**	0.22	0.22	0.22	0.22	0.22	0.22
Total time	1.60	1.59	1.59	1.48	1.48	**1.48**	1.65	1.65	1.65	1.52
# constr	1213	1214	1214	**1249**	**1249**	**1249**	1209	1210	1209	1210
Constr time	78.22	77.89	77.70	75.08	75.11	75.40	81.89	81.15	81.30	**74.63**
Constr oom	322	322	322	**288**	**288**	**288**	324	326	326	327
Constr oot	132	131	131	**130**	**130**	**130**	134	131	132	**130**
Perfect h	71	71	71	**81**	**81**	**81**	71	71	71	72
Linear tree	33.39	33.36	33.36	34.59	34.59	34.59	34.08	34.05	34.08	**8.76**
	Prefer atomic (PA)									
Coverage	582	553	553	558	**597**	555	557	556	558	
Exp 50th perc	4356	**4288**	4375	4300	**4288**	**4288**	4379	**4288**	**4288**	
Exp 75th perc	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	
Search time	0.23	**0.23**	0.23	0.23	0.23	0.23	0.23	0.23	0.23	
Total time	1.42	**1.29**	1.43	1.48	1.34	1.43	1.47	1.35	1.42	
# constr	1236	1206	1206	1209	**1251**	1207	1206	1208	1211	
Constr time	77.07	74.28	76.01	74.57	**71.96**	74.00	78.97	76.55	78.75	
Constr oom	300	330	330	324	**283**	327	326	328	326	
Constr oot	131	131	131	134	133	133	135	131	**130**	
Perfect h	**81**	74	72	74	72	72	76	74	74	
Linear tree	**5.34**	5.47	5.47	5.87	5.68	5.88	5.64	5.63	5.62	

Table A.4.: Tie-breaking with DFP, for shrink strategies F (upper half) and G (lower half).

Section 6.6.3: sbMIASM Tie-breaking

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	576	569	574	576	573	577	**582**	569	579	572
Exp 50th perc	**0**	**0**	**0**	9	9	**0**	8	8	16	29
Exp 75th perc	317k	313k	307k	279k	279k	279k	**268k**	**268k**	268k	404k
Search time	0.10	**0.10**	0.10	0.11	0.11	0.11	0.11	0.12	0.11	0.11
Total time	3.08	2.99	3.04	3.44	3.36	3.40	3.39	3.31	3.35	**2.83**
# constr	**1063**	1056	1053	1058	1056	1054	1060	1051	1051	1059
Constr time	138.87	**137.21**	143.18	143.88	141.37	144.33	168.65	167.47	170.49	155.30
Constr oom	262	271	274	271	278	281	253	264	**250**	276
Constr oot	342	340	340	338	333	**332**	354	352	366	**332**
Perfect h	322	319	**327**	316	317	321	308	305	313	292
Linear tree	61.71	62.12	61.82	62.19	62.31	62.52	62.08	62.51	62.61	**14.64**
	Prefer atomic (PA)									
Coverage	578	571	575	576	582	572	578	**586**	575	
Exp 50th perc	572	572	572	**35**	572	293	520	293	227	
Exp 75th perc	367k	459k	405k	773k	662k	661k	**255k**	396k	538k	
Search time	0.13	0.13	0.14	0.14	**0.13**	0.14	0.13	0.14	0.14	
Total time	3.00	2.68	2.95	3.05	**2.63**	2.98	3.04	2.66	3.04	
# constr	1073	1083	1073	1079	**1089**	1060	1068	1076	1060	
Constr time	107.21	**100.57**	127.22	119.37	104.31	114.77	127.38	106.81	124.97	
Constr oom	269	**268**	276	276	273	299	285	303	301	
Constr oot	325	316	318	312	305	308	314	**288**	306	
Perfect h	294	289	290	**306**	297	299	289	280	290	
Linear tree	10.72	**10.62**	10.62	11.40	11.39	11.79	10.96	10.97	11.13	
	Prefer composite (PC)									
Coverage	557	558	558	570	**571**	**571**	566	567	567	560
Exp 50th perc	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**
Exp 75th perc	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**
Search time	0.26	0.26	0.26	0.25	0.25	0.25	**0.25**	0.25	0.25	0.25
Total time	1.86	**1.83**	1.85	2.06	2.04	2.03	2.10	2.09	2.10	2.03
# constr	1157	1161	1165	1162	1173	**1175**	1162	**1175**	1167	1154
Constr time	124.29	116.60	117.89	119.01	**106.09**	107.48	132.62	118.26	119.59	107.78
Constr oom	264	267	265	263	262	258	259	**252**	259	283
Constr oot	246	239	237	242	232	234	246	240	241	**230**
Perfect h	71	71	71	**76**	**76**	**76**	72	72	72	71
Linear tree	9.08	9.04	9.01	10.24	10.14	10.13	9.98	9.87	9.94	**5.11**
	Prefer atomic (PA)									
Coverage	564	558	565	566	**567**	**567**	560	560	564	
Exp 50th perc	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	
Exp 75th perc	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	**1129k**	
Search time	**0.25**	0.26	0.26	0.26	0.26	0.25	0.25	0.25	0.25	
Total time	2.16	**1.86**	1.98	1.94	1.86	2.06	2.21	2.03	2.11	
# constr	1163	1154	**1171**	1158	1163	1168	1156	1167	1164	
Constr time	129.03	118.50	118.06	119.69	**105.05**	107.64	133.82	116.80	119.08	
Constr oom	263	278	**262**	273	275	271	274	268	271	
Constr oot	241	235	234	236	229	**228**	237	232	232	
Perfect h	**76**	71	71	71	72	**76**	73	72	74	
Linear tree	1.46	1.47	**1.45**	1.73	1.72	1.71	1.47	1.46	1.46	

Table A.5.: Tie-breaking with sbMIASM, for shrink strategies F (upper half) and G (lower half).

Section 6.6.4: The SCC Framework

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	776	777	**778**	752	753	753	734	734	734	733
Exp 50th perc	**1267**	**1267**	**1267**	2175	2175	2175	10k	10k	16k	9481
Exp 75th perc	**509k**	**509k**	**509k**	749k	749k	749k	701k	701k	701k	1164k
Search time	**0.17**	0.17	0.17	0.22	0.21	0.22	0.26	0.26	0.26	0.26
Total time	**2.30**	2.31	2.31	2.85	2.82	2.86	2.84	2.81	2.83	2.31
# constr	1491	1490	1488	1442	1442	1441	1487	1485	1486	**1494**
Constr time	58.07	58.14	58.47	81.13	80.72	80.68	69.76	69.98	70.36	**54.89**
Constr oom	**40**	42	42	91	93	92	43	45	45	**40**
Constr oot	136	135	137	134	**132**	134	137	137	136	133
Perfect h	**310**	309	307	294	294	291	277	278	278	273
Linear tree	60.83	60.94	60.95	61.17	61.23	61.21	63.35	63.43	63.39	**10.11**
	Prefer atomic (PA)									CGGL
Coverage	751	730	736	**764**	748	744	741	734	741	744
Exp 50th perc	4603	6054	8049	**1994**	6947	3631	7767	14k	12k	2915
Exp 75th perc	693k	1110k	769k	**605k**	785k	677k	856k	933k	816k	920k
Search time	0.23	0.27	0.25	**0.18**	0.23	0.22	0.23	0.27	0.24	0.21
Total time	2.05	1.74	1.97	2.11	**1.71**	1.88	2.27	1.82	1.95	3.10
# constr	1493	1494	1492	**1501**	1498	1494	1491	1490	1490	1441
Constr time	59.09	52.94	54.67	57.17	**51.20**	52.82	58.09	53.21	54.69	79.38
Constr oom	**40**	42	42	**40**	42	44	42	44	44	95
Constr oot	134	131	133	**126**	127	129	134	133	133	131
Perfect h	279	265	270	282	271	275	275	258	268	**300**
Linear tree	6.16	**6.16**	6.17	6.33	6.34	6.36	6.17	6.17	6.17	65.09
	Prefer composite (PC)									
Coverage	**765**	756	750	739	736	734	746	741	742	761
Exp 50th perc	1001	1277	1001	1799	1799	1799	1345	1345	1319	**995**
Exp 75th perc	**459k**	482k	**459k**	487k	487k	487k	562k	562k	562k	552k
Search time	0.19	0.20	0.20	0.21	0.21	0.20	0.20	0.20	0.21	**0.19**
Total time	4.68	4.50	4.61	5.73	5.55	5.71	5.43	5.27	5.31	**4.11**
# constr	1377	1376	1379	1345	1344	1342	1359	1357	1353	**1405**
Constr time	141.08	136.42	142.88	202.01	192.05	200.59	182.08	177.09	184.14	**113.71**
Constr oom	41	**40**	41	42	43	42	44	43	42	44
Constr oot	249	251	247	280	280	283	264	267	272	**218**
Perfect h	**336**	328	328	318	310	310	322	314	323	318
Linear tree	36.38	36.34	36.19	35.84	35.86	35.84	35.91	35.96	36.07	**13.88**
	Prefer atomic (PA)									RL
Coverage	767	**769**	766	766	763	757	757	762	764	762
Exp 50th perc	4977	**1724**	6967	1823	3877	5707	3736	4756	3617	1730
Exp 75th perc	466k	503k	611k	**449k**	478k	546k	893k	540k	508k	927k
Search time	**0.17**	0.17	0.18	0.17	0.18	0.19	0.19	0.18	0.18	0.23
Total time	4.53	3.67	4.20	4.37	3.97	4.39	4.47	3.84	4.22	**2.64**
# constr	1406	1432	1426	1409	1414	1408	1416	1426	1426	**1498**
Constr time	177.54	134.13	145.10	160.37	136.18	151.30	161.56	133.26	145.23	**37.22**
Constr oom	43	41	42	**40**	**40**	43	42	43	42	**40**
Constr oot	218	194	199	218	213	216	209	198	199	**129**
Perfect h	330	321	316	**334**	321	320	325	314	328	301
Linear tree	11.10	10.89	10.94	10.36	10.33	10.37	10.10	**10.03**	**10.03**	53.27

Table A.6.: Tie-breaking with SCC-DFP (top) and SCC-sbMIASM (bottom), and results for SCC-CGGL (top) and SCC-RL (bottom), all using shrink strategy B.

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	538	537	538	548	540	546	523	520	520	**584**
Exp 50th perc	292	292	292	**0**	**0**	8	443	443	439	152
Exp 75th perc	517k	416k	492k	334k	334k	**333k**	419k	396k	397k	363k
Search time	0.12	0.13	0.13	**0.09**	0.09	0.09	0.12	0.11	0.11	0.11
Total time	1.51	1.52	1.51	1.24	1.22	1.22	1.33	1.32	1.33	**1.11**
# constr	1153	1151	1148	1095	1099	1099	1057	1057	1058	**1200**
Constr time	36.19	36.12	36.17	38.87	38.86	38.99	41.48	41.59	41.33	**29.37**
Constr oom	372	374	374	432	431	432	466	464	465	**332**
Constr oot	142	141	142	137	**135**	**135**	143	144	144	**135**
Perfect h	266	263	265	**294**	289	292	271	270	271	265
Linear tree	55.42	55.52	55.23	51.32	51.32	51.41	53.74	53.64	53.69	**12.83**
	Prefer atomic (PA)									CGGL
Coverage	545	568	553	564	576	**592**	538	573	554	537
Exp 50th perc	80	266	79	66	89	70	36	78	72	**0**
Exp 75th perc	383k	564k	399k	352k	460k	459k	466k	**322k**	346k	418k
Search time	0.12	0.13	0.12	0.11	0.12	0.12	0.12	0.12	0.12	**0.10**
Total time	1.00	**0.93**	0.95	1.08	0.96	0.99	1.14	0.97	0.98	1.26
# constr	1182	1223	1202	1158	1225	**1265**	1142	1198	1187	1084
Constr time	31.54	**27.34**	28.79	33.04	27.47	28.75	32.97	27.90	29.80	44.42
Constr oom	347	311	329	378	309	**270**	388	331	343	455
Constr oot	137	133	135	131	132	132	136	137	135	**128**
Perfect h	263	244	259	265	245	255	261	237	255	**306**
Linear tree	7.61	**7.36**	7.49	8.20	7.76	7.51	8.14	7.68	7.75	57.38
	Prefer composite (PC)									
Coverage	563	557	562	**568**	565	563	566	557	558	563
Exp 50th perc	**0**	26	8	41	45	45	25	25	76	188
Exp 75th perc	210k	256k	289k	**204k**	249k	257k	257k	249k	257k	293k
Search time	**0.10**	0.10	0.10	0.10	0.11	0.10	0.10	0.10	0.10	0.11
Total time	2.64	2.58	2.60	2.90	2.86	2.91	2.80	2.76	2.78	**2.57**
# constr	**1080**	1076	1073	1070	1068	1061	1076	1065	1069	1037
Constr time	113.26	**112.39**	112.77	116.13	114.94	115.44	123.04	123.45	122.87	119.39
Constr oom	286	290	290	288	291	303	**273**	283	274	336
Constr oot	299	300	304	308	306	301	317	319	324	**293**
Perfect h	310	302	306	**311**	302	305	304	297	305	297
Linear tree	40.00	40.15	39.89	40.84	40.64	40.90	40.52	40.85	40.69	**16.20**
	Prefer atomic (PA)									RL
Coverage	**579**	562	566	576	577	567	578	576	570	521
Exp 50th perc	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	878
Exp 75th perc	**90k**	173k	105k	145k	223k	169k	147k	147k	103k	416k
Search time	**0.06**	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.12
Total time	1.47	**1.37**	1.42	1.45	1.38	1.42	1.60	1.43	1.53	1.63
# constr	1082	1084	1073	1075	1093	1070	1070	1084	1069	**1122**
Constr time	83.42	75.56	78.09	86.01	75.39	80.62	90.32	81.05	85.09	**32.52**
Constr oom	**285**	296	314	320	293	322	311	323	317	418
Constr oot	298	286	277	272	276	269	286	260	279	**127**
Perfect h	303	284	298	302	292	300	**306**	283	297	261
Linear tree	11.09	11.07	11.09	11.35	11.16	11.21	10.37	**10.24**	10.48	46.08

Table A.7.: Tie-breaking with SCC-DFP (top) and SCC-sbMIASM (bottom), and results for SCC-CGGL (top) and SCC-RL (bottom), all using shrink strategy F.

	RL			L			RND			Random
	NTO	OTN	RND	NTO	OTN	RND	NTO	OTN	RND	
	Prefer composite (PC)									
Coverage	563	563	563	**606**	605	**606**	570	570	570	562
Exp 50th perc	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**
Exp 75th perc	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**
Search time	0.21	0.21	0.21	0.20	0.20	**0.20**	0.21	0.21	0.21	0.21
Total time	1.70	1.69	1.69	1.62	1.63	**1.62**	1.82	1.81	1.81	1.68
# constr	1165	1165	1165	**1196**	1195	1195	1170	1172	1169	1159
Constr time	81.27	80.76	79.98	80.32	80.19	80.43	86.34	86.27	86.26	**74.68**
Constr oom	366	366	366	**338**	**338**	**338**	360	358	361	375
Constr oot	136	136	136	**133**	134	134	137	137	137	**133**
Perfect h	71	71	71	**81**	**81**	**81**	71	71	71	72
Linear tree	27.55	27.55	27.55	28.18	28.12	28.20	26.32	26.28	26.35	**6.47**
	Prefer atomic (PA)									CGGL
Coverage	592	562	563	569	**609**	567	568	567	570	578
Exp 50th perc	**3662**	**3662**	**3662**	**3662**	**3662**	**3662**	**3662**	**3662**	**3662**	**3662**
Exp 75th perc	**680k**	**680k**	**680k**	**680k**	**680k**	**680k**	**680k**	**680k**	**680k**	**680k**
Search time	0.16	**0.16**	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.17
Total time	1.28	**1.17**	1.30	1.33	1.20	1.28	1.34	1.22	1.28	1.40
# constr	1187	1163	1156	1166	**1207**	1163	1160	1161	1167	1085
Constr time	68.91	68.41	68.87	66.38	**64.09**	66.14	68.95	69.06	68.44	67.43
Constr oom	346	371	376	366	**327**	371	371	372	366	461
Constr oot	134	133	135	135	133	133	136	134	134	**121**
Perfect h	**81**	74	72	74	72	72	76	74	74	76
Linear tree	**2.61**	2.67	2.68	2.92	2.82	2.92	2.84	2.84	2.83	54.10
	Prefer composite (PC)									
Coverage	569	569	569	**571**	**571**	**571**	567	567	567	564
Exp 50th perc	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**
Exp 75th perc	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**	**988k**
Search time	0.23	0.24	0.24	0.22	0.23	0.23	**0.22**	0.23	0.23	0.23
Total time	2.04	**2.01**	2.02	2.21	2.20	2.18	2.26	2.24	2.26	2.15
# constr	1123	**1135**	1129	1123	**1135**	1128	1119	1126	1123	1122
Constr time	110.39	105.36	104.04	102.29	99.28	**96.70**	110.40	107.09	108.05	102.72
Constr oom	306	**301**	308	310	305	312	308	308	310	321
Constr oot	238	231	230	234	227	227	240	233	234	**224**
Perfect h	71	71	71	**76**	**76**	**76**	72	72	72	71
Linear tree	9.97	9.87	9.92	10.33	10.22	10.28	10.10	10.04	10.06	**4.46**
	Prefer atomic (PA)									RL
Coverage	567	**569**	567	**569**	568	567	567	567	567	561
Exp 50th perc	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**	**4177**
Exp 75th perc	**802k**	**802k**	**802k**	**802k**	**802k**	**802k**	**802k**	**802k**	**802k**	802k
Search time	**0.20**	0.21	0.21	0.21	0.21	0.20	0.20	0.20	0.20	0.22
Total time	1.90	1.67	1.76	1.70	1.66	1.83	1.95	1.81	1.88	**1.63**
# constr	1121	1133	1127	1120	1130	1125	1120	1126	1125	**1162**
Constr time	88.49	84.54	83.34	78.08	74.49	76.04	89.11	84.71	85.49	**62.73**
Constr oom	311	**306**	313	318	316	321	315	316	315	371
Constr oot	235	228	227	229	221	221	232	225	227	**134**
Perfect h	**76**	71	71	71	73	**76**	73	72	74	71
Linear tree	1.52	**1.50**	1.51	1.79	1.86	1.78	1.52	1.51	1.51	50.09

Table A.8.: Tie-breaking with SCC-DFP (top) and SCC-sbMIASM (bottom), and results for SCC-CGGL (top) and SCC-RL (bottom), all using shrink strategy G.

Section 6.7: Pruning

	original				abstraction			original				abstraction		
	FP	IP	UP	NP	FP	IP	UP	FP	IP	UP	NP	FP	IP	UP
	CGGL							RL						
Coverage	**520**	517	505	462	508	508	502	**507**	**507**	499	481	506	499	501
E 50th perc	**137**	143	**137**	143	143	143	143	**1425**	2090	1525	2090	1436	2090	1436
E 75th perc	**1342k**	**1342k**	**1342k**	**1342k**	**1342k**	**1342k**	**1342k**	1133k	1133k	1133k	1133k	1133k	**1133k**	1133k
Search time	0.23	0.24	0.24	**0.23**	0.24	0.24	0.23	**0.19**	0.19	0.19	0.19	0.19	0.19	0.19
Total time	**2.17**	2.98	2.35	3.82	2.75	3.58	2.91	**2.57**	3.36	2.65	3.57	3.23	3.67	3.29
# constr	**1095**	1085	1067	998	1078	1081	1053	**1144**	1123	1131	1075	**1144**	1122	1127
Constr time	**81.58**	82.80	83.28	94.24	127.83	113.19	126.57	**80.60**	81.78	80.73	82.70	102.79	88.65	101.68
Constr oom	**444**	454	471	540	446	455	473	396	419	410	467	**389**	420	407
Constr oot	**128**	**128**	129	129	143	131	141	127	**125**	126	**125**	134	**125**	133
Perfect h	**286**	285	284	254	284	285	282	242	239	238	225	**243**	239	239
	DFP PA/L/RND							MIASM						
Coverage	**596**	593	589	581	590	585	588	**586**	564	568	524	576	550	568
E 50th perc	**4658**	7118	7007	7786	9056	7786	9056	**2056**	**2056**	**2056**	3071	**2056**	**2056**	**2056**
E 75th perc	**1124k**	1124k	**1124k**	1251k	1194k	1194k	1194k	**780k**	788k	781k	910k	780k	787k	780k
Search time	**0.25**	0.30	0.26	0.32	0.28	0.31	0.28	**0.18**	0.20	0.20	0.23	0.18	0.21	0.20
Total time	**1.86**	2.40	1.99	2.63	2.36	2.76	2.47	**3.18**	4.16	3.57	5.25	3.92	4.95	4.30
# constr	**1283**	1281	1270	1263	**1283**	1280	1261	**1240**	1225	1216	1165	1227	1214	1204
Constr time	54.98	52.65	**52.42**	53.70	65.78	61.59	63.99	**87.53**	90.97	93.06	91.86	115.62	111.62	115.53
Constr oom	252	256	266	275	**251**	255	273	**329**	347	351	407	334	347	359
Constr oot	132	130	131	**129**	133	132	133	98	**95**	100	**95**	106	106	104
Perfect h	**237**	226	230	215	232	223	228	**327**	296	313	272	325	296	313
	sbMIASM PA/RL/OTN							SCC-DFP PC/RL/NTO						
Coverage	580	584	586	566	**587**	570	579	**586**	535	541	461	566	535	541
E 50th perc	**2102**	6315	**2102**	8508	**2102**	6662	**2102**	**786**	857	861	860	843	858	993
E 75th perc	**643k**	1016k	829k	1113k	829k	1016k	829k	786k	691k	**667k**	691k	**667k**	691k	**667k**
Search time	**0.22**	0.28	0.23	0.31	0.23	0.31	0.24	0.15	0.14	0.13	0.15	**0.13**	0.15	0.13
Total time	**1.67**	2.30	1.84	2.68	2.13	2.77	2.34	**1.85**	4.46	1.97	5.74	2.29	4.77	2.50
# constr	1237	1260	**1265**	1257	1263	1261	1261	**1076**	969	984	857	1039	974	968
Constr time	55.97	55.69	**55.44**	56.73	68.21	64.53	66.12	**113.80**	173.18	115.60	180.88	163.50	185.78	169.87
Constr oom	**262**	277	272	279	271	274	275	303	259	348	424	266	**257**	337
Constr oot	131	**130**	**130**	131	133	132	131	**288**	439	335	386	362	436	362
Perfect h	249	236	245	230	**250**	235	243	280	283	278	238	**303**	287	289

Table A.9.: Comparison of different pruning mechanisms: pruning via throwing away states (*original*) and pruning via abstracting (*abstraction*), full pruning (FP), pruning only irrelevant states (IP), pruning only unreachable states (UP), and no pruning (NP). Results for different merge strategies, all using shrink strategy F.

	original				abstraction			original				abstraction		
	FP	IP	UP	NP	FP	IP	UP	FP	IP	UP	NP	FP	IP	UP
	CGGL							RL						
Coverage	**579**	529	578	527	578	527	577	**554**	550	**554**	548	552	550	552
E 50th perc	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**4296**	**4296**	**4296**	**4296**	**4296**	**4296**	**4296**
E 75th perc	**1132k**	**1132k**	**1132k**	**1132k**	**1132k**	**1132k**	**1132k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**
Search time	**0.21**	0.21	0.21	0.21	0.22	0.21	0.22	**0.22**	0.22	0.22	0.22	0.23	0.22	0.23
Total time	**1.40**	1.69	1.43	1.75	1.49	1.72	1.50	**1.53**	1.79	1.56	1.82	1.63	1.81	1.65
# constr	**1084**	1034	1083	1032	1079	1032	1079	1209	1206	**1210**	1204	1205	1206	1204
Constr time	**76.15**	88.64	77.18	88.99	80.26	89.55	79.38	**79.89**	87.67	80.63	88.77	81.95	88.64	81.32
Constr oom	**462**	513	**462**	512	468	513	467	**324**	328	**324**	330	330	328	330
Constr oot	121	**120**	122	123	**120**	122	121	134	133	133	133	**132**	133	133
Perfect h	**76**	71	**76**	71	**76**	71	**76**	**71**	**71**	**71**	**71**	**71**	**71**	**71**
	DFP PA/L/RND							MIASM						
Coverage	**605**	552	603	552	603	552	603	548	544	549	542	548	544	**551**
E 50th perc	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**4232**	**4232**	**4232**	**4232**	**4232**	**4232**	**4232**
E 75th perc	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**1058k**	**806k**	**806k**	**806k**	**806k**	**806k**	**806k**	**806k**
Search time	**0.22**	0.22	0.22	0.22	0.22	0.22	0.22	**0.22**	0.22	0.22	0.22	0.22	0.22	0.22
Total time	**1.57**	1.90	1.60	1.96	1.66	1.93	1.66	**2.12**	2.43	2.17	2.49	2.23	2.46	2.24
# constr	**1249**	1196	1247	1196	1247	1196	1247	1149	1144	1150	1142	1147	1142	**1151**
Constr time	77.28	93.52	**77.09**	93.91	80.03	95.32	80.35	**70.94**	78.56	73.19	79.65	79.29	80.23	78.82
Constr oom	**288**	341	289	339	292	340	294	**422**	428	**422**	429	423	427	423
Constr oot	130	130	131	132	128	131	**126**	96	95	95	96	97	98	**93**
Perfect h	**81**	76	**81**	76	**81**	76	**81**	**71**	**71**	**71**	**71**	**71**	**71**	**71**
	sbMIASM PA/RL/OTN							SCC-DFP PC/RL/NTO						
Coverage	**609**	594	**609**	594	607	594	607	**571**	503	534	502	534	504	534
E 50th perc	**7066**	7068	**7066**	7068	**7066**	7068	**7066**	**2183**	**2183**	**2183**	**2183**	**2183**	**2183**	**2183**
E 75th perc	**999k**	**999k**	**999k**	**999k**	**999k**	**999k**	**999k**	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**
Search time	0.23	**0.23**	0.23	0.23	0.23	0.23	0.23	0.15	**0.14**	0.14	0.14	0.14	0.14	0.14
Total time	1.59	1.70	**1.59**	1.69	1.69	1.78	1.70	**1.26**	2.24	1.70	2.30	1.74	2.27	1.75
# constr	1207	1193	**1208**	1192	1206	1193	1205	**1175**	865	890	856	890	858	889
Constr time	77.13	75.80	**75.10**	75.81	76.57	76.86	77.72	**94.64**	104.29	99.33	105.85	103.22	105.12	108.54
Constr oom	**327**	342	**327**	342	331	342	331	**258**	636	615	645	608	623	627
Constr oot	133	132	132	133	**130**	132	131	234	166	162	166	169	186	**151**
Perfect h	**72**	**72**	**72**	**72**	**72**	**72**	**72**	76	76	**78**	76	**78**	76	**78**

Table A.10.: Comparison of different pruning mechanisms: pruning via throwing away states (*original*) and pruning via abstracting (*abstraction*), full pruning (FP), pruning only irrelevant states (IP), pruning only unreachable states (UP), and no pruning (NP). Results for different merge strategies, all using shrink strategy G.

Section 6.8: State of the Art

Compared to Table 6.17, the merge strategies use different tie-breaking criteria depending on the shrink strategy they are combined with. For F (upper half of Table A.11), we use the following variants:

- CGGL/RL: the vanilla linear merge strategies.
- DFP1/DFP2: DFP1 corresponds to the previous notion of DFP, i.e. the tie-breaking configuration PC/RL/NTO, whereas DFP2 is the best variant of DFP with tie-breaking configuration PA/L/RND.
- sbMIASM: the best configuration with tie-breaking PA/RND/OTN.
- MIASM1/MIASM2/MIASM3: the original MIASM strategy with different fallback merge strategies DFP1, DFP2, and sbMIASM as above. MIASM1 corresponds to the previous notion of MIASM and the other two variants have not been tested previously.
- SCC: the best variant ordering SCCs in topological order and using DFP2 as a secondary merge strategy.
- *symm*: symmetry-enhanced variants of all above merge strategies (except for SCC for which no direct integration is possible), including the new combinations with sbMIASM and different tie-breaking variants of DFP and MIASM.

For G (lower half of Table A.11), we use the following variants:

- CGGL/RL: the vanilla linear merge strategies.
- DFP1/DFP2: DFP1 corresponds to the previous notion of DFP, i.e. the tie-breaking configuration PC/RL/NTO, whereas DFP2 is the best variant of DFP with tie-breaking configuration PC/L/RND.
- sbMIASM: the best configuration with tie-breaking PC/L/RND.
- MIASM1/MIASM2/MIASM3: the original MIASM strategy with different fallback merge strategies DFP1, DFP2, and sbMIASM as above. MIASM1 corresponds to the previous notion of MIASM and the other two variants have not been tested previously.
- SCC: the best variant ordering SCCs in topological order and using DFP2 as a secondary merge strategy.
- *symm*: symmetry-enhanced variants of all above merge strategies (except for SCC for which no direct integration is possible), including the new combinations with sbMIASM and different tie-breaking variants of DFP and MIASM.

	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3	SCC
Coverage	520	507	539	596	586	586	**618**	590	592
Exp 50th perc	**0**	437	151	306	**0**	**0**	**0**	**0**	75
Exp 75th perc	420k	505k	489k	186k	258k	**101k**	145k	258k	356k
Search time	0.09	0.13	0.11	0.10	0.07	**0.06**	0.06	0.06	0.09
Total time	1.18	1.55	1.32	0.89	1.16	1.01	0.88	1.20	**0.81**
# constr	1095	1144	1196	**1283**	1076	1240	1279	1152	1265
Constr time	30.21	30.60	26.23	16.33	109.34	19.58	15.50	86.30	**11.93**
Constr oom	444	396	331	**252**	303	329	293	332	270
Constr oot	128	127	140	132	288	98	**95**	183	132
Perfect h	286	242	263	237	280	**327**	306	308	255
Linear tree	100.00	100.00	92.89	10.21	10.97	75.81	24.78	25.78	**7.51**

symm

	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3
Coverage	522	512	514	553	548	549	563	**564**
Exp 50th perc	**0**	358	476	87	**0**	19	25	9
Exp 75th perc	404k	428k	460k	**160k**	313k	202k	**160k**	230k
Search time	0.10	0.14	0.14	0.10	0.08	0.09	**0.08**	0.08
Total time	4.07	4.63	5.03	**3.00**	3.10	4.40	3.83	4.03
# constr	1123	1166	1152	**1227**	1038	1183	1208	1128
Constr time	60.05	57.78	59.18	**46.52**	134.37	56.03	52.46	110.27
Constr oom	417	375	389	**314**	337	395	368	362
Constr oot	127	126	126	126	292	**89**	91	177
Perfect h	276	238	252	244	280	**292**	282	288
Linear tree	23.86	25.21	28.21	**4.65**	5.59	19.61	8.11	8.95

	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3	SCC
Coverage	579	554	554	605	571	548	594	556	**609**
Exp 50th perc	**3139**	**3139**	**3139**	**3139**	**3139**	**3139**	**3139**	**3139**	**3139**
Exp 75th perc	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**	**715k**
Search time	0.19	0.18	0.18	**0.17**	0.17	0.18	0.18	0.18	0.17
Total time	1.26	1.28	1.32	1.24	1.46	1.77	1.56	1.63	**1.09**
# constr	1084	1209	1213	**1249**	1175	1149	1182	1128	1207
Constr time	54.69	52.26	49.92	49.87	78.43	51.43	53.29	69.03	**48.05**
Constr oom	462	324	322	288	**258**	422	394	382	327
Constr oot	121	134	132	130	234	96	**91**	157	133
Perfect h	76	71	71	**81**	76	71	76	71	72
Linear tree	100.00	100.00	33.39	34.59	10.13	42.73	42.89	27.30	**2.82**

symm

	CGGL	RL	DFP1	DFP2	sbMIASM	MIASM1	MIASM2	MIASM3
Coverage	531	520	525	**535**	**535**	524	532	529
Exp 50th perc	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**	**3919**
Exp 75th perc	**859k**	**859k**	**859k**	**859k**	**859k**	**859k**	**859k**	**859k**
Search time	0.18	0.18	0.18	0.18	**0.18**	0.18	0.18	0.18
Total time	1.55	1.57	1.61	1.47	**1.44**	2.03	1.95	1.87
# constr	1032	1052	1059	**1063**	1052	1048	1053	1037
Constr time	67.70	68.59	69.07	**66.20**	74.84	68.84	69.05	74.80
Constr oom	509	489	482	472	**451**	528	521	527
Constr oot	126	126	126	132	164	**91**	93	103
Perfect h	**63**	**63**	**63**	**63**	**63**	**63**	**63**	**63**
Linear tree	8.53	8.65	4.53	4.23	**2.19**	6.39	6.65	6.36

Table A.11.: Overview of state-of-the-art merge-and-shrink configurations, see the list in the text for details on the used tie-breaking criteria. Used shrink strategies: F (upper half) and G (lower half).