Formally Verified Compositional Algorithms for Factored Transition Systems

Mohammad Ahmad Abdulaziz Ali Mansour

A thesis submitted for the degree of
Doctor of Philosophy of
The Australian National University

June 2018

Declaration

Except where otherwise indicated, this thesis is my own original work.

Mohammad Ahmad Abdulaziz Ali Mansour
27 June 2018

Acknowledgement

Firstly, I would like to thank god who blessed me with the health, will, and ability to do the work in this thesis.

Secondly, I would like to thank Data61 (formerly NICTA) and the Australian National University for the PhD scholarship as well as the multiple travel grants, without which I would have not been able to conduct my research.

Thirdly, I would like to thank my supervisors, Dr. Charles Gretton and Dr. Michael Norrish, who were mentors, as well as friends to me. Our meetings and style of interaction were intellectually very stimulating and enjoyable, and in addition to their rich technical content, our discussions spanned different topics like politics, history, and culture. Dr. Michael Norrish introduced me to the basics of interactive theorem proving and spent countless hours with me in meetings helping me through the different hurdles I faced during my research. Also, without his encouragement and leveraging of his connections, it would have not been possible for me to proceed with my internships. Dr. Charles Gretton's help and guidance made it much easier to get habituated to the literature and the different concepts of artificial intelligence planning. I would also like to thank him for his relentless effort in reading and editing all the drafts I wrote in the course of my PhD studies. I would also like to thank my advisor Prof. Sylvie Thiébaux who helped gauge my bearings research-wise and career-wise, at many points during my PhD program.

In addition to my supervisors, I would like thank different researchers with whom I had very useful interactions related to the work in this thesis. I would like to thank Prof. Brendan McKay who helped me understand many concepts in group theory and graph automorhpism, and formulate and prove theorems that are crucial for significant parts of this thesis. I would also like to thank Dr. Alban Grastien for our many engaging discussions and for his very useful feedback on many parts of my research, as well as for reading many drafts that I wrote through my research. I would also like to thank Dr. Patrik Haslum for his useful feedback and suggestions for my research and on some of my drafts, and for being available to answer many of my questions on the fundamentals and practicalities of AI planning. I would like to thank Prof. Daniel Jackson for his suggestion of the hotel key protocol as an example to try the upper-bounding algorithms on. I would also like to thank Dr. Miquel Ramírez for reading one of the drafts I wrote.

I would also like to thank the researchers who hosted me during my different internships and research visits. Firstly I would like to thank Prof. Lawrence Paulson for hosting me in the University of Cambridge Computer Laboratory, for his patience and generosity with his time and effort, and for our consequent fruitful and joyful collaboration. Secondly, I would like to thank Dr. Bruno Dutertre, Dr. Dejan Jovanovic, and Dr. Natrajan Shankar for hosting me in the Computer Science Laboratory in SRI International and for the very helpful discussions I had with them. From the Computer Science and Artificial Intelligence Laboratory in Massachusetts Institute of Technology, I would like to thank Prof. Brian Williams and Dr. David Wang who hosted me during my visit there.

Last, but not least, I would like to thank my family and my wife for their unlimited help and support in general, and escpecially during my PhD program. I would also like to thank my comrade PhD candidates: Josh, Jan, Fazlul, Karsten, Mazen, and Zuhair, for the many chats we had, technical or otherwise.

Abstract

Artificial Intelligence (AI) planning and model checking are two disciplines that found wide practical applications. It is often the case that a problem in those two fields concerns a transition system whose behaviour can be encoded in a digraph that models the system's state space. However, due to the very large size of state spaces of realistic systems, they are compactly represented as propositionally factored transition systems. These representations have the advantage of being exponentially smaller than the state space of the represented system.

Many problems in AI planning and model checking involve questions about state spaces, which correspond to graph theoretic questions on digraphs modelling the state spaces. However, existing techniques to answer those graph theoretic questions effectively require, in the worst case, constructing the digraph that models the state space, by expanding the propositionally factored representation of the system. This is not practical, if not impossible, in many cases because of the state space size compared to the factored representation.

One common approach that is used to avoid constructing the state space is the compositional approach, where only smaller abstractions of the system at hand are processed and the given problem (e.g. reachability) is solved for them. Then, a solution for the problem on the concrete system is derived from the solutions of the problem on the abstract systems. The motivation of this approach is that, in the worst case, one need only construct the state spaces of the abstractions which can be exponentially smaller than the state space of the concrete system.

We study the application of the compositional approach to two fundamental problems on transition systems: upper-bounding the topological properties (e.g. the largest distance between any two states, i.e. the diameter) of the state space, and computing reachability between states. We provide new compositional algorithms to solve both problems by exploiting different structures of the given system. In addition to the use of an existing abstraction (usually referred to as projection) based on removing state space variables, we develop two new abstractions for use within our compositional algorithms. One of the new abstractions is also based on state variables, while the other is based on assignments to state variables. We theoretically and experimentally show that our new compositional algorithms improve the state-of-the-art in solving both problems, upper-bounding state space topological parameters and reachability. We designed the algorithms as well as formally verified them with the aid of an interactive theorem prover. This is the first application that we are aware of, for such a theorem prover based methodology to the design of new algorithms in either AI planning or model checking.

Contents

Chapter 1

Introduction

The state spaces of problems in fields such as model checking and Artificial Intelligence (AI) planning can be modelled as digraphs, where vertices and edges represent states and transitions, respectively. The digraph modelling the state space is represented as a *propositionally factored transition system* in languages such as STRIPS by Fikes and Nilsson (1971) and SMV by McMillan (1993). This representation has the advantage of being exponentially smaller than an explicit representation of the digraph. That compactness is necessary due to the enormous size of state spaces of realistic systems.

Two well studied problems on digraphs are upper-bounding the digraph's topological properties (e.g. the *diameter*) and testing reachability between vertices. Solving those two problems for digraphs modelling state spaces is crucial for solving problems in AI planning and model checking. However, mainstream algorithms to solve those two problems either assume the digraph is available in an explicit form (i.e. not in a factored form) or they effectively expand the factored representation and construct the digraph modelling its state space. This is impossible in many practical cases, where factored representations of state spaces with 2^{1000} states are not uncommon. That issue is referred to as the state space explosion problem (Clarke et al. (2001)). Furthermore, assuming that constructing the digraph was possible, worst-case run-times of state-of-the-art algorithms to solve those problems are worse than linear in the size of the digraph, which is infeasible.

Different techniques were invented to mitigate the state space explosion. Computing reachability is one notable problem for which many techniques were developed to circumvent the state space explosion. For example, different modifications to path search algorithms were developed. Those techniques can exponentially reduce the path search run-time in many practical examples. Although those methods were developed multiple decades ago, they are still active research topics. Examples of such methods are symmetry breaking (Emerson and Sistla, 1996; Clarke et al., 1996, 1998; Fox and Long, 1999, 2002), partial order reduction (Godefroid, 1990; Gerth et al., 1995; Alur et al., 1997; Chen and Yao, 2009; Xu et al., 2011), and heuristic search (McDermott, 1996; Blum and Furst, 1997; Bonet and Geffner, 2001; Hoffmann and Nebel, 2001). A central drawback with many of these techniques is that they are tailored to specific solution algorithms. For instance, many heuristic search techniques are designed for explicit state space search, but it is not obvious how to combine them with SAT-based planning methods to get a significant performance boost.

Another more general approach to mitigate state space explosion is the *compositional* approach: a solution to the original problem instance is found or approximated via composing solutions to one or many (possibly significantly) smaller derived sub-problems, or "abstractions" (e.g. Clarke et al. (1994); Knoblock (1994); Williams and Nayak (1997); Berezin et al. (1998); Amir and Engelhardt (2003); Kroening (2006); Case et al. (2009); Filiot et al. (2011)). An advantage of this approach is that it is not specific to a certain solution method, i.e. it does not matter how the abstractions' solutions were computed. In this thesis we study the application of this approach to

solve diameter upper bounding and reachability. In particular, we consider computing the topological property (resp. solving the reachability problem) for a set of *minors* of the digraph modelling the state space of the system under consideration. Minors are digraphs derived by vertex contractions as well as edge and vertex deletions. These minors of the state space model the state spaces of abstractions of the given system. Minors' topological properties (resp. reachability witness paths) are then composed into an upper bound on the topological property (resp. a witness reachability path) for the digraph modelling the given system's state space.

In many practical cases the compositional approach is one of the few known feasible approaches to solve problems on the state space of the given factored system. This is because it avoids constructing and performing computations on the large digraph modelling the state space, and only constructs and processes (significantly smaller) minors of it. On the other hand, a drawback of the compositional approach is that it is not always the case that it is usefully applicable to the given transition system. Nonetheless, we identify structures whose presence in the transition system make it possible to compositionally solve the problems. These structures are acyclicity (in "state variable dependencies" and the state space) for compositionally upper-bounding the diameter, and symmetry (between state variables) for compositionally solving the reachability problem. We show that those structures are abundant in systems coming from standard AI planning benchmarks and standard model checking problems. We thus show that our algorithms improve the state-of-the-art in both scenarios of AI planning: proving plan existence and plan non-existence.

In addition to developing new compositional algorithms, we formally verify their validity in the interactive theorem prover HOL4 by Slind and Norrish (2008). To do that we develop a rich formal proof library about transition systems. Many theorems in our formalisation apply to infinite state transition systems and accordingly this library can be used for application well beyond the algorithms that we verify in this work, like algorithms for hybrid systems, for instance. A problem that we faced while developing these compositional algorithms is the sheer difficulty of manually verifying their properties and soundness. Using the proof assistant HOL4, we discovered many mistakes in our work at different stages and even mistakes in the literature, in a process we would describe as "computer-aided algorithm design".

In addition to aiding us develop our algorithms and giving us extra assurance about our results, we believe that the formalisation of AI algorithms in general is of great utility, if those algorithms are to be deployed in safety critical applications or in autonomous exploration of space. There is a twofold justification for requiring formal correctness guarantees. First, if the output of the algorithm is hard to test—e.g. that a plan does not exist—then we require correctness to assure the sound operation of the system at hand. For example, if a chemical plant can be rendered safe in the event of a potentially dangerous subsystem failure, the planner must be able to identify that course of action which averts a catastrophic failure. Secondly, if the output is computationally easy to test—e.g. sound operation is easily guaranteed—it remains a potentially dangerous waste of time and resources to have a computationally expensive planning subsystem that produces unusable plans. This is especially the case when such resources are scarce and one cannot afford to invest in an algorithm without guarantees, like with autonomous exploration of space.

Propositionally Factored Transition Systems In this representation, every state (i.e. vertex) is defined as an assignment to a set of Boolean state variables. For example, consider a hotel that has a number of rooms, each of which opens with one of a set of keys at a given time. In this case, the set of state variables assigned indicating whether a certain room opens with a certain key at the state.

In the factored representation multiple transitions (i.e. edges) are factored into actions that identify which states are connected to which states. For instance, consider an action that, given

the variable indicating that a certain room opens with a key i is true (its precondition), negates that variable and sets another one indicating that the room currently opens with a different key j (its effect). This action factors all the transitions that go from states where the room opens with key i to states where it opens with key j. In addition to the compactness of this representation, without it, we would not be able to pursue the proofs of our results or define concepts like the *sublist diameter* (see Section 3.6), as naturally.

Minors and Abstractions Compositional approaches involve solving given problems for state spaces of "abstractions" of the given system, which are minors of the digraph modelling the given system's state space. Abstractions that we consider are *projection* over a set of state variables (see Section 3.3.1), *snapshotting* over an assignment of a set of state variables (see Section 3.6.2), and *quotienting* (see Section 4.5) over a partition of state variables. In projection, state variables are removed from the factored system. This is equivalent to repetitive vertex contraction operations in the state space. In snapshotting, actions that violate the given assignment (i.e. if it holds, they either cannot execute or change it in their effect) are removed from the factored system. This is equivalent to consecutive edge and vertex deletions in the state space. In quotienting, given a partition of the state variables, all members of an equivalence class are removed, except for one representative from each equivalence class. This is also equivalent to consecutive edge and vertex deletions in the state space.

1.1 Contributions

We start by discussing our results on upper-bounding topological properties of digraphs modelling state spaces. Our first two results are negative results concerning compositional bounding of two prominent topological properties: the diameter (the longest distance between any two states) and the recurrence diameter (the length of the longest simple path). We show that any method that compositionally bounds the (recurrence) diameter using only "state variable dependency" analysis is unsound. We do so by demonstrating that the (recurrence) diameters of minors modelling projections' state spaces cannot be composed, solely based on inter-projection variable dependencies, to upper-bound the (recurrence) diameter of the state space of the system. In other words, a compositional algorithm needs more "information" than just variable dependencies to compositionally upper-bound the (recurrence) diameter.

We define two new topological properties that can be compositionally bounded, which are themselves upper bounds on the diameter and/or the recurrence diameter. Thus, they can be used to compositionally upper-bound the diameter and/or the recurrence diameter. The first property, the *traversal diameter*, can always be compositionally bounded, regardless of the structure of the transition system. However, it can be exponentially looser than the diameter and the recurrence diameter. The second property, the *sublist diameter*, can be compositionally bounded in systems that can be partitioned into sub-systems with acyclic dependency relations among them. It is an upper bound on the diameter and a lower bound on the recurrence diameter. We prove that the compositional bounds that we provide for both properties cannot be improved.

In deriving the above results, we also provide theoretical constructs that we believe are interesting themselves. The first is a novel proof artifact, *the stitching function* ($\wr$), which can be used to merge paths from different projections of a transition system into a path in the concrete system. We also provide digraph constructions that we refer to as "flowers" and "inverted flowers". The main feature of the flower constructions is that they are digraphs that can have arbitrarily many vertices added to them, without changing their diameters, sublist diameters, or longest paths.

We provide a new algorithm to compositionally upper-bound the diameter by using the projections' recurrence diameters. We prove that bounds computed by this algorithm cannot be improved. We also experimentally show that this algorithm significantly outperforms previous algorithms in terms of tightness of computed bounds. We combine this algorithm with a novel approach to exploiting acyclicity in the state space to form a hybrid bounding procedure. We show that this algorithm enables the decomposition of a given system into very small abstractions compared to state-of-the-art algorithms. Also experimentally, compared to existing practical approaches, ours can yield exponentially tighter bounds, which we use to significantly improve the coverage and performance of a propositional satisfiability based planner.

The other problem which we seek to solve compositionally is reachability between states. We provide a novel procedure to exploit symmetries for efficient computation of reachability between states in transition systems. This procedure differs from existing approaches in that we use symmetries to obtain an abstraction of the concrete problem description, which we call the *descriptive quotient*. The reachability computation is then done on the abstraction, and if a path is found in the descriptive quotient, it is used to synthesise a path in the concrete system. We show experimentally that this approach significantly outperforms other approaches to exploit symmetries for finding paths between states in many standard AI planning benchmarks. A drawback with this method, nonetheless, is that it is incomplete.

Our last contribution is that we provide a library of formal proofs for transition systems in the interactive theorem prover HOL4 and we use it to formally verify the algorithms that we developed. Although there are many formal proof libraries on the subject of transition systems, the library that we developed is the first we are aware of that is dedicated for systems in the factored representation.

Availability All our HOL scripts, experimental code and data are available from `https://MohammadAbdulaziz@bitbucket.org/MohammadAbdulaziz/planning.git`.

1.1.1 Publications

During the course of my PhD I wrote as a main contributor, with other contributors, the papers and manuscripts that are listed below:

Mohammad Abdulaziz and Lawrence C Paulson. An Isabelle/HOL formalisation of Green's theorem. In *International Conference on Interactive Theorem Proving*, pages 3–19. Springer, 2016.

Mohammad Abdulaziz, Charles Gretton, and Michael Norrish. Mechanising Theoretical Upper Bounds in Planning. In *Workshop on Knowledge Engineering for Planning and Scheduling*, 2014.

Mohammad Abdulaziz, Charles Gretton, and Michael Norrish. Verified Over-Approximation of the Diameter of Propositionally Factored Transition Systems. In *Interactive Theorem Proving*, pages 1–16. Springer, 2015a.

Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Exploiting symmetries by planning for a descriptive quotient. In *Proc. of the 24th International Joint Conference on Artificial Intelligence, IJCAI*, pages 25–31, 2015b.

Mohammad Abdulaziz, Charles Gretton, and Michael Norrish. A State Space Acyclicity Property for Exponentially Tighter Plan Length Bounds. In *International Conference on Automated Planning and Scheduling (ICAPS)*. In Press, 2017a.

Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Formally Verified Algorithms for Upper Bounding State Space Diameters. In *To appear in the Journal of Automated Reasoning*, 2017b.

1.2 Thesis Structure

This thesis is organised as follows. In Chapter 2 we discuss the basics concepts related to factored transition systems and the action execution semantics in our framework. In Chapter 3 we describe previous work in the literature related to computing upper bounds on topological properties of directed graphs in general as well as directed graphs that model state spaces. We then discuss more formal definitions that are more specific to compositional upper-bounding. We then describe our different compositional algorithms and results to compute upper bounds, and then we end this chapter by concluding remarks regarding our approach for compositional upper-bounding as well as some open questions that we believe are interesting. In Chapter 4 we discuss different approaches in the literature to compute reachability in general directed graphs as well as state spaces, and focus more on the approaches that try to exploit symmetry to enhance the efficiency of the reachability computation. Then we discuss our approach for compositional computation of reachability, where we formally define concepts that are relevant to our algorithm, and we focus on computing reachability within the framework of automated planning. Then we finalise that chapter with remarks summarising the merits of our compositional approach to compute reachability and possible ways of extending it. Lastly, in Chapter 5, we discuss previous work in the literature on formalisation and we focus on results related to formalising transition systems. We then discuss our formalisation of factored transition systems and our compositional algorithms in the interactive theorem prover HOL4, and then finish that chapter by some concluding remarks on our experience in formalising our algorithms.

Chapter 2

Basic Concepts and Notations

Where we do not refer to a specific function, we shall use the symbol f. We formalise functions as sets of key-value pairs ($k \mapsto v$). We use the term *domain* mathematically. We write $\mathcal{D}(f)$ for the domain of f, i.e., $\{k \mid (k \mapsto v) \in f\}$. We write $\mathcal{R}(f)$ for the range of f, i.e., $\{v \mid (k \mapsto v) \in f\}$.

Definition 1 (Digraph)**.** *A digraph whose vertices are labelled by values of type α (α-graph) $\mathcal{G}_\alpha$ is represented by the tuple $\langle V, E, f\rangle$. V is the set of vertices, which we denote as $V(\mathcal{G}_\alpha)$. E is a set of edges, that are distinct ordered pairs of* vertices *from $V(\mathcal{G}_\alpha)$, which we denote as $E(\mathcal{G}_\alpha)$. $f : V(\mathcal{G}_\alpha) \Rightarrow \alpha$ is a labelling function, where a vertex $u_1 \in V(\mathcal{G}_\alpha)$ has the label $f(u_1)$ of type α, which we denote as $\mathcal{G}_\alpha(u_1)$. We note that when we illustrate a digraph in a figure, a vertex u_1 is docketed with the label $f(u_1)$ instead of the vertex name u_1. We also omit self loops. For $u_1 \in V(\mathcal{G}_\alpha)$, the set of children of u_1 is $\mathsf{children}_{\mathcal{G}_\alpha}(u_1) = \{u_2 \mid (u_1, u_2) \in E(\mathcal{G}_\alpha)\}$. If f is injective, we use $l \in \mathcal{G}_\alpha$ to denote that l is a label of some $u_1 \in V(\mathcal{G}_\alpha)$, i.e. $l = f(u_1)$, and we use $\mathsf{children}_{\mathcal{G}_\alpha}(l)$ to refer to the labels of the children of u_1, i.e. $\{\mathcal{G}_\alpha(u_2) \mid u_2 \in \mathsf{children}_{\mathcal{G}_\alpha}(u_1)\}$. For $\mathcal{G}_\alpha \equiv \langle V, E, f\rangle$, the image of a function $g : \alpha \Rightarrow \beta$ on $\mathcal{G}_\alpha$ is the β-graph $g(\!|\mathcal{G}_\alpha|\!) \equiv \langle V, E, g \circ f\rangle$, where $g \circ f$ is the composition of f and g. Effectively, the image operation on a digraph is a relabelling operation.*

We now define factored representations of digraphs, which are mainly used to compactly represent digraphs that model state spaces. In these representations a digraph is propositionally factored, where vertices correspond to "states" and sets of edges are compactly described in terms of "actions". In this formalism edges and paths between vertices correspond to executing actions and action sequences, respectively, between states.[1]

Definition 2 (States and Actions)**.** *A state, x, is a finite map from variables—i.e., state-characterising propositions—to Booleans, i.e. a set of mappings $v \mapsto b$. We write $\mathcal{D}(x)$ to denote $\{v \mid v \mapsto b \in x\}$, the domain of x. For states x_1 and x_2, the union, $x_1 \uplus x_2$, is defined as $\{v \mapsto b \mid$ $\mathsf{if}\ (v \in \mathcal{D}(x_1))\ \mathsf{then}\ b = x_1(v)\ \mathsf{else}\ b = x_2(v)\}$. Note that the state x_1 takes precedence. An action is a pair of finite maps, (p, e), where p represents the* preconditions *and e represents the* effects*. For action $\pi = (p, e)$, $\mathcal{D}(\pi) \equiv \mathcal{D}(p) \cup \mathcal{D}(e)$.*

Definition 3 (Execution)**.** *When an action $\pi\ (= (p, e))$ is executed at state x, it produces a successor state $\mathsf{ex}(x, \pi)$, formally defined as $\mathsf{ex}(x, \pi) = \mathsf{if}\ p \not\subseteq x\ \mathsf{then}\ x\ \mathsf{else}\ e \uplus x$. We lift ex to lists of actions $\overrightarrow{\pi}$ as the second argument. So $\mathsf{ex}(x, \overrightarrow{\pi})$ denotes the state resulting from successively applying each action from $\overrightarrow{\pi}$ in turn, starting at x, which corresponds to a path in the state space.*

[1] This representation is equivalent to representations commonly used in the model checking and AI planning communities (e.g. STRIPS by Fikes and Nilsson (1971) and SMV by McMillan (1993)). Whereas conventional expositions in the planning and model-checking literature would also define initial conditions and goal/safety criteria, here we omit those features from discussion. Our bounds and the different topological properties we consider are independent of those features.

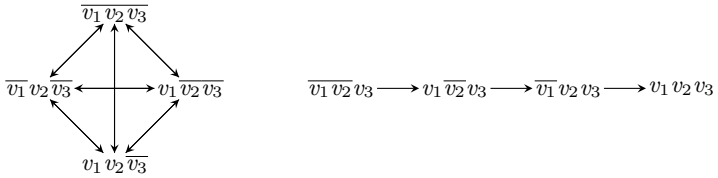


Figure 2.1: The state space of δ in Example 1.

We give examples of states and actions using sets of literals. For example, $\{v_1, \overline{v_2}\}$ is a state where state variables v_1 is (maps to) true, and v_2 is false and its domain is $\{v_1, v_2\}$. $(\{v_1, \overline{v_2}\}, \{v_3\})$ is an action that if executed in a state where v_1 and $\overline{v_2}$ hold, it sets v_3 to true. $\mathcal{D}((\{v_1, \overline{v_2}\}, \{v_3\})) = \{v_1, v_2, v_3\}$.

Definition 4 (Factored System)**.** *For a set of actions δ we write $\mathcal{D}(\delta)$ for the domain of δ, which is the union of the domains of all the actions in δ. The set of valid states, written $\mathbb{U}(\delta)$, is $\{x \mid \mathcal{D}(x) = \mathcal{D}(\delta)\}$. The set of valid action sequences is the Kleene closure of δ, i.e $\delta^* = \{\overrightarrow{\pi} \mid \mathsf{set}(\overrightarrow{\pi}) \subseteq \delta\}$, where $\mathsf{set}(l)$ is the set of members in list l.*

δ is the factored representation of the digraph $\mathcal{G}(\delta) \equiv \langle V, E, f\rangle$, where $V = \mathbb{U}(\delta)$, $E \equiv \{(x, \mathsf{ex}(x, \pi)) \mid x \in \mathbb{U}(\delta), \pi \in \delta\}$, and f is the identity function. We refer to $\mathcal{G}(\delta)$ as the state space of δ. For states x and x', $x \rightsquigarrow x'$ denotes that there is a $\overrightarrow{\pi} \in \delta^*$ such that $\mathsf{ex}(x, \overrightarrow{\pi}) = x'$. Let the connected component for a state x be $\mathcal{S}(\delta, x) = \{y \mid x \rightsquigarrow y \vee y \rightsquigarrow x\}$.

Example 1. *Consider the following factored representation, δ, and the digraph in Figure 2.1 that represents its state space.*

$$\left\{ \begin{array}{l} p_1 = (\{\overline{v_1}, \overline{v_2}, v_3\}, \{v_1\}), p_2 = (\{v_1, \overline{v_2}, v_3\}, \{\overline{v_1}, v_2\}), p_3 = (\{\overline{v_1}, v_2, v_3\}, \{v_1\}), \\ k_1 = (\{\overline{v_3}\}, \{v_1, v_2\}), k_2 = (\{\overline{v_3}\}, \{\overline{v_1}, v_2\}), k_3 = (\{\overline{v_3}\}, \{v_1, \overline{v_2}\}), k_4 = (\{\overline{v_3}\}, \{\overline{v_1}, \overline{v_2}\}) \end{array} \right\}.$$

In the figure different states defined on the variables $\mathcal{D}(\delta) = \{v_1, v_2\, v_3\}$ are shown. Interpreting δ as a transition system, it has two "modes" of operation, where the variable v_3 in the preconditions of actions in δ determines the mode of operation. Each of those modes represents one connected component of the digraph in Figure 2.1, where actions k_1, k_2, k_3, k_4 (which execute successfully if v_3 maps to false) represent edges in the clique component. For instance, action k_1 represents all incoming edges to vertex $\{v_1, v_2\, \overline{v_3}\}$. On the other hand, each one of the actions p_1, p_2, p_3 (which execute successfully if v_3 maps to true) represents an edge in the simple path component.

Chapter 3

Compositional Upper-Bounding of Topological Properties

Two fundamental properties of digraphs are the *diameter* and the length of the longest simple path (the latter also known as the *recurrence diameter* in the verification community). Knowing an upper bound on the diameter or the recurrence diameter of a state space comes up as a necessary ingredient for many practical applications and algorithms in model checking and AI planning. A prominent example is the bounded model-checking algorithm by Biere et al. (1999), whose completeness depends on having an upper bound on the diameter (for checking safety properties or plan existence) or the recurrence diameter (for checking liveness properties). Because of this practical importance a lot of effort has been spent trying to efficiently compute or approximate both properties.

However, and as we pointed out earlier to compute either property, one needs to expand the factored representation δ and construct the digraph modelling its state space, $\mathcal{G}(\delta)$, which is infeasible. This is further compounded by the fact that state-of-the-art algorithms to compute or reasonably approximate the diameter and the longest path, have worse than quadratic (e.g. Yuster (2010); Aingworth et al. (1999)) and exponential (e.g. Kroening and Strichman (2003)) run-times, respectively, in the size of the digraph. In this chapter we study applying the compositional approach to compute upper bounds on both properties for digraphs that model state spaces. In particular, we consider computing the topological property for a set of minors of the state space. Minors' properties are then composed into an upper bound on the property of the original digraph.

State Variable Dependency Informally, a state variable v_1 depends on v_2 iff the assignment of v_2 at some state can possibly affect assignment of v_1 in a current or a future state. This relation lifts to sets of state variables, where a set vs_2 depends on vs_1 iff vs_2 has a variable that depends on some variable in vs_1. Dependency based analysis of model-checking and planning problems has been used for a long time (e.g. Rintanen and Gretton (2013); Baumgartner et al. (2002); Williams and Nayak (1997); Helmert (2006a); Kroening (2006)). A practically important class of projections are ones done on partitions of the state variables whose members are closed under mutual variable dependency, i.e. variable dependencies between different equivalence classes are acyclic. Those partitions capture the abundantly present acyclic dependencies between different modules of real-world systems, such as different circuit components.

3.1 Related Work

Computing the diameter exactly can be done via solving the All Pairs Shortest Path (APSP) problem for the (di)graph at hand. APSP cannot be solved in better than quadratic time (in the number

of vertices of the (di)graph), and existing exact solutions have a run-time close to cubic (e.g. Fredman (1976); Alon et al. (1997); Chan (2010); Yuster (2010)). Furthermore, there is strong evidence that the diameter cannot be computed in time better than quadratic (Roditty and Vassilevska Williams, 2013). This run-time can be very limiting in digraphs arising in practical applications due to their size. Accordingly, in the algorithms community a lot of work has been done on developing methods to approximate the diameter. In a seminal paper, Aingworth et al. (1999) devised an algorithm that computes a $\frac{3}{2}$-approximation of the diameter for digraphs in $O(m\sqrt{n \log n} + n^2 \log n)$ time, where n is the number of vertices and m is the number of edges. Examples of other work studying approximation algorithms for digraph diameters include Roditty and Vassilevska Williams (2013); Chechik et al. (2014); Abboud et al. (2016).

On the other hand, computing the recurrence diameter is an NP-hard problem (Pardalos and Migdalas, 2004), with only known exact solutions in exponential time. Accordingly it is much harder to compute than the diameter, especially, for practical digraphs. Furthermore, the hardness of computing the recurrence diameter in digraphs was reaffirmed by Björklund et al. (2004), where they showed that, in the general case, it is impossible (under plausible assumptions) to get a polynomial approximation of the length of the longest path in polynomial time. Existing polynomial time approximation algorithms for the longest path, like Alon et al. (1995); Björklund and Husfeldt (2003), can only find paths of lengths logarithmic in the length of the longest path. Also, to the best of our knowledge, all approximation techniques studied in the algorithms community are lower bounding techniques.

In the graph theory and the combinatorics communities, diameters of undirected graphs have been extensively studied since 1965, where work like Moon et al. (1965); Erdős et al. (1989); Knyazev (1987) computed upper bounds for different classes of undirected graphs. However, for digraphs, work on upper-bounding diameters started more recently. In 1992 Soares (1992) provided a tight upper bound on the diameter of biregular digraphs. Then starting in 2000, Peter Dankelmann et al. treated different structures of digraphs (e.g. Dankelmann (2005); Dankelmann and Volkmann (2010); Dankelmann and Dorfling (2016)).

The completeness of bounded model-checking (Biere et al., 1999, 2003) and satisfiability (SAT) based planning (Kautz and Selman, 1992) depends on having an upper bound on the diameter or the recurrence diameter (depending on the verification problem at hand), and the tighter that bound the more quickly the algorithm will likely terminate. Because of that, studying the diameter and the recurrence diameter is an active research topic in the verification community (see for example Biere et al. (1999, 2003); Kroening and Strichman (2003); Sheeran et al. (2000); Kroening et al. (2011); Bundala et al. (2012); Clarke et al. (2009)). The most dominant approach to compute the diameter or the recurrence diameter in verification applications is via encodings in SAT (see Biere et al. (1999); Kroening and Strichman (2003)). Most notably, Biere et al. (1999) conjecture that for the question of "whether for a certain digraph, a number N is its diameter", there is not a SAT encoding of size polynomial in N. However, if the question is "whether N is the recurrence diameter", they provide a SAT encoding of size $O(N^2)$, which was improved by Kroening and Strichman (2003) to $O(N \log N)$.

Applying the compositional approach for diameter upper-bounding was pioneered by Baumgartner et al. (2002), in the context of bounded model-checking. They showed that the diameter of the system's state space is upper-bounded by a polynomial in the recurrence diameters of the system's projections on sets of state variables closed under mutual dependency (i.e. a set of contractions of the state space). Also, more recently, Rintanen and Gretton applied a similar approach to upper-bound the diameters for AI planning Rintanen and Gretton (2013) problems, but using the number of the contractions' states instead of the recurrence diameter.

3.2 Results

Our contributions are both theoretical and practical. We first discuss the theoretical contributions. First let $\mathcal{G}_i$ denote a digraph and vs_i denote a set of state variables. Then let $\mathcal{G}_{1..n}$ denote the set of digraphs $\{\mathcal{G}_1, \mathcal{G}_2, .., \mathcal{G}_n\}$ and $vs_{1..n}$ denote the sets of variables $\{vs_1, vs_2, .., vs_n\}$. For some function b, let $b_{1..n}$ denote $\{b(\mathcal{G}_1), b(\mathcal{G}_2), .., b(\mathcal{G}_n)\}$. For some partition $vs_{1..n}$, let $\mathcal{G}_{\text{VS}}$ denote the dependencies between different members of $vs_{1..n}$.

Our first two results are negative. We show that recurrence diameters of projections' state spaces cannot be composed, solely based on dependencies between different sets of variables in $vs_{1..n}$, to upper-bound the recurrence diameter of the state space of the system (Theorem 2). Also, if $vs_{1..n}$ satisfies the additional requirement of having two members with a dependency between them, then the same result applies to the diameter (Theorem 1).

To circumvent these negative results we define two new topological properties that can be compositionally bounded, which are themselves upper bounds on the diameter and the recurrence diameter. Thus they can be used to compositionally upper-bound the diameter and the recurrence diameter. We consider projections induced by two types of partitions of the set of state variables of δ: arbitrary partitions and partitions whose equivalence classes are closed under mutual variable dependency.

The first new topological property, is the *traversal diameter* (td), which is one less than the largest number of vertices that can be traversed by any path in a digraph. td can be seen as a measure of branching in $\mathcal{G}(\delta)$. It is an upper bound on (and can be exponentially larger than) the recurrence diameter, and a lower bound on (and can be exponentially smaller than) the number of states in $\mathcal{G}(\delta)$ (Theorem 5). Its main advantage is that it can be compositionally bounded using projections obtained from an arbitrary partition $vs_{1..n}$. It is upper-bounded by $\prod_{1\leq i\leq n}(td_i+1)-1$, if the contractions $\mathcal{G}_{1..n}$ of $\mathcal{G}(\delta)$ are state spaces of projections of δ on members of $vs_{1..n}$ (Theorem 3). Thus it provides a very flexible tool to compositionally upper-bound both the diameter and recurrence diameter. We also prove a strong statement of tightness for that bound: any function taking as input $td_{1..n}$ and $\mathcal{G}_{\text{VS}}$ whose output can be less than $\prod_{1\leq i\leq n}(td_i+1)-1$, is not a sound bound on td, for any partition. We also show that td can be computed in time that is linear in the size of the digraph (Theorem 6).

Although compositional upper-bounding using the traversal diameter has the advantage of being applicable to any partition, it can be a rather loose upper bound on the diameter. To alleviate that, we define the sublist diameter (ℓ), an upper bound on the diameter and a lower bound on the recurrence diameter. A main feature of it is that it can be exponentially smaller than the recurrence diameter, but it can also be exponentially larger than the diameter (Theorem 10).

We introduce a function $\mathsf{N}_{\mathsf{sum}}$ that takes $\ell_{1..n}$ and $\mathcal{G}_{\text{VS}}$. We show that it computes an upper bound on the sublist diameter that is a polynomial in $\ell_{1..n}$, if members of $vs_{1..n}$ are closed under mutual variable dependency (Lemma 3). This is significantly better than using $td_{1..n}$ to upper-bound the diameter since ℓ can be exponentially smaller than td and because the polynomials produced by $\mathsf{N}_{\mathsf{sum}}$ do not necessarily have all possible product terms, as $\prod_{1\leq i\leq n}(td_i+1)-1$ does. Compared to similar earlier approaches in Baumgartner et al. (2002); Rintanen and Gretton (2013), this is a substantial improvement in two ways. Firstly, we prove that any function taking the same input as $\mathsf{N}_{\mathsf{sum}}$, and whose output can be less that computed by $\mathsf{N}_{\mathsf{sum}}$ is not a sound upper bound on the diameter, i.e. $\mathsf{N}_{\mathsf{sum}}$ is a lower bound on all the previous approaches (Theorem 9). Furthermore, we experimentally verify that $\mathsf{N}_{\mathsf{sum}}$ computes significantly tighter bounds than previous approaches. Secondly, we use the contractions' sublist diameters which can be exponentially tighter than functions used previously (namely, the recurrence diameter and the state space size). However, we show that computing the sublist diameter is NP-hard (Theorem 12). The relation

between the different topological properties and bounding functions can be found in Figure 3.1.

Our proof of Lemma 3 is the most technical. It is constructive, and it depends on the fact that a sequence of transitions in δ (i.e. edges in $\mathcal{G}(\delta)$) is equivalent to different sequences of transitions in the different projections (i.e. edges in $\mathcal{G}_{1..n}$) that are interleaved. We show that for each transition sequence from a projection, removing redundant transitions makes it shorter than the sublist diameter of the projection. We then recombine all shortened transition sequences with a novel proof artifact, *the stitching function* ($\wr$), and show that the recombined sequence both begins and ends in the same vertices as the original one, and is shorter than the bound produced by $\mathsf{N}_{\mathsf{sum}}$. Although they had different purposes from ours, in some sense this is a generalisation of the theory developed in Knoblock (1994) and in Brafman and Domshlak (2003).

Another interesting proof is that of Theorem 10. It relies on the fact that, unlike the diameter or the recurrence diameter, the sublist diameter depends on the factored representation δ rather than the underlying state space $\mathcal{G}(\delta)$. We show that it can be minimised through factoring more edges into fewer actions.

Another point to highlight is in our tightness proofs. They depend on digraph constructions that we refer to as "flowers" and "inverted flowers" (see Figure 3.6a and Figure 3.10 for an example). The main feature of those constructions is that they are digraphs that can have arbitrarily many vertices added to them, without changing their diameters, sublist diameters, or longest paths.

We combine exploitation of acyclicity in variable dependency constituted by Theorem 8, with a novel approach to exploiting acyclicity in the state space $\mathcal{G}(\delta)$, to form a hybrid bounding procedure. This algorithm enables the decomposition of a given system to abstract sub-systems that are much smaller than what is attainable using state-of-the-art algorithms, which all depend only on acyclicity in variable dependency. This provides the potential for exponentially easier computation of upper bounds. Also experimentally, compared to existing practical approaches, ours can yield exponentially tighter bounds.

This procedure is based on first finding a partition of the states in $\mathcal{G}(\delta)$ that induces an acyclic "quotient" contraction of $\mathcal{G}(\delta)$. Then, we show that the diameter of $\mathcal{G}(\delta)$ is upper-bounded by the weightiest path in the acyclic quotient of $\mathcal{G}(\delta)$. The weight of a path is the number of edges in it added to the cost of every vertex it traverses. The cost of visiting a vertex in that quotient digraph is the diameter of the subgraph of $\mathcal{G}(\delta)$ induced by the states in that quotient vertex. However, to avoid computing the acyclic quotient by constructing $\mathcal{G}(\delta)$ and the possible state space explosion, we get it through projecting δ on a subset of its state variables. That way, the acyclic quotient is the state space of the projection of δ. Also, to avoid the possible state space explosion in computing the subgraphs induced by different quotient vertices explicitly, we compute them in the factored form by snapshotting δ on different states of the projection.

We experimentally show that this algorithm significantly outperforms—both in the tightness of the bounds obtained and in the quality of decomposition—existing approaches. Using the upper bounds computed by that algorithm as horizons for a SAT-based planner, we prove the unsolvability of problems that cannot be proven using the state-of-the-art state space search planners and model-checkers. One notable example is the problem of model-checking the safety of the hotel key protocol. We also significantly improve the coverage of the SAT-based planner Madagascar MP by using upper bounds rather than ramp-up strategies Rintanen (2012).

3.3 Compositional Upper-Bounding: Negative Results

Topological properties that we consider upper-bounding are the diameter and the recurrence diameter. The diameter is the length of the longest shortest path between any two valid states. This can be formally defined as follows for a propositionally factored system.

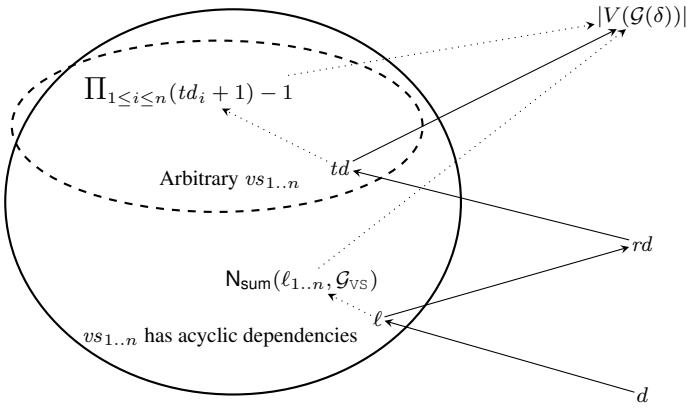


Figure 3.1: A diagram that represents the upper-bounding relations between different topological properties. An arrow from f_1 to f_2 denotes that we showed $f_1(\delta) \leq f_2(\delta)$, for any δ, and f_2 is a "tight" bound on f_1. A solid arrow denotes that we showed an exponential separation between f_1 and f_2. Functions in the dotted ellipse can be compositionally bounded by projections induced by arbitrary partitions of state variables, while those in the solid ellipse need the partitions' members to be closed under mutual dependency.

Definition 5 (Diameter)**.** *Let $|l|$ be the length of a list l. The diameter is defined as follows.*

$$d(\delta) = \max_{x\in\mathbb{U}(\delta)} \max_{\overrightarrow{\pi}\in\delta^*} \min\{|\overrightarrow{\pi}'| \mid \overrightarrow{\pi}' \in \delta^* \wedge \mathsf{ex}(x,\overrightarrow{\pi}) = \mathsf{ex}(x,\overrightarrow{\pi}')\}$$

We would like to note that instead of quantifying over pairs of states, we quantify over states and paths going outside of those states. This is to avoid the complexities arising from the situation when two states are not connected.

The recurrence diameter is the length of the longest simple path in the digraph modelling the state space. This can be formally defined as follows.

Definition 6 (Recurrence Diameter)**.** *Let* $\mathsf{distinct}(x,\overrightarrow{\pi})$ *denote that all states traversed by executing $\overrightarrow{\pi}$ at x are distinct states.*

$$rd(\delta) = \max_{x\in\mathbb{U}(\delta)} \max_{\overrightarrow{\pi}\in\delta^*} \max\{|\overrightarrow{\pi}| \mid \mathsf{distinct}(x,\overrightarrow{\pi})\}$$

If there is a valid action sequence between any two states, then there is a valid action sequence between them that is no longer than the diameter. Also, it should be clear that the recurrence diameter is an upper bound on the diameter.

3.3.1 Projection and Variable Dependency

A key concept for compositional reasoning about factored representations of digraphs is *projection*. Projection of a transition system is equivalent to a sequence of vertex contractions in the digraph modelling the state space.

Definition 7 (Projection)**.** *Projecting an object (a state x, an action π, a sequence of actions $\overrightarrow{\pi}$ or a factored representation δ) on a set of variables vs restricts the domain of the object or the components of composite objects to vs. Projection is denoted as $x\downharpoonright_{vs}$, $\pi\downharpoonright_{vs}$, $\overrightarrow{\pi}\downharpoonright_{vs}$ and $\delta\downharpoonright_{vs}$ for a state, action, action sequence and factored representation, respectively. In the case of action sequences, an action with no effects after projection is dropped entirely.*

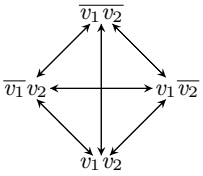
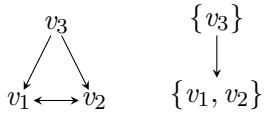


Figure 3.2: The contraction of the state space equivalent to the projection of δ on $\{v_1, v_2\}$.



(a) (b)

Figure 3.3: (a) the dependency graph of δ from Example 1, and (b) a lifted dependency graph of δ. Actions induce edges in (a): for example, v_1 and v_2 co-occur in action effects, while v_3 only happens to be in the precondition.

Example 2. *Letting $vs = \{v_1, v_2\}$, below is the projection $\delta\!\downarrow_{vs}$, for δ from Example 1. Figure 3.2 shows $\mathcal{G}(\delta\!\downarrow_{vs})$.*

$$\left\{ \begin{array}{l} p_1\!\downarrow_{vs} = (\{\overline{v_1}, \overline{v_2}\}, \{v_1\}), p_2\!\downarrow_{vs} = (\{v_1, \overline{v_2}\}, \{\overline{v_1}, v_2\}), p_3\!\downarrow_{vs} = (\{\overline{v_1}, v_2\}, \{v_1\}), \\ k_1\!\downarrow_{vs} = (\emptyset, \{v_1, v_2\}), k_2\!\downarrow_{vs} = (\emptyset, \{\overline{v_1}, v_2\}), k_3\!\downarrow_{vs} = (\emptyset, \{v_1, \overline{v_2}\}), k_4\!\downarrow_{vs} = (\emptyset, \{\overline{v_1}, \overline{v_2}\}) \end{array} \right\}$$

State variable dependency analysis captures many structures present in factored systems. It is used to obtain practically very useful projections, e.g. projections on sets of variables that are closed under mutual dependency. Also in many cases upper-bounding algorithms rely on the state variable dependencies which are described by the *dependency graph*.

Definition 8 (Dependency)**.** *A variable v_2 is dependent on v_1 in δ (written $v_1 \rightarrow v_2$) iff one of the following statements holds:* [1] *(i) v_1 is the same as v_2, (ii) there is $(p, e) \in \delta$ such that $v_1 \in \mathcal{D}(p)$ and $v_2 \in \mathcal{D}(e)$, or (iii) there is a $(p, e) \in \delta$ such that both v_1 and v_2 are in $\mathcal{D}(e)$. A set of variables vs_2 is dependent on vs_1 in δ (written $vs_1 \rightarrow vs_2$) iff: (i) vs_1 and vs_2 are disjoint, and (ii) there are $v_1 \in vs_1$ and $v_2 \in vs_2$, where $v_1 \rightarrow v_2$.*

Definition 9 (Dependency Graph)**.** *This graph, sometimes called the* causal graph*, was described independently by Knoblock Knoblock (1994) and then Williams an Nayak Williams and Nayak (1997). $\mathcal{G}_{\mathcal{D}(\delta)}$ is a dependency graph of δ iff (i) its vertices are bijectively labelled by variables from $\mathcal{D}(\delta)$, and (ii) it has an edge from vertex u_1 to u_2 iff $v_1 \rightarrow v_2$, where v_1 and v_2 are the labels of u_1 and u_2, respectively.*

Definition 10 (Lifted Dependency Graph)**.** *$\mathcal{G}_{\mathrm{VS}}$ is a lifted dependency graph of δ iff (i) its vertices are bijectively labelled by members of a partition of $\mathcal{D}(\delta)$, and (ii) it has an edge from vertex u_1 to u_2 (labelled by vs_1 and vs_2, respectively) iff $vs_1 \rightarrow vs_2$.*

Example 3. *Figure 3.3a and Figure 3.3b show the dependency graph and a lifted dependency graph of δ from Example 1.*

As stated earlier, many compositional algorithms compute reachability in succinct digraphs successfully exploit structures in the variable dependencies of a system via computing reachability in projections. Analogously, to compositionally bound the diameter (resp. the recurrence

[1] Our definition is equivalent to those in Williams and Nayak (1997); Knoblock (1994) in the context of AI planning.

diameter) one can imagine a function f that takes the projections' diameters and the dependencies between those projections and returns an upper bound on the diameter of the entire system. As arguments to f, projections' diameters and their dependencies are encoded as a $\mathbb{N}$-graph, $\mathcal{G}_{\mathbb{N}}$, with one vertex per projection and every edge represents a dependency between two projections. Every vertex is labelled by the diameter of the corresponding projection.

In the coming sections we show that there is no dependency structure (e.g. acyclic dependencies, symmetric dependencies, etc.) for which such a compositional upper-bounding scheme can work. This holds for both, (i) the diameter (Theorem 1), and (ii) the recurrence diameter (Theorem 2). Theorem 1 (resp. Theorem 2) shows that given projections' diameters (resp. recurrence diameters) and the dependencies between the projections, we can always construct a system with an arbitrarily large diameter, that has projections with diameters and dependencies as the given ones. Thus a function f that takes only projections' diameters and their dependencies cannot compute an upper bound on the given system's diameter, since they do not have enough information.

3.3.2 Diameter Cannot be Compositionally Upper-Bounded

Theorem 1. *For any function $f : \mathbb{N}$-graph $\Rightarrow \mathbb{N}$, and an $\mathbb{N}$-graph, $\mathcal{G}_{\mathbb{N}}$ that has at least one edge, there is a factored system δ such that: (i) $f(\mathcal{G}_{\mathbb{N}}) < d(\delta)$, and (ii) there is a lifted dependency graph $\mathcal{G}_{\text{VS}}$ for δ, such that $\mathcal{G}_{\mathbb{N}} = \mathfrak{D}(\!|\mathcal{G}_{\text{VS}}|\!)$, where $\mathfrak{D}(vs) = d(\delta\lfloor_{vs})$.*[2]

The proof is made of three main steps. Firstly, for each given projection diameter n (i.e. label in $\mathcal{G}_{\mathbb{N}}$) we construct a factored system γ_n, that is a path with diameter n. Those paths are constructed such that: i) their union is a system with a diameter that is more than $f(\mathcal{G}_{\mathbb{N}})$, and ii) they are projections of the final construction δ, so their domains are disjoint, as the projections are supposed to be performed on a partition. Secondly, for every dependency from projection γ^{u_1} to γ^{u_2} (i.e. an edge $\mathcal{G}_{\mathbb{N}}$), we construct an action that has preconditions from γ^{u_1} and effects from γ^{u_2}. Those actions are supposed to not change the state space of the final construction, they only add dependencies corresponding to the edges in $\mathcal{G}_{\mathbb{N}}$. Thirdly, we show that the union of the constructed projections and the dependency inducing actions is the required witness δ, i.e. its diameter exceeds $f(\mathcal{G}_{\mathbb{N}})$.

To facilitate our constructions, we use the following notation for states: states are annotated with a superscript, such that variables in states' domains are chosen to have different names if the superscripts of the states are different.[3] Formally, for two states x_a^i and x_b^j, $\mathcal{D}(x_a^i) = \mathcal{D}(x_b^j)$ if $i = j$, and $\mathcal{D}(x_a^i) \cap \mathcal{D}(x_b^j) = \emptyset$ otherwise. A path system is defined as $\gamma_n^i = \{(x_j^i, x_{j+1}^i) \mid 1 \leq j \leq n+1\}$, where the state space γ_n^i contains a simple path with $n+1$ states. Note that $d(\gamma_n^i) = n$.

Proof. Let $n = f(\mathcal{G}_{\mathbb{N}}) + 1$. This is the number that the constructed system will have as its diameter. Take an arbitrary edge $(u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})$. For $u_3 \in V(\mathcal{G}_{\mathbb{N}})$, let γ^{u_3} denote the simple path $\gamma^{u_2}_{n+\mathcal{G}_{\mathbb{N}}(u_2)}$ for $u_3 = u_2$ and $\gamma^{u_3}_{\mathcal{G}_{\mathbb{N}}(u_3)}$ otherwise. Firstly, we construct a system δ_{u_2} whose state space has a simple path with $\mathcal{G}_{\mathbb{N}}(u_2)$ states and a clique with n states attached to the last state in the path. Thus, δ_{u_2} has $\mathcal{G}_{\mathbb{N}}(u_2)$ as its diameter. Formally $\delta_{u_2} = \{(x_i^{u_2}, x_{i+1}^{u_2}) \mid 1 \leq i \leq \mathcal{G}_{\mathbb{N}}(u_2)\} \cup \{(x_i^{u_2}, x_j^{u_2}) \mid \mathcal{G}_{\mathbb{N}}(u_2) \leq i, j \leq n + \mathcal{G}_{\mathbb{N}}(u_2) + 1\}$.

Consider a fresh variable z (i.e. $z \notin \mathcal{D}(\gamma^{u_3})$, for any $u_3 \in V(\mathcal{G}_{\mathbb{N}})$). Let $\delta'_{u_2} = \{(\{z\} \uplus p, e) \mid (p, e) \in \delta_{u_2}\} \cup \{(\{\overline{z}\} \uplus p, e) \mid (p, e) \in \gamma^{u_2}\}$, i.e. a transition system with two modes of operation determined by the value of z. If z is false then it operates as a path with a clique attached to its end, i.e. exactly like δ_{u_2}. Otherwise, it operates as a path of length $n + \mathcal{G}_{\mathbb{N}}(u_2) + 1$, i.e. exactly

[2] $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\text{VS}}$: a vertex labelled by vs in $\mathcal{G}_{\text{VS}}$ is relabelled by the diameter of $\delta\lfloor_{vs}$

[3] In our proofs we use the nodes in the dependency graph as a superscript.

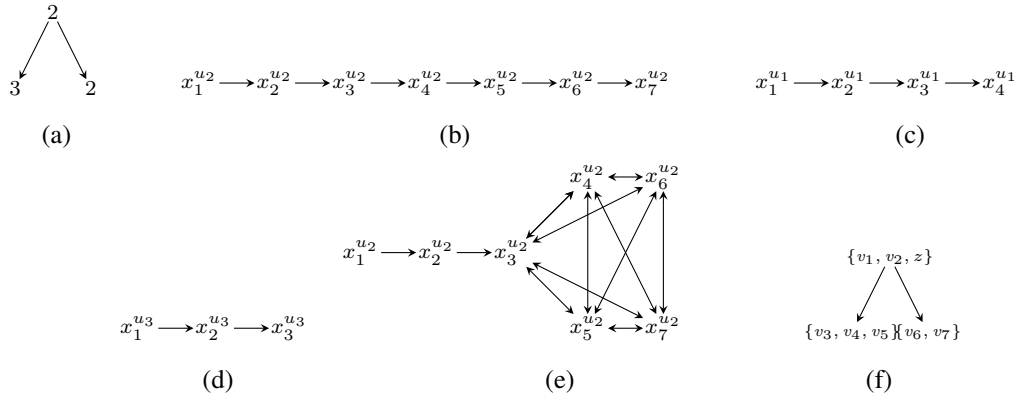


Figure 3.4: Referring to Example 4: (a) is the natural number labelled graph. (b), (c), (d) and (e) are the largest connected components in $\mathcal{G}(\gamma^{u_2})$, $\mathcal{G}(\gamma^{u_1})$, $\mathcal{G}(\gamma^{u_3})$, and $\mathcal{G}(\delta_{u_2})$, respectively. (f) is a lifted dependency graph of the factored system.

like γ^{u_2}. In this way the largest component in the state space of δ'_{u_2} has a short diameter if z is removed from the actions, otherwise the diameter blows up to $d(\delta'_{u_2}) = n + \mathcal{G}_{\mathbb{N}}(u_2)$.

Let $\delta_{u_1} = \{(\{\overline{z}\} \uplus p, \{\overline{z}\} \uplus e) \mid (p, e) \in \gamma^{u_1}\}$. This system operates exactly as the path γ^{u_1} when z is false. Since the value of z does not change, $d(\delta_{u_1}) = \mathcal{G}_{\mathbb{N}}(u_1)$. We now show that $\delta = \{(x_1^{u_3} \uplus x_1^{u_4}, x_2^{u_4}) \mid (u_3, u_4) \in E(\mathcal{G}_{\mathbb{N}})\} \cup \delta'_{u_2} \cup \delta_{u_1} \cup \bigcup_{u_3 \in V(\mathcal{G}_{\mathbb{N}})} \gamma^{u_3}$ is our required witness.

To show that δ satisfies the first requirement (i), consider the states $x_1 = \{\overline{z}\} \uplus \biguplus_{u_3 \in V(\mathcal{G}_{\mathbb{N}})} x_1^{u_3}$ and $x_2 = \{\overline{z}\} \uplus x^{u_2}_{n+\mathcal{G}_{\mathbb{N}}(u_2)+1} \uplus \biguplus_{u_3 \in V(\mathcal{G}_{\mathbb{N}}) \setminus u_2} x_1^{u_3}$. The action sequence $\overrightarrow{\pi} = [(\{\overline{z}\} \uplus x_1^{u_2}, x_2^{u_2}); (\{\overline{z}\} \uplus x_2^i, x_3^{u_2}) \ldots; (\{\overline{z}\} \uplus x^{u_2}_{n+\mathcal{G}_{\mathbb{N}}(u_2)}, x^{u_2}_{n+\mathcal{G}_{\mathbb{N}}(u_2)+1})]$ reaches x_2 if executed at x_1, and there is not an action sequence from δ^* that can reach x_2 from x_1 that is shorter than $\overrightarrow{\pi}$. This means that the diameter of δ is more than the length of $\overrightarrow{\pi}$, which is $n + \mathcal{G}_{\mathbb{N}}(u_2)$.

To show that δ satisfies the second requirement (ii) on the witness, consider a relabelling, $\mathcal{G}_{\text{VS}}$, of $\mathcal{G}_{\mathbb{N}}$, where u_1 is relabelled by $\mathcal{D}(\delta_{u_1})$ and every other every vertex u_3 is relabelled by the domain of the path γ^{u_3}. Recall that δ had the set of actions $\{(x_1^{u_3} \uplus x_1^{u_4}, x_2^{u_4}) \mid (u_3, u_4) \in E(\mathcal{G}_{\mathbb{N}})\}$ as a subset. These actions are constructed such that they add dependency from $\mathcal{D}(\gamma^{u_3})$ to $\mathcal{D}(\gamma^{u_4})$ in δ iff $(u_3, u_4) \in E(\mathcal{G}_{\mathbb{N}})$. This means that the edges of $\mathcal{G}_{\text{VS}}$ represent the dependencies of δ and accordingly it is a lifted dependency graph of δ. Also since, $\delta|_{\mathcal{D}(\gamma^{u_1})} = \gamma^{u_1}$, for $u_1 \in V(\mathcal{G}_{\mathbb{N}})$, and since by construction $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\text{VS}}$, we have $\mathcal{G}_{\mathbb{N}} = \mathfrak{D}(\!|\mathcal{G}_{\text{VS}}|\!)$. □

Example 4. *This example shows the previous construction for a function* $f : \mathbb{N}$*-graph* $\Rightarrow \mathbb{N}$*, and the graph* $\mathcal{G}_{\mathbb{N}}$ *shown in Figure 3.4a, where in this example* $f(\mathcal{G}_{\mathbb{N}}) = 4$*, and accordingly* $n = 5$ *. In* $\mathcal{G}_{\mathbb{N}}$ *there are three vertices* u_1 *(the root),* u_2*, and* u_3*, labelled by the numbers* $2, 3$*, and* 2*, respectively. In this case the first simple path system is* $\gamma^{u_2} = \{(x_1^{u_2}, x_2^{u_2}), (x_2^{u_2}, x_3^{u_2}), (x_3^{u_2}, x_4^{u_2}), (x_4^{u_2}, x_5^{u_2}), (x_5^{u_2}, x_6^{u_2}), (x_6^{u_2}, x_7^{u_2})\}$*, where* $x_1^{u_2} = \{\overline{v_3}, \overline{v_4}, \overline{v_5}\}$*,* $x_2^{u_2} = \{\overline{v_3}, \overline{v_4}, v_5\}$*,* $x_3^{u_2} = \{\overline{v_3}, v_4, \overline{v_5}\}$*,* $x_4^{u_2} = \{\overline{v_3}, v_4, v_5\}$*,* $x_5^{u_2} = \{v_3, \overline{v_4}, \overline{v_5}\}$*,* $x_6^{u_2} = \{v_3, \overline{v_4}, v_5\}$*, and* $x_7^{u_2} = \{v_3, v_4, \overline{v_5}\}$*. The second simple path system is* $\gamma^{u_1} = \{(x_1^{u_1}, x_2^{u_1}), (x_2^{u_1}, x_3^{u_1}), (x_3^{u_1}, x_4^{u_1})\}$*, where* $x_1^{u_1} = \{\overline{v_1}, \overline{v_2}\}$*,* $x_2^{u_1} = \{\overline{v_1}, v_2\}$*,* $x_3^{u_1} = \{v_1, \overline{v_2}\}$*, and* $x_4^{u_1} = \{v_1, v_2\}$*. The third simple path system is* $\gamma^{u_3} = \{(x_1^{u_3}, x_2^{u_3}), (x_2^{u_3}, x_3^{u_3})\}$*, where* $x_1^{u_3} = \{\overline{v_6}, \overline{v_7}\}$*,* $x_2^{u_3} = \{\overline{v_6}, v_7\}$*, and* $x_3^{u_3} = \{v_6, \overline{v_7}\}$*. The largest connected components of the state spaces of the three paths are shown in Figures 3.4b-3.4d, and their diameters are* $d(\gamma^{u_2}) = n + \mathcal{G}_{\mathbb{N}}(u_1) = 6$*,* $d(\gamma^{u_1}) = \mathcal{G}_{\mathbb{N}}(u_2) = 3$*, and* $d(\gamma^{u_3}) = \mathcal{G}_{\mathbb{N}}(u_3) = 2$*.*

The system δ_{u_2} *that has a simple path of length* $\mathcal{G}_{\mathbb{N}}(u_2) = 3$ *and a clique of size* $n = 5$ *attached*

to its last state, $x_3^{u_2}$, is

$$\delta_{u_2} \quad = \quad \{(x_1^{u_2}, x_2^{u_2}), (x_2^{u_2}, x_3^{u_2})\} \cup \{(x_i^{u_2}, x_j^{u_2}) \mid 3 \leq i, j \leq 7\}$$

The largest component of its state space is shown in Figure 3.4e and $d(\delta_{u_2}) = \mathcal{G}_{\mathbb{N}}(u_2) = 3$. To construct the system that has two modes of operation, δ'_{u_2}, we use a variable z to determine the two modes of δ'_{u_2}. We add the literal z to the preconditions of every action in δ_{u_2} to ensure that they are activated when z is true, and we add the literal $\overline{z}$ to the preconditions of every action in γ^{u_2} to ensure that they are activated when $\overline{z}$ is false. Thus, $\delta'_{u_2} = \{(\{z\} \uplus p, e) \mid (p, e) \in \delta_{u_2}\} \cup \{(\{\overline{z}\} \uplus p, e) \mid (p, e) \in \gamma^{u_2}\}$, where for instance, the action $(x_4^{u_2}, x_5^{u_2})$ from δ_{u_2} becomes $(\{z\} \uplus x_4^{u_2}, x_5^{u_2}) = (\{z, \overline{v_3}, v_4, v_5\}, \{v_3, \overline{v_4}, \overline{v_5}\})$ and the action $(x_4^{u_2}, x_5^{u_2})$ from γ^{u_2} becomes $(\{\overline{z}\} \uplus x_4^{u_2}, x_5^{u_2}) = (\{\overline{z}\} \uplus x_4^{u_2}, x_5^{u_2}) = (\{\overline{z}, \overline{v_3}, v_4, v_5\}, \{v_3, \overline{v_4}, \overline{v_5}\})$.

To construct δ_{u_1} we add $\overline{z}$ to the preconditions and effects of each action in γ^{u_1}. δ_{u_1} will have two modes of operation depending on the value of z. If z is true no actions at all can be executed, otherwise actions from γ^{u_1} can be executed, and accordingly $d(\delta_{u_1}) = d(\gamma^{u_1})$.

The required witness is $\delta = \{(x_1^{u_1} \uplus x_1^{u_2}, x_2^{u_2}), (x_1^{u_1} \uplus x_1^{u_3}, x_2^{u_3})\} \cup \delta'_{u_2} \cup \delta_{u_1} \cup \gamma^{u_3}$. The combined system has a diameter of 6 because it has a path isomorphic to the path in γ^{u_2} from $\{\overline{z}\} \uplus x_1^{u_2}$ $\{\overline{z}\} \uplus x_7^{u_2}$. In particular, this path is between the state $x_1 = \{\overline{z}\} \uplus x_1^{u_2} \uplus x_1^{u_1} \uplus x_1^{u_3} = \{\overline{z}, \overline{v_3}, \overline{v_4}, \overline{v_5}, \overline{v_1}, \overline{v_2}, \overline{v_6}, \overline{v_7}\}$ and the state $x_2 = \{\overline{z}\} \uplus x_7^{u_2} \uplus x_1^{u_1} \uplus x_1^{u_3}\{\overline{z}, v_3, v_4, \overline{v_5}, \overline{v_1}, \overline{v_2}, \overline{v_6}, \overline{v_7}\}$. The action sequence $\overrightarrow{\pi} = [(\overline{z} \uplus x_1^{u_2}, x_2^{u_2}); (\overline{z} \uplus x_2^{u_2}, x_3^{u_2}); (\overline{z} \uplus x_3^{u_2}, x_4^{u_2}); (\overline{z} \uplus x_4^{u_2}, x_5^{u_2}); (\overline{z} \uplus x_5^{u_2}, x_6^{u_2}); (\overline{z} \uplus x_6^{u_2}, x_7^{u_2})]$ reaches x_2 if executed at x_1, and no action sequence shorter than $|\overrightarrow{\pi}|$ can reach x_2 from x_1. This means that $n + \mathcal{G}_{\mathbb{N}}(u_1) = 6 \leq d(\delta)$. Also, Figure 3.4f is a lifted dependency graph of δ, as it has the actions $\{(x_1^{u_1} \uplus x_1^{u_2}, x_2^{u_2}), (x_1^{u_1} \uplus x_1^{u_3}, x_2^{u_3})\}$.

3.3.3 Recurrence Diameter Cannot be Compositionally Bounded

Theorem 2. *For a function $f : \mathbb{N}$-graph $\Rightarrow \mathbb{N}$, and an $\mathbb{N}$-graph, $\mathcal{G}_{\mathbb{N}}$, there is a system δ where: (i) $f(\mathcal{G}_{\mathbb{N}}) < rd(\delta)$, and (ii) there is a lifted dependency graph $\mathcal{G}_{\text{VS}}$ for δ, such that $\mathcal{G}_{\mathbb{N}} = \mathfrak{R}(\!|\mathcal{G}_{\text{VS}}|\!)$, where $\mathfrak{R}(vs) = rd(\delta\lfloor_{vs})$ (i.e. $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\text{VS}}$: a vertex labelled by vs in $\mathcal{G}_{\text{VS}}$ is relabelled by the recurrence diameter of $\delta\lfloor_{vs}$).*

The proof of this theorem is similar in structure to that of Theorem 1. One difference is that instead of constructing paths for projections, we construct systems whose state spaces are "flowers" (Figure 3.5). A flower can have arbitrarily many states without changing its recurrence diameter. This is needed to construct arbitrarily long paths with distinct all states in the final construction, while keeping the recurrence diameters of projections constant. Flowers are formally described as follows. First let a petal $\eta^i_{l,m}$ be a loop of $\frac{m}{2} + 1$ states labelled with 0 and $\frac{m}{2}$ other numbers. We formally define it as $\eta^i_{l,m} = \{(x^i_{(l-2)c+k}, x^i_{(l-2)c+(k+1 \bmod a)}) \mid 0 \leq k \leq a-1\}$, where $a = \lceil \frac{m}{2} \rceil + 1$ if m is odd, and $a = \frac{m}{2} + 1$ otherwise, and $c = \lfloor \frac{m}{2} \rfloor + 1$. A flower system, $\mathbb{1}^i_{l,m}$, denotes $\bigcup_{1 \leq j < k \leq l} \{(x^i_j, x^i_k), (x^i_k, x^i_j)\}$ (i.e. a clique) if $m = 1$, and $\eta^i_{l,m} \cup \bigcup_{1 \leq j \leq l-2} \{(x^i_{jc+k}, x^i_{jc+(k+1 \bmod c)}) \mid 0 \leq k \leq c-1\}$ otherwise. $\mathbb{1}^i_{l,m}$ is a factored system whose state space contains l cycles (the petals). All other petals are of length $\lfloor \frac{m}{2} \rfloor + 1$, except for the last one, if m is odd its length is $\lceil \frac{m}{2} \rceil + 1$ otherwise its length is $\frac{m}{2} + 1$. *All* of the petals have exactly *one* state in common between them (the pistil), which is x^i_0. In each petal j there is a state t^i_j (the "petal tip"). $t^i_j = x^i_{(l-2)c+\lceil \frac{a}{2} \rceil}$ for the last petal (i.e. $j = l-1$), and otherwise $t^i_j = x^i_{jc+\lceil \frac{c}{2} \rceil}$. t^i_0 can be reached by exactly $\lceil \frac{a}{2} \rceil$ actions from x^i_0, and x^i_0 can be reached from t^i_0 by exactly $\lfloor \frac{a}{2} \rfloor$ actions. Similarly, for $j \neq 1$, t^i_j can be reached by exactly $\lceil \frac{c}{2} \rceil$ actions from x^i_0, and x^i_0 can be reached from t^i_j by exactly $\lfloor \frac{c}{2} \rfloor$ actions. Denoting by $\overrightarrow{\pi}^i_{a \mapsto b}$ the shortest action sequence that joins the petal tip

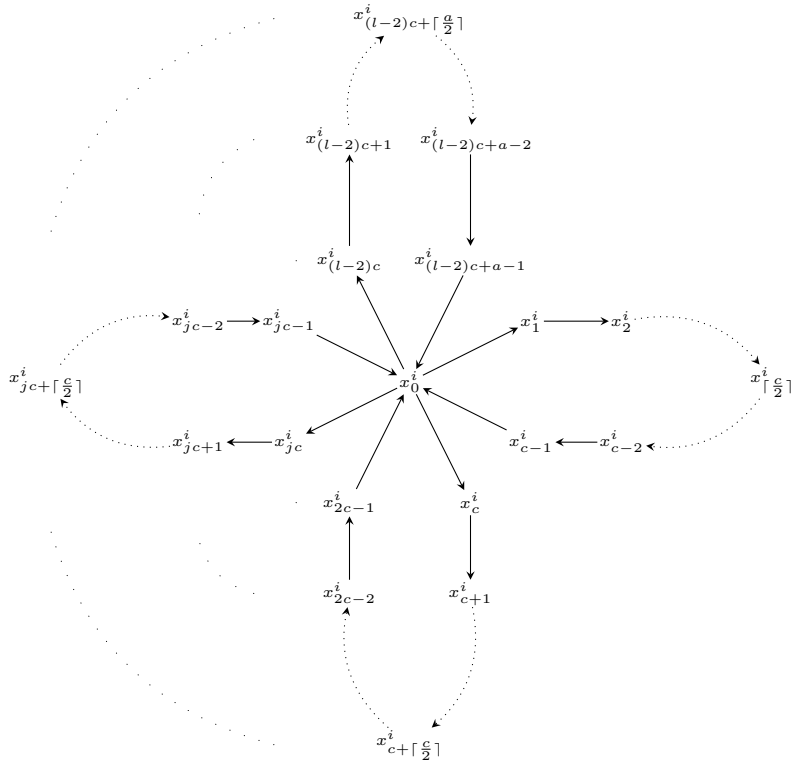


Figure 3.5: The largest connected component in the state space of a flower with $2 \leq m$.

t^i_a with t^i_b, $c \leq |\overrightarrow{\pi}^i_{a\mapsto b}|$ holds. Also $rd(\mathbb{I}^i_{l,m}) = m$ holds. Lastly we use $\mathcal{S}(\mathbb{I}^u)$ to denote the largest connected component for any state in the state space of $\mathbb{I}^u$, which is always unique for a flower

Proof. Let $n = f(\mathcal{G}_{\mathbb{N}})$. This is the number that the constructed system will have as its recurrence diameter. For every vertex $u_1 \in V(\mathcal{G}_{\mathbb{N}})$, let $\mathbb{I}^{u_1}$ denote the flower $\mathbb{I}^{u_1}_{n+2,\mathcal{G}_{\mathbb{N}}(u_1)}$. Let $\delta = \{(x^{u_1}_0 \uplus x^{u_2}_0, x^{u_2}_1) \mid (u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})\} \cup \bigcup_{u_1 \in V(\mathcal{G}_{\mathbb{N}})} \mathbb{I}^{u_1}$. For each flower $\mathbb{I}^{u_1}$, denote its $n+2$ petal tips by $t^{u_1}_j$, for $0 \leq j \leq n+1$. Consider the two states: (i) $x_1 = \biguplus_{u_1 \in V(\mathcal{G}_{\mathbb{N}})} t^{u_1}_0$, and (ii) $x_2 = \biguplus_{u_1 \in V(\mathcal{G}_{\mathbb{N}})} t^{u_1}_{n+1}$. We show that δ satisfies requirement (i) on the witness by giving a simple path between the states x_1 and x_2, whose length is more than n. Consider the action sequence $\overrightarrow{\pi}$ constructed by Algorithm 1.

Algorithm 1:
$\overrightarrow{\pi}$=: []
for $0 \leq i \leq n+1$ // Looping over petals
  for $u_1 \in V(\mathcal{G}_{\mathbb{N}})$ // Looping over flowers
    $\overrightarrow{\pi}$ =: $\overrightarrow{\pi} \frown \overrightarrow{\pi}^{u_1}_{i\mapsto i+1}$[4] // Go from petal tip $t^{u_1}_i$ to $t^{u_1}_{i+1}$

The length of $\overrightarrow{\pi}$ is at least $(n+2)\sum_{u_1 \in V(\mathcal{G}_{\mathbb{N}})}(c) - 1$, since $c \leq |\overrightarrow{\pi}^{u_1}_{a\mapsto b}|$ for $a \neq b$. Since $c = \lfloor \frac{\mathcal{G}_{\mathbb{N}}(u_1)}{2} \rfloor + 1$ then $n \leq |\overrightarrow{\pi}|$. Also $\overrightarrow{\pi}$ is constructed such that, if $\overrightarrow{\pi}$ is executed at x_1 it only traverses distinct states, i.e. it induces a simple path.

To show that δ satisfies requirement (ii), consider a relabelling, $\mathcal{G}_{\text{VS}}$, of $\mathcal{G}_{\mathbb{N}}$, where every vertex u_1 is relabelled by $\mathcal{D}(\mathbb{I}^{u_1})$, the domain of the flower $\mathbb{I}^{u_1}$ associated with the vertex. Recall that

[4]For two lists l_1 and l_2, $l_1 \frown l_2$ denotes their concatenation.

δ had the set of actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}) \mid (u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})\}$ as a subset. These actions are constructed such that they add dependency from $\mathcal{D}(\mathbb{I}^{u_1})$ to $\mathcal{D}(\mathbb{I}^{u_2})$ in δ iff $(u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})$. This means that the edges of $\mathcal{G}_{\text{VS}}$ represent the dependencies of δ and accordingly it is a lifted dependency DAG of δ. Also since $\delta|_{\mathcal{D}(\mathbb{I}^{u_1})} = \mathbb{I}^{u_1}$, and by construction $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\text{VS}}$, we have $\mathcal{G}_{\mathbb{N}} = \mathfrak{R}(\!|\mathcal{G}_{\text{VS}}|\!)$. □

Example 5. *This example shows the previous construction for a function $f : \mathbb{N}$-graph$\Rightarrow \mathbb{N}$, and the graph $\mathcal{G}_{\mathbb{N}}$ shown in Figure 3.4a, where $f(\mathcal{G}_{\mathbb{N}}) = 2$. In this graph there are three vertices u_1 (the root), u_2, and u_3, labelled by the numbers $2, 3$, and 2, respectively. We construct three flowers, one per vertex. For u_2 the constructed flower is $\mathbb{I}^{u_2}_{4,3} = \{(x_0^{u_2}, x_4^{u_2}), (x_4^{u_2}, x_5^{u_2}), (x_5^{u_2}, x_0^{u_2})\} \cup \bigcup\{\{(x_0^{u_2}, x_l^{u_2}), (x_l^{u_2}, x_0^{u_2})\} \mid 1 \leq l \leq 3\}$ where the states are defined as follows $x_0^{u_2} = \{\overline{v_4}, \overline{v_5}, \overline{v_6}\}$ (the pistil), $x_1^{u_2} = \{\overline{v_4}, \overline{v_5}, v_6\}$ (the first petal tip), $x_2^{u_2} = \{\overline{v_4}, v_5, \overline{v_6}\}$ (the second petal tip), $x_3^{u_2} = \{\overline{v_4}, v_5, v_6\}$ (the third petal tip), $x_4^{u_2} = \{v_4, \overline{v_5}, \overline{v_6}\}$, $x_5^{u_2} = \{v_4, \overline{v_5}, v_6\}$ (the fourth petal tip). Note that since for the flower $\mathbb{I}^{u_2}_{4,3}$, $m = 3$ (i.e. odd), the last petal has one more state in it (the petal at the bottom of Figure 3.6a). For u_3 the constructed flower is $\mathbb{I}^{u_3}_{4,2} = \bigcup\{\{(x_0^{u_3}, x_l^{u_3}), (x_l^{u_3}, x_0^{u_3})\} \mid 1 \leq l \leq 4\}$, where the states are defined as follows $x_0^{u_3} = \{\overline{v_7}, \overline{v_8}, \overline{v_9}\}$ (the pistil), $x_1^{u_3} = \{\overline{v_7}, \overline{v_8}, v_9\}$ (the first petal tip), $x_2^{u_3} = \{\overline{v_7}, v_8, \overline{v_9}\}$ (the second petal tip), $x_3^{u_3} = \{\overline{v_7}, v_8, v_9\}$ (the third petal tip), and $x_4^{u_3} = \{v_7, \overline{v_8}, \overline{v_9}\}$ (the fourth petal tip). For u_1 the constructed flower is $\mathbb{I}^{u_1}_{4,2} = \{\{(x_0^{u_1}, x_1^{u_1}), (x_l^{u_1}, x_0^{u_1})\} \mid 1 \leq 4\}$, where the states are defined as follows $x_0^{u_1} = \{\overline{v_1}, \overline{v_2}, \overline{v_3}\}$ (the pistil), $x_1^{u_1} = \{\overline{v_1}, \overline{v_2}, v_3\}$ (the first petal tip), $x_2^{u_1} = \{\overline{v_1}, v_2, \overline{v_3}\}$ (the second petal tip), $x_3^{u_1} = \{\overline{v_1}, v_2, v_3\}$ (the third petal tip), and $x_4^{u_1} = \{v_1, \overline{v_2}, \overline{v_3}\}$ (the fourth petal tip).*

The required witness $\delta = \mathbb{I}^{u_1}_{4,2} \cup \mathbb{I}^{u_2}_{4,3} \cup \mathbb{I}^{u_3}_{4,2} \cup \{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\}$ where the actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\}$ add to δ dependencies equivalent to the edges of $\mathcal{G}_{\mathbb{N}}$, i.e. the dependencies shown in Figure 3.6d.

Consider the states $x_1 = x_1^{u_1} \uplus x_1^{u_2} \uplus x_1^{u_3} = \{\overline{v_1}, \overline{v_2}, v_3, \overline{v_4}, \overline{v_5}, v_6, \overline{v_7}, \overline{v_8}, v_9\}$ and $x_2 = x_4^{u_1} \uplus x_4^{u_2} \uplus x_4^{u_3} = \{v_1, \overline{v_2}, \overline{v_3}, l, \overline{v_5}, v_6, v_7, \overline{v_8}, \overline{v_9}\}$. Following Algorithm 1, the resulting action sequence is $\overrightarrow{\pi} = \pi^{\rightarrow u_1}_{1\mapsto 2} \frown \pi^{\rightarrow u_2}_{1\mapsto 2} \frown \pi^{\rightarrow u_3}_{1\mapsto 2} \frown \pi^{\rightarrow u_1}_{2\mapsto 3} \frown \pi^{\rightarrow u_2}_{2\mapsto 3} \frown \pi^{\rightarrow u_3}_{2\mapsto 3} \frown \pi^{\rightarrow u_1}_{3\mapsto 4} \frown \pi^{\rightarrow u_2}_{3\mapsto 4} \frown \pi^{\rightarrow u_3}_{3\mapsto 4}$ and its length is 18. $\overrightarrow{\pi}$ will reach x_2 if executed at x_1, while traversing all distinct states. The largest connected component of $\mathcal{G}(\delta)$ and the path traversed by executing $\overrightarrow{\pi}$ from x_1 are shown in Figure 3.6e.

3.3.4 Discussion

In both proofs above, it might seem that the constructions and accordingly the theorem statements are based on a bad choice of projections. However, the point that these results make is subtle but more profound: any compositional algorithm cannot know whether the given projections are "sound" just by "looking" in the dependency graph. In other words, if a compositional algorithm only uses the dependency graph, then there will always be a rouge projection. Thus to guarantee that the given projections can be soundly used for upper bounding, an algorithm needs more information than just what is available in the dependency graph. This accordingly shows that all existing compositional bounding algorithms that use only dependency structures (which are the majority) need some non-trivial extensions that look into more than just dependencies (e.g. they need to look into state space structures) in order for them to bound the system's (recurrence) diameter using projections' (recurrence) diameters.

3.4 The Traversal Diameter

The traversal diameter is one less than the largest number of states that can be traversed by any path. It is defined as follows.

Definition 11 (Traversal Diameter)**.** *Let* $\mathsf{ss}(x, \overrightarrow{\pi})$ *be the set of states traversed by executing* $\overrightarrow{\pi}$ *from* x*.*

$$td(\delta) = \max_{x \in \mathbb{U}(\delta)} \max_{\overrightarrow{\pi} \in \delta^*} |\mathsf{ss}(x, \overrightarrow{\pi})| - 1.$$

Example 6. *For* $\delta\lfloor_{vs}$ *from Example 2 the traversal diameter is 3, since the longest path in its state space has 4 states.*

The most appealing feature of the traversal diameter is that it is an upper bound on the recurrence diameter (and accordingly the diameter) that can be compositionally bounded with projections from arbitrary partitions of the state variables, as shown in the following theorem.

Theorem 3. *For a factored representation* δ *and a partition* $vs_{1..n}$ *of* $\mathcal{D}(\delta)$*,* $td(\delta) \leq \prod_{vs \in vs_{1..n}} (td(\delta\lfloor_{vs}) + 1) - 1$*.*

Proposition 1. *For some* k*, if for every* $x \in \mathbb{U}(\delta)$ *and* $\overrightarrow{\pi} \in \delta^*$*,* $|\mathsf{ss}(x, \overrightarrow{\pi})| \leq k+1$*, then* $td(\delta) \leq k$*.*

Proposition 2. *For a set of states* S*, let* $S\lfloor_{vs}$ *denote* $\{x\lfloor_{vs} \mid x \in S\}$*. Let* $\mathsf{sat\text{-}pre}(x, \overrightarrow{\pi})$ *denote that preconditions of every action in* $\overrightarrow{\pi}$ *are satisfied, if* $\overrightarrow{\pi}$ *is executed from* x*. If* $\mathsf{sat\text{-}pre}(x, \overrightarrow{\pi})$*, then* $\mathsf{ss}(x, \overrightarrow{\pi})\lfloor_{vs} = \mathsf{ss}(s\lfloor_{vs}, \overrightarrow{\pi}\lfloor_{vs})$*.*

Proposition 3. *For any partition* $vs_{1..n}$ *of* $\mathcal{D}(\delta)$*,* $|\mathsf{ss}(x, \overrightarrow{\pi})| \leq \prod_{vs \in vs_{1..n}} |\mathsf{ss}(x, \overrightarrow{\pi})\lfloor_{vs}|$*.*

The proof of the above proposition comes from the fact that a set of states is a subset of the cartesian product of its own projections, given that the projection is on a partition of the state variables.

Proof of Theorem 3. Consider $x \in \mathbb{U}(\delta)$ and without loss of generality, an action sequence $\overrightarrow{\pi} \in \delta^*$ such that $\mathsf{sat\text{-}pre}(x, \overrightarrow{\pi})$. From Definition 11, for any vs, $x\lfloor_{vs} \in \mathbb{U}(\delta\lfloor_{vs})$ and $\overrightarrow{\pi}\lfloor_{vs} \in \delta\lfloor_{vs}{}^*$, we have $|\mathsf{ss}(x\lfloor_{vs}, \overrightarrow{\pi}\lfloor_{vs})| - 1 \leq td(\delta\lfloor_{vs})$. Theorem 3 then follows from Proposition 2, Proposition 3 and Proposition 1. □

The bound in the previous theorem is a polynomial that has all possible terms from the traversal diameters of the projections. This bound cannot be improved regardless of the dependency structure inducing the projections, as follows.

Theorem 4. *For any* $\mathbb{N}$*-graph,* $\mathcal{G}_{\mathbb{N}}$*, there is a factored system* δ *such that: (i)* $\prod_{u_1 \in V(\mathcal{G}_{\mathbb{N}})} (\mathcal{G}_{\mathbb{N}}(u_1) + 1) - 1 \leq td(\delta)$*, and (ii) there is a lifted dependency graph* $\mathcal{G}_{\mathrm{VS}}$ *for* δ*, such that* $\mathcal{G}_{\mathbb{N}} = \mathfrak{T}(\!|\mathcal{G}_{\mathrm{VS}}|\!)$*, where* $\mathfrak{T}(vs) = td(\delta\lfloor_{vs})$*.*

Proof. For $u \in V(\mathcal{G}_{\mathbb{N}})$, let $\mathbb{I}^u$ denote the flower $\mathbb{I}^u_{\mathcal{G}_{\mathbb{N}}(u),2}$. For all $u \in V(\mathcal{G}_{\mathbb{N}})$, $x \rightsquigarrow y$ holds for any $x, y \in \mathcal{S}(\mathbb{I}^u)$, thus $td(\mathbb{I}^u) = |\mathcal{S}(\mathbb{I}^u)| - 1 = \mathcal{G}_{\mathbb{N}}(u)$. Let $\delta = \{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}) \mid (u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})\} \cup \bigcup_{u \in V(\mathcal{G}_{\mathbb{N}})} \mathbb{I}^u$. We now show that δ satisfies requirement (i). First let $\mathcal{S}(\delta)$ denote the largest connected component in the state space of δ, which is unique. Since $x \rightsquigarrow y$ holds for any $x, y \in \mathcal{S}(\mathbb{I}^u)$, then $x \rightsquigarrow y$ holds for any $x, y \in \mathcal{S}(\delta)$, and therefore there is a path that traverses every member of $\mathcal{S}(\delta)$. Since for $u_1 \neq u_2$ we have $\mathbb{I}^{u_1} \cap \mathbb{I}^{u_2} = \emptyset$, we have $|\mathcal{S}(\delta)| = \prod_{u \in V(\mathcal{G}_{\mathbb{N}})} |\mathcal{S}(\mathbb{I}^u)|$. Since $\prod_{u \in V(\mathcal{G}_{\mathbb{N}})} |\mathcal{S}(\mathbb{I}^u)| = \prod_{u \in V(\mathcal{G}_{\mathbb{N}})} (\mathcal{G}_{\mathbb{N}}(u) + 1)$, we have that $\prod_{u \in V(\mathcal{G}_{\mathbb{N}})} (\mathcal{G}_{\mathbb{N}}(u) + 1) - 1 \leq td(\delta)$.

To show that δ satisfies requirement (ii), consider a relabelling, $\mathcal{G}_{\text{VS}}$, of $\mathcal{G}_{\mathbb{N}}$, where every vertex u is relabelled by the domain of the flower $\mathbb{I}^u$. Recall that δ had the set of actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}) \mid (u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})\}$ as a subset. These actions are constructed such that they add dependency from $\mathcal{D}(\mathbb{I}^{u_1})$ to $\mathcal{D}(\mathbb{I}^{u_2})$ in δ iff $(u_1, u_2) \in E(\mathcal{G}_{\mathbb{N}})$. Accordingly edges of $\mathcal{G}_{\text{VS}}$ represent the dependencies of δ and accordingly it is a lifted dependency graph of δ. Also since $\delta|_{\mathcal{D}(\mathbb{I}^u)} = \mathbb{I}^u$, for $u \in V(\mathcal{G}_{\mathbb{N}})$, and since by construction $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\text{VS}}$, we have $\mathcal{G}_{\mathbb{N}} = \mathfrak{T}(\!|\mathcal{G}_{\text{VS}}|\!)$. □

Example 7. *This is an example of the construction from Theorem 4, for the natural number labelled DAG in Figure 3.4a. We construct three flowers, one per vertex, shown in Figures 3.7a-3.7c. For u_2 the constructed flower is $\mathbb{I}^{u_2}_{3,2} = \{(x_0^{u_2}, x_1^{u_2}), (x_0^{u_2}, x_2^{u_2}), (x_0^{u_2}, x_3^{u_2}), (x_1^{u_2}, x_0^{u_2}), (x_2^{u_2}, x_0^{u_2}), (x_3^{u_2}, x_0^{u_2})\}$. The states are defined as follows $x_0^{u_2} = \{\overline{v_3}, \overline{v_4}\}$ (the pistil), $x_1^{u_2} = \{\overline{v_3}, v_4\}$ (the first petal tip), $x_2^{u_2} = \{v_3, \overline{v_4}\}$ (the second petal tip), $x_3^{u_2} = \{v_3, v_4\}$ (the third petal tip). For u_3 the constructed flower is $\mathbb{I}^{u_3}_{2,2} = \{(x_0^{u_3}, x_1^{u_3}), (x_0^{u_3}, x_2^{u_3}), (x_1^{u_3}, x_0^{u_3}), (x_2^{u_3}, x_0^{u_3})\}$. The states are defined as follows $x_0^{u_3} = \{\overline{v_5}, \overline{v_4}\}$ (the pistil), $x_1^{u_3} = \{\overline{v_5}, v_4\}$ (the first petal tip) and $x_2^{u_3} = \{v_5, \overline{v_4}\}$ (the second petal tip). For u_1 the constructed flower is $\mathbb{I}^{u_1}_{2,2} = \{(x_0^{u_1}, x_1^{u_1}), (x_0^{u_1}, x_2^{u_1}), (x_1^{u_1}, x_0^{u_1}), (x_2^{u_1}, x_0^{u_1})\}$. The states are defined as follows $x_0^{u_1} = \{\overline{v_1}, \overline{v_2}\}$ (the pistil), $x_1^{u_1} = \{\overline{v_1}, v_2\}$ (the first petal tip), and $x_2^{u_1} = \{v_1, \overline{v_2}\}$ (the second petal tip). For any $x_0, x_1 \in \mathcal{S}(\mathbb{I}^{u_1})$, $x_0 \rightsquigarrow x_1$ holds and accordingly $td(\mathbb{I}^{u_1}) = 2$. Similarly, $td(\mathbb{I}^{u_2}) = 3$ and $td(\mathbb{I}^{u_3}) = 2$.*

The required witness is $\delta = \{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\} \cup \mathbb{I}^{u_1} \cup \mathbb{I}^{u_2} \cup \mathbb{I}^{u_3}$, where the actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\}$ add dependencies equivalent to the edges of $A_{\mathbb{N}}$, i.e. the dependencies shown in Figure 3.7d. Also, in the constructed witness, for all states $x_0, x_1 \in \mathcal{S}(\delta)$ (shown in Figure 3.7) $x_0 \rightsquigarrow x_1$ holds, and accordingly $td(\delta) = 35$.

There are two reasons the traversal diameter and the above theorems are interesting. Firstly, the traversal diameter is a non-trivial upper bound on the recurrence diameter, in the sense that it can be exponentially smaller than the number of states, which is the best known upper bound on the recurrence diameter. Secondly, the traversal diameter can be computed in linear time, unlike the NP-hard recurrence diameter.

3.4.1 Tightness of the Traversal Diameter

Theorem 5. *There are infinitely many factored systems whose traversal diameter is (i) equal to their recurrence diameter, and (ii) exponentially smaller (in the number of state variables) than their state space. There are also infinitely many factored systems whose traversal diameter is (i) exponentially larger (in the number of state variables) than their recurrence diameter, and (ii) equal to the size of the largest connected component on their state space.*

Theorem 5 follows from Lemma 2 and Lemma 1:

Lemma 1. *There are infinitely many factored systems whose traversal diameters are exponentially smaller (in the number of state variables) than the size of their state spaces.*

Proof. For an arbitrary number $n \in \mathbb{N}$, we construct a system whose state space size is a factor of n more than its traversal diameter. Consider the system $\{(x_0^1, x_i^1) \mid 1 \leq i \leq n\}$. The traversal diameter of this system is 1 since, the only possible transitions are from state x_0^1 to a state x_i^1, for $1 \leq i \leq n$. However the system's state space has at least $n + 1$ states. □

Lemma 2. *There are infinitely many factored systems whose recurrence diameters are exponentially smaller (in the number of state variables) than their traversal diameters.*

Proof. We show that for an arbitrary number $n \in \mathbb{N}$, there is a system whose traversal diameter is a factor of n more than its recurrence diameter. Consider the flower system $\mathbb{I}^1_{2n,2}$. The recurrence diameter of this flower is 2 since the length of each of its petals is 1, and any action sequence that traverses more than 3 states will traverse x^1_0 (the pistil) more than once. Its traversal diameter on the other hand is $2n$, since for any $x, y \in \mathcal{S}(\mathbb{I}^1_{2n,2})$, $x \rightsquigarrow y$ holds and $|\mathcal{S}(\mathbb{I}^1_{2n,2})| = 2n + 1$. □

3.4.2 Computing the Traversal Diameter

An important aspect of td is that it can be computed in linear time using the following algorithm. However, we first formally define the *quotient* of a labelled digraph that is induced by repetitive vertex contraction operations, and define a function that computes the longest path in a given digraph weighted by a given function on the vertices.

Definition 12 (Digraph Quotient)**.** *For a digraph $\mathcal{G}_\alpha$, and a partition P of $V(\mathcal{G}_\alpha)$, the quotient $\mathcal{G}_\alpha/P$ of $\mathcal{G}_\alpha$ is the digraph:*

(i) $V(\mathcal{G}_\alpha/P) = P$,

(ii) $E(\mathcal{G}_\alpha/P) = \{(U, W) \mid U, W \in P \wedge \exists u_2 \in U, u_3 \in W.(u_2, u_3) \in E(\mathcal{G}_\alpha)\}$, *and*

(iii) the label $\mathcal{G}_\alpha/P(U)$ is $\{\mathcal{G}_\alpha(u_1) \mid u_1 \in U\}$, for $U \in V(\mathcal{G}_\alpha/P)$.

Definition 13 (Weighted Longest Path)**.** *For a digraph $\mathcal{G}_\alpha$, let the function $b : \alpha \Rightarrow \mathbb{N}$ be a function that assigns a natural number for the label of every vertex in $V(\mathcal{G}_\alpha)$.* S *is*

$$\mathsf{S}\langle b\rangle(u_1, \mathcal{G}_\alpha) = b(\mathcal{G}_\alpha(u_1)) + \max_{u_1' \in \mathsf{children}_{\mathcal{G}_\alpha}(u_1)} (\mathsf{S}\langle b\rangle(u_1', \mathcal{G}_\alpha) + 1)$$

Then, let $\mathsf{S}_{\mathsf{max}}\langle b\rangle(\mathcal{G}_\alpha) = \max_{u_1 \in V(\mathcal{G}_\alpha)} \mathsf{S}\langle b\rangle(u_1, \mathcal{G}_\alpha)$.

$\mathsf{S}_{\mathsf{max}}$ is only well-defined if $\mathcal{G}_\alpha$ is acyclic. The definition of $\mathsf{S}_{\mathsf{max}}$ follows the scheme of an algorithm that finds the length of the longest path in a DAG, so its runtime is linear in the size of $V(\mathcal{G}_\alpha)$, if the values of S for different vertices are memoised.[5] However, $\mathsf{S}_{\mathsf{max}}$ computes the weightiest path, where an edge weighs one, and a vertex u_1 weighs $b(\mathcal{G}_\alpha(u_1))$.

Algorithm 2: TRAVDIAM(δ)

SCC := set of strongly connected components of $\mathcal{G}(\delta)$
$\mathcal{G} := \mathcal{G}(\delta)/SCC$
Return $\mathsf{S}_{\mathsf{max}}\langle\mathbb{C}\rangle(\mathcal{G}) - 1$, where $\mathbb{C}(scc) = |scc|$ for $scc \in SCC$

Theorem 6. TRAVDIAM$(\delta) = td(\delta)$.

Proof. For notational brevity, let $\mathcal{G} = \mathcal{G}(\delta)/SCC$, and for a strongly connected component scc, $\mathsf{S}(scc) = \mathsf{S}\langle\mathbb{C}\rangle(scc, \mathcal{G})$ and $\mathsf{children}(scc) = \mathsf{children}_{\mathcal{G}}(scc)$. Since $\mathcal{G}$ is a DAG, its vertices can be topologically ordered in a list l_{SCC}.

Firstly, we prove TRAVDIAM$(\delta) \leq td(\delta)$. We show that for any strongly connected component $scc \in \mathcal{G}$ there is an action sequence $\vec{\pi}_{scc} \in \delta^*$ and a state $x_{scc} \in scc$, such that $\mathsf{S}(scc) \leq |\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})|$, which from Definitions 11 and 13, implies TRAVDIAM$(\delta) \leq td(\delta)$. We prove this by induction on l_{SCC}. The base case, $l_{SCC} = []$, is straightforward. For the step

[5] This assumes that b is at most of linear complexity.

case $l_{SCC} = scc :: l'_{SCC}$,[6] and for any $scc' \in l'_{SCC}$, there is a state $x' \in scc'$ and an action sequence $\vec{\pi}' \in \delta^*$ where $\mathsf{S}(scc') \leq |\mathsf{ss}(x', \vec{\pi}')|$. Since scc is a strongly connected component of states in $\mathcal{G}(\delta)$, then there is $\vec{\pi}'_{scc} \in \delta^*$ and a state $x_{scc} \in scc$, where $\vec{\pi}_{scc}$ traverses exactly all the states in scc, if executed at $x_{scc} \in scc$, i.e. $\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc}) = scc$. We have two cases:

Case 1 ($\mathsf{children}(scc) = \emptyset$)**.** *From Definition 13,* $\mathsf{S}(scc) = |scc| = |\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc})|$ *holds for this case. Accordingly the required witness* $\vec{\pi}_{scc}$ *is the same as* $\vec{\pi}'_{scc}$*.*

Case 2 ($\mathsf{children}(scc) \neq \emptyset$)**.** *Let* scc_{max} *be the strongly connected component such that* $\forall\ scc' \in \mathsf{children}(scc)$. $\mathsf{S}(scc) \leq \mathsf{S}(scc_{\mathsf{max}})$*. Because* $scc_{\mathsf{max}} \in \mathsf{children}(scc)$ *we have* $scc_{\mathsf{max}} \in l'_{SCC}$*, and accordingly from the inductive hypothesis there are* $x_{\mathsf{max}} \in scc_{\mathsf{max}}$ *and* $\vec{\pi}_{\mathsf{max}} \in \delta^*$ *such that* $\mathsf{S}(scc_{\mathsf{max}}) \leq |\mathsf{ss}(x_{\mathsf{max}}, \vec{\pi}_{\mathsf{max}})|$*.(†) Also, from* $scc_{\mathsf{max}} \in \mathsf{children}(scc)$ *and the fact that both* scc *and* scc_{max} *are strongly connected components, there must be* $\vec{\pi}' \in \delta^*$*, such that* $\mathsf{ex}(x_{scc}, \vec{\pi}'_{scc} \frown \vec{\pi}') = x_{\mathsf{max}}$*. We now show that* $\vec{\pi}_{scc} = \vec{\pi}'_{scc} \frown \vec{\pi}' \frown \vec{\pi}_{\mathsf{max}}$ *is the required witness. First it is easy to see that* $\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc}) \cup \mathsf{ss}(\mathsf{ex}(x_{scc}, \vec{\pi}'_{scc}), \vec{\pi}') \cup \mathsf{ss}(x_{\mathsf{max}}, \vec{\pi}_{\mathsf{max}}) = \mathsf{ss}(x_{scc}, \vec{\pi}_{scc})$*. Since* $scc_{\mathsf{max}} \in \mathsf{children}(scc)$*, we have that* $\mathsf{ss}(x_{\mathsf{max}}, \vec{\pi}_{\mathsf{max}})$ *is disjoint with* scc *and accordingly* $|\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc})| + |\mathsf{ss}(x_{\mathsf{max}}, \vec{\pi}_{\mathsf{max}})| \leq |\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})|$*. From this, (†), and Definition 13 we have* $\mathsf{S}(scc) \leq |\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})|$*.*

Secondly, we prove $td(\delta) \leq \textsc{TravDiam}(\delta)$ by showing that for any $scc \in \mathcal{G}$, $x_{scc} \in scc$, and $\vec{\pi}_{scc} \in \delta$, we have $|\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})| \leq \mathsf{S}(scc)$. Our proof is again by induction on the list l_{SCC}. The base case, $l_{SCC} = []$, is straightforward. The step case $l_{SCC} = scc :: l'_{SCC}$, and we have that for any strongly connected component $scc' \in l'_{SCC}$, $x' \in scc'$, and $\vec{\pi}' \in \delta^*$, $|\mathsf{ss}(x', \vec{\pi}')| \leq \mathsf{S}(scc')$ holds. We have two cases:

Case 1 ($\mathsf{ss}(x_{scc}, \vec{\pi}_{scc}) \subseteq scc$)**.** *Since* $\mathsf{ss}(x_{scc}, \vec{\pi}_{scc}) \subseteq scc$ *then* $|\mathsf{ss}(x_{scc}, \vec{\pi})| \leq |scc|$*. From Definition 13, we know that* $|scc| \leq \mathsf{S}(scc)$*, and accordingly* $|\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})| \leq \mathsf{S}(scc)$*.*

Case 2 ($\mathsf{ss}(x_{scc}, \vec{\pi}_{scc}) \not\subseteq scc$)**.** *Since* $\mathsf{ss}(x_{scc}, \vec{\pi}_{scc}) \not\subseteq scc$*, then there are* $\vec{\pi}_{scc}$*,* π*, and* $\vec{\pi}_{\mathsf{children}}$ *such that: (i)* $\vec{\pi}_{scc} = \vec{\pi}'_{scc} \frown \pi :: \vec{\pi}_{\mathsf{children}}$*, (ii)* $\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc}) \subseteq scc$*, and (iii) letting* $x_{\mathsf{children}} = \mathsf{ex}(\mathsf{ex}(x_{scc}, \vec{\pi}'_{scc}), \pi)$*,* $x_{\mathsf{children}} \in scc_{\mathsf{children}}$ *holds, for some* $scc_{\mathsf{children}} \in \mathsf{children}(scc)$*. Using the same argument as the last case, we have* $|\mathsf{ss}(x_{scc}, \vec{\pi}'_{scc})| \leq |scc|$*.(*) Since* $x_{\mathsf{children}} \in scc_{\mathsf{children}}$*, and from the inductive hypothesis, we have that* $|\mathsf{ss}(x_{\mathsf{children}}, \vec{\pi}_{\mathsf{children}})| \leq \mathsf{S}(scc_{\mathsf{children}})$*. Then using (*) and since* $\mathsf{ss}(x_{\mathsf{children}}, \vec{\pi}_{\mathsf{children}})$ *and* scc *are disjoint, we have* $|\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})| \leq |scc| + \mathsf{S}(scc_{\mathsf{children}})$*. From Definition 13, we have* $|scc| + \mathsf{S}(scc_{\mathsf{children}}) \leq \mathsf{S}(scc)$ *and accordingly* $|\mathsf{ss}(x_{scc}, \vec{\pi}_{scc})| \leq \mathsf{S}(scc)$*.*

□

3.5 Using Structural Knowledge for Better Bounds: Acyclic Dependency

In this section we investigate restricting the type of abstractions to get better compositional bounding properties of different diameter functions. We consider abstractions induced by acyclicity in the variable dependency graph.

Although both, the diameter and the recurrence diameter are not susceptible to compositional bounding as shown in Theorem 2 and Theorem 1, Theorem 3 suggests they can be bounded via

[6] For a list l, $h :: l$ is l with the element h appended to its front.

compositional bounding of the traversal diameter. This guarantees the main utility of compositional bounding: restricting computations to be done using small abstractions, under very general conditions. However, bounds computed using this technique can be very loose, especially for the diameter, and Theorem 4 shows that it impossible to improve them.

Restricting the projections to be induced by acyclic dependency graphs gives rise to a new possibility: upper-bounding the diameter through computing the recurrence diameter of projections. We propose a new way of upper-bounding the diameter of a factored system through computing the recurrence diameters of projections. This has two advantages over just using the aforementioned technique of compositional bounding using the traversal diameter.

(i) We produce polynomials that are substantially tighter than the product expression from Theorem 3.

(ii) The recurrence diameter can be exponentially smaller than the traversal diameter.

On the other hand, computing the recurrence diameter is an NP-hard problem, while computing the traversal diameter can be done in linear time.

3.5.1 Upper-Bounding the Diameter using Abstractions' Recurrence Diameters

Previous attempts to employ the compositional approach to upper-bound the diameter computed an upper bound on the diameter of the system's abstractions and then combined them into an upper bound on the diameter of the concrete system. For instance, Baumgartner et al. (2002) used the recurrence diameters of abstractions of the system at hand. A similar approach was tried by Rintanen and Gretton (2013), where they computed upper-bounds on the size of the state spaces of the abstractions. Two common features of these approaches are:

(i) both approaches used algorithms defined recursively "bottom-up" on acyclic dependency graphs (or equivalently, design netlists)

(ii) the algorithms map every dependency graph to a polynomial, where the terms of the polynomial depend only on the structure of the dependency graph.

In this section we formulate the underlying algorithm of the two previous approaches as a polynomial generator recursively defined on acyclic dependency graphs, which we call $\mathsf{M}_{\mathsf{sum}}$. We then introduce a new polynomial generator, $\mathsf{N}_{\mathsf{sum}}$, that does the recursion in the opposite direction (i.e. "top-down"), and experimentally show that it significantly dominates the bottom-up approach in terms of tightness. We then show that bounds computed by $\mathsf{N}_{\mathsf{sum}}$ are the best any function that is a sound bound on the diameter can compute using abstractions' recurrence diameters and dependencies, given that the dependencies are acyclic.

The Bottom-Up Approach

Before we model the bottom-up calculation, we need to define the concepts of *ancestors* and *leaves*.

Definition 14 (Leaves)**.** *We define the set of leaves* $\mathsf{leaves}(\mathcal{G}_{\mathrm{VS}})$ *to contain those vertices of* $\mathcal{G}_{\mathrm{VS}}$ *from which there are no outgoing edges.*

Definition 15 (Ancestors)**.** *We write* $\mathsf{ancestors}(vs)$ *to denote the set of ancestor vertices of* vs *in* $\mathcal{G}_{\mathrm{VS}}$*. It is the set* $\{vs_0 \mid vs_0 \in \mathcal{G}_{\mathrm{VS}} \wedge vs_0 \rightarrow^+ vs\}$*, where* $\rightarrow^+$ *is the transitive closure of* $\rightarrow$*.*

Definition 16 (Bottom-Up Acyclic Dependency Compositional Bound)**.**

$$\mathsf{M}\langle b\rangle(vs,\delta,\mathcal{G}_{\mathrm{VS}}) = b(\delta\downarrow_{vs}) + (1 + b(\delta\downarrow_{vs})) \sum_{a\in\mathsf{ancestors}(vs)} \mathsf{M}\langle b\rangle(\delta\downarrow_{a})$$

Then, let $\mathsf{M}_{\mathsf{sum}}\langle b\rangle(\delta,\mathcal{G}_{\mathrm{VS}}) = \sum_{vs\in\mathsf{leaves}(\mathcal{G}_{\mathrm{VS}})} \mathsf{M}\langle b\rangle(vs,\delta,\mathcal{G}_{\mathrm{VS}})$.

The functional parameter b is used to bound abstract subproblems, $\mathcal{G}_{\mathrm{VS}}$ is a lifted dependency graph of δ used to identify abstract subproblems, δ is the system of interest. Valid instantiations of b are the recurrence diameter (used in Baumgartner et al. (2002)) and the size of state space $2^{|\mathcal{D}(\delta)|}$ (used in Rintanen and Gretton (2013)). Also, $\mathsf{M}_{\mathsf{sum}}$ is recursively defined on the structure of $\mathcal{G}_{\mathrm{VS}}$, and accordingly it is only well-defined if $\mathcal{G}_{\mathrm{VS}}$ is a DAG. In Baumgartner et al. (2002), it was shown that $\mathsf{M}_{\mathsf{sum}}\langle rd\rangle$ can be used to upper-bound the diameter, in case δ has an acyclic lifted dependency graph. A restatement of Theorem 1 from Baumgartner et al. (2002) is as follows.

Theorem 7. *For any factored representation δ with acyclic lifted dependency graph A_{VS}, $d(\delta) \leq \mathsf{M}_{\mathsf{sum}}\langle rd\rangle(\delta, A_{\mathrm{VS}})$.*

The following example illustrates the operation of $\mathsf{M}_{\mathsf{sum}}$.

Example 8. *Figure 3.8 shows a lifted dependency DAG, A_{VS}, of some factored system δ. Since A_{VS} is a DAG, this implies that for $1 \leq i \leq 4$, the set of variables vs_i is closed under mutual dependency, and $\mathcal{D}(\delta) = \bigcup vs_i$. Given b, and letting b_i be $b(\delta\downarrow_{vs_i})$ and $\mathsf{M}\langle b\rangle(vs_i,\delta,A_{\mathrm{VS}}) = \mathsf{M}_i$, we have*

(i) $\mathsf{M}_1 = b_1$,

(ii) $\mathsf{M}_2 = b_2$,

(iii) $\mathsf{M}_3 = b_3 + \mathsf{M}_1 + b_3\mathsf{M}_1 = b_3 + b_1 + b_1b_3$,

(iv) $\begin{aligned}\mathsf{M}_4 &= b_4 + (1 + b_4)(\mathsf{M}_1 + \mathsf{M}_2 + \mathsf{M}_3)\\ &= 2b_1 + b_2 + b_3 + b_4 + b_1b_3 + 2b_1b_4 + b_2b_4 + b_3b_4 + b_1b_3b_4.\end{aligned}$

Since vs_4 is the only leaf in the dependency graph, the polynomial evalated by $\mathsf{M}_{\mathsf{sum}}$ will be M_4.

The previous example should make it clear that $\mathsf{M}_{\mathsf{sum}}$ can be viewed as a polynomial generating function *recursively* defined on a DAG. The terms of the polynomial $\mathsf{M}_{\mathsf{sum}}$ returns depends *only* on the structure of A_{VS} (i.e. the number of vertices and their connectivity), regardless of δ or the values of b on different projections. However, $\mathsf{M}_{\mathsf{sum}}$ has a problem: it repeatedly adds the terms $\mathsf{M}\langle b\rangle(vs_i)$ as many times as there are children for vs_i. Except for the first $\mathsf{M}\langle b\rangle(vs_i)$ term it adds, all those terms are redundant as we show in the next section. We also note that the function $\mathsf{M}_{\mathsf{sum}}$ is monotonic: for a bounding function b_1 that is an upper bound on another bounding function b_2, $\mathsf{M}_{\mathsf{sum}}\langle b_2\rangle(\delta, A_{\mathrm{VS}}) \leq \mathsf{M}_{\mathsf{sum}}\langle b_1\rangle(\delta, A_{\mathrm{VS}})$ holds.

The Top-Down Approach

In this section we define a new polynomial generator. The motivation in defining this polynomial generator is to avoid redundant terms as highlighted in the previous section. We shall also see that it is easy to formally verify it as an upper bound on the diameter.

Definition 17 (Acyclic Dependency Compositional Bound)**.**

$$\mathsf{N}\langle b\rangle(vs,\delta,\mathcal{G}_{\text{VS}}) = b(\delta\lfloor_{vs})(1+\sum_{c\in\mathsf{children}_{\mathcal{G}_{\text{VS}}}(vs)}\mathsf{N}\langle b\rangle(c,\delta,\mathcal{G}_{\text{VS}}))$$

Then, let $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta,\mathcal{G}_{\text{VS}}) = \sum_{vs\in\mathcal{G}_{\text{VS}}}\mathsf{N}\langle b\rangle(vs,\delta,\mathcal{G}_{\text{VS}})$.

$\mathsf{N}_{\mathsf{sum}}$ is recursively defined on the structure of $\mathcal{G}_{\text{VS}}$, and accordingly it is only well-defined if $\mathcal{G}_{\text{VS}}$ is a DAG. We note that $\mathsf{N}_{\mathsf{sum}}$ is monotonic: for a bounding function b_1 that is an upper bound on another bounding function b_2, $\mathsf{N}_{\mathsf{sum}}\langle b_2\rangle(\delta,A_{\text{VS}}) \leq \mathsf{N}_{\mathsf{sum}}\langle b_1\rangle(\delta,A_{\text{VS}})$ holds, for an acyclic dependency graph A_{VS}.

Theorem 8. *For any factored representation δ with an acyclic lifted dependency graph A_{VS}, $d(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta,A_{\text{VS}})$.*

We postpone the proof of this theorem to the next section. However, we now compare the bounds computed by $\mathsf{N}_{\mathsf{sum}}\langle b\rangle$ with the ones computed using $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$.

Example 9. *Again we refer to A_{VS} from Figure 3.8. Given a topological property b, and letting $b(\delta\lfloor_{vs_i})$ be b_i and $\mathsf{N}\langle b\rangle(vs_i,\delta,A_{\text{VS}}) = \mathsf{N}_i$, we have*

(i) $\mathsf{N}_4 = b_4$,

(ii) $\mathsf{N}_3 = b_3 + b_3b_4$,

(iii) $\mathsf{N}_2 = b_2 + b_2b_4$,

(iv) $\mathsf{N}_1 = b_1 + b_1\mathsf{N}_3 + b_1\mathsf{N}_4 = b_1 + b_1b_3 + b_1b_3b_4 + b_1b_4$, *and the polynomial returned by* $\mathsf{N}_{\mathsf{sum}}$ *is*

(v) $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta,A_{\text{VS}}) = b_1 + b_2 + b_3 + b_4 + b_1b_3 + b_1b_4 + b_2b_4 + b_3b_4 + b_1b_3b_4$.

The value of $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$ has an extra b_1 term and an extra b_1b_4 term, over that of $\mathsf{N}_{\mathsf{sum}}\langle b\rangle$. This is because $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$ counts every ancestor vertex in the lifted dependency graph as many times as the size of its posterity. We found this phenomenon to be highly prevalent in standard benchmarks, where $\mathsf{N}_{\mathsf{sum}}$ substantially outperforms $\mathsf{M}_{\mathsf{sum}}$ in terms of the tightness of the computed bounds. Figure 3.9 shows the computed bounds of $\mathsf{N}_{\mathsf{sum}}\langle b\rangle$ versus $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$ with the function $2^{|\mathcal{D}(\delta)|}-1$ as the base function for a 1030 different Ineternational Planning Competition benchmarks. It shows that $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$ computes looser bounds as it repeats counting the ancestor vertices unnecessarily.

Theoretical Guarantees on $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$ as a Bound on the Diameter

The next theorem shows that for every dependency graph, there is a system where the diameter lower bounds $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$. This, together with Theorem 8 that states that $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$ upper-bounds the diameter, show that the top-down algorithm is tight and cannot be improved.

Theorem 9. *For any $\mathbb{N}$-DAG $A_{\mathbb{N}}$, there is a factored system δ with a lifted dependency DAG, A_{VS}, such that (i) $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta,A_{\text{VS}}) \leq d(\delta)$, and (ii) $A_{\mathbb{N}} = \mathfrak{R}(\!|A_{\text{VS}}|\!)$, where $\mathfrak{R}(vs) = rd(\delta\lfloor_{vs})$.*

We use an "inverted flower" system, defined as follows.

$$\mathbb{I}^i_{l,m} = \begin{cases} \bigcup_{1\leq j<k\leq l}\{(x^i_j,x^i_k),(x^i_k,x^i_j)\} \text{ if } m=1 \\ \bigcup_{2\leq j\leq l+1}\{(x^i_1,x^i_j),(x^i_j,x^i_1)\} \text{ if } m=2 \\ \gamma^i_{m-2}\cup\bigcup_{0\leq j\leq l-1}\{(x^i_{m-1},x^i_{m+j}),(x^i_{m+j},x^i_1)\} \text{ o/w} \end{cases}$$

$\mathbb{I}^i_{l,m}$ is a system whose state space contains a simple path with $m-1$ states and l additional states (petals). Each petal has an outgoing edge to the first state in the path and an incoming edge from the last state in the path. For $\mathbb{I}^i_{l,m}$ we denote the shortest action sequence that joins x^i_a with x^i_b, with $\overrightarrow{\pi}^i_{a\mapsto b}$, for $m \leq a, b \leq m+l-1$. Since for $m \leq a, b \leq m+l-1$ the length of $\overrightarrow{\pi}^i_{a\mapsto b}$ will always be m, we have $d(\mathbb{I}^i_{l,m}) = rd(\mathbb{I}^i_{l,m}) = m$.

Proof. Our proof is constructive. The first step of is to construct an inverted flower relabel for every $u_1 \in V(A_{\mathbb{N}})$, so we denote by $\mathbb{I}^{u_1}$ the inverted flower $\mathbb{I}^{u_1}_{p_{u_1},A_{\mathbb{N}}(u_1)}$. The relabelling inverted flowers are constructed with two properties in mind. Firstly, the recurrence diameter of an inverted flower label of u_1 matches $A_{\mathbb{N}}(u_1)$, the original label of u_1. Secondly, the number of petals, p_{u_1}, in the inverted flower labelling a vertex u_1 must be two more than the actions in all of inverted flowers labelling the children of u_1, i.e. $p_{u_1} = 2 + \sum_{u_2 \in \mathsf{children}_{A_{\mathbb{N}}}(u_1)} | \mathbb{I}^{u_2} |$. Having that many petals is crucial to the proof: we add a different petal as a precondition in every action in the inverted flowers of the children of a vertex u_1. This is to necessitate the execution of $A_{\mathbb{N}}(u_1)$ actions from $\mathbb{I}^{u_1}$ between every action from the inverted lotus associated with every child of u_1, so that the constructed system has the required diameter.

The next step in this construction is to build the required witness system δ. If we follow the same strategy as the ones in the constructions of Theorem 1 or Theorem 2, δ can be formed by the union of all the inverted flowers and then redundant actions with superfluous preconditions can be added to δ, to give the required dependencies. This would guarantee that δ satisfies condition ii. However, δ has to be constructed such that there is an action sequence $\overrightarrow{\pi} \in \delta^*$ that is $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta, A_{\mathrm{VS}})$ actions long, and such that there are two states $x_1 \in \mathbb{U}(\delta)$ and $x_2 = \mathsf{ex}(\overrightarrow{\pi}, x_1)$, with no action sequence shorter than $\overrightarrow{\pi}$ being able to reach x_2 from x_1. To guarantee this, actions with preconditions added (for the purpose of adding dependencies) must replace the original actions, they cannot be redundant. Additionally, those preconditions have to guarantee that in $\overrightarrow{\pi}$, between every pair of actions from an inverted flower associated with a vertex u_1, there needs to be as many action sequences as there are parents to u_1, each of which is as long as the recurrence diameter of the corresponding parent's inverted flower. To achieve that δ and $\overrightarrow{\pi}$ are constructed by Algorithm 3.

Algorithm 3:

$\overrightarrow{\pi} := []$; $\delta := \bigcup_{u_1 \in V(A_{\mathbb{N}})}(\mathbb{I}^{u_1})$
// Loop over $V(A_{\mathbb{N}})$ in reverse topological order
for $u_1 \in l_{\mathbb{N}}$
  $i := A_{\mathbb{N}}(u_1)$; $\mathsf{children} := \bigcup_{u_2 \in \mathsf{children}_{A_{\mathbb{N}}}(u_1)} \mathbb{I}^{u_2}$
  // Loop over every action in $\overrightarrow{\pi}$, in execution order
  for $(p, e) \in \overrightarrow{\pi}$
    // check if the current action belongs to a child of u_1
    if $\mathcal{D}(e) \cap \mathcal{D}(\mathsf{children}) \neq \emptyset$
      // Add preconditions to ensure it needs $\overrightarrow{\pi}_{i\mapsto i+1}$
      $\delta := \delta \setminus \{(p, e)\}$; $\delta := \delta \cup \{(x^{u_1}_{i+1} \uplus p, e)\}$
      // Add $\overrightarrow{\pi}_{i\mapsto i+1}$ before[7] (p, e);
  $\overrightarrow{\pi} := \mathsf{pre}(\overrightarrow{\pi}, (p, e)) \frown \overrightarrow{\pi}^v_{i\mapsto i+1} \frown (p, e) :: \mathsf{suf}(\overrightarrow{\pi}, (p, e))$
    // Increment the petal counter
    $i := i + 1$
  // Add an action sequence to visit one more petal
  $\overrightarrow{\pi} := \overrightarrow{\pi} \frown \overrightarrow{\pi}^{u_1}_{i\mapsto i+1}$

We now show that δ satisfies requirement i. Consider two states, a state at which every inverted flower is at the first petal $x_1 = \biguplus_{u_1 \in V(A_{\mathbb{N}})} x^{u_1}_{A_{\mathbb{N}}(u_1)}$ and a state at which every inverted flower is at the last petal $x_2 = \biguplus_{v \in V(A_{\mathrm{VS}})} x^{u_1}_{A_{\mathbb{N}}(u_1)+p_{u_1}+1}$. By construction, $\mathsf{ex}(x_1, \overrightarrow{\pi}) = x_2$. Since every action in $\overrightarrow{\pi}$ had a different petal added as a precondition from every parent, for any $\overrightarrow{\pi}' \in \delta^*$ that is shorter than $\overrightarrow{\pi}$, $\mathsf{ex}(x_1, \overrightarrow{\pi}') \neq \overrightarrow{\pi}$. Accordingly the diameter of δ is at least $|\overrightarrow{\pi}|$. Since for an inverted flower $\mathbb{I}^{u_1}$, the length of $\overrightarrow{\pi}^{u_1}_{a \mapsto b}$ is $rd(\mathbb{I}^{u_1}) = A_{\mathbb{N}}(u_1)$, then the length of $\overrightarrow{\pi}$ is at least $\mathsf{N}_{\mathsf{sum}}\langle rd \rangle(\delta, A_{\mathrm{VS}})$, and δ satisfies conclusion i.

We now show that δ satisfies requirement ii. Consider a relabelling, A_{VS}, of $A_{\mathbb{N}}$, where every vertex u_1 is relabelled by $\mathcal{D}(\mathbb{I}^{u_1})$, the domain of the inverted flower $\mathbb{I}^{u_1}$ associated with the vertex. Since for a vertex u_1, actions added from $\mathbb{I}^{u_1}$ to δ had preconditions added to them from the petals of inverted flowers labelling all the parents of u_1 and only the parents, edges of A_{VS} represent dependencies of δ. Accordingly A_{vs} is a lifted dependency DAG of δ. Also since $\delta\lfloor_{\mathcal{D}(\mathbb{I}^{u_1})} = \mathbb{I}^{u_1}$, and $\mathcal{G}_{\mathbb{N}}$ is a relabelling of $\mathcal{G}_{\mathrm{VS}}$, then $\mathcal{G}_{\mathbb{N}} = \mathfrak{R}(\!|\mathcal{G}_{\mathrm{VS}}|\!)$. □

Example 10. *For the $\mathbb{N}$-DAG shown in Figure 3.4a, a reverse topological ordering of the vertices in this graph is the list $[u_2; u_3; u_1]$, where leaves u_1 and u_3 come first. We first build an inverted flower system to label each of the vertices (Figures 3.11a-3.11c). As u_2 is a leaf, $p_{u_2} = 0$, so we construct an inverted flower $\mathbb{I}^{u_2}_{2,3}$ with two petals and a path with two states. Concretely, $\mathbb{I}^{u_2}_{2,3} = \{(x^{u_2}_1, x^{u_2}_2), (x^{u_2}_2, x^{u_2}_3), (x^{u_2}_2, x^{u_2}_4), (x^{u_2}_3, x^{u_2}_1), (x^{u_2}_4, x^{u_2}_1)\}$. The states are $x^{u_2}_1 = \{\overline{v_4}, \overline{v_5}\}$, $x^{u_2}_2 = \{\overline{v_4}, v_5\}$, $x^{u_2}_3 = \{v_4, \overline{v_5}\}$ (the first petal), $x^{u_2}_4 = \{v_4, v_5\}$ (the second petal). For u_3 we construct $\mathbb{I}^{u_3}_{2,2} = \{(x^{u_3}_1, x^{u_3}_3), (x^{u_3}_1, x^{u_3}_2), (x^{u_3}_3, x^{u_3}_1), (x^{u_3}_2, x^{u_3}_1)\}$. The states are $x^{u_3}_1 = \{\overline{v_6}, \overline{v_7}\}$, $x^{u_3}_2 = \{\overline{v_6}, v_7\}$ (the first petal), and $x^{u_3}_3 = \{v_6, \overline{v_7}\}$ (the second petal). For the root vertex u_1, $p_{u_1} = 7$, as inverted flowers labelling its children have 5 actions, so $\mathbb{I}^{u_1}_{7,2}$ will have 7 petals. Concretely, $\mathbb{I}^{u_1}_{7,2} = \bigcup\{\{(x^{u_1}_1, x^{u_1}_j), (x^{u_1}_j, x^{u_1}_1)\} \mid 2 \leq j \leq 8\}$. The states are $x^{u_1}_1 = \{\overline{v_1}, \overline{v_2}, \overline{v_3}\}$, $x^{u_1}_2 = \{\overline{v_1}, \overline{v_2}, v_3\}$, $x^{u_1}_3 = \{\overline{v_1}, v_2, \overline{v_3}\}$, $x^{u_1}_4 = \{\overline{v_1}, v_2, v_3\}$, $x^{u_1}_5 = \{v_1, \overline{v_2}, \overline{v_3}\}$, $x^{u_1}_6 = \{v_1, \overline{v_2}, v_3\}$, $x^{u_1}_7 = \{v_1, v_2, \overline{v_3}\}$, and $x^{u_1}_8 = \{v_1, v_2, v_3\}$.*

After constructing the inverted flowers we construct the system δ and the action sequence $\overrightarrow{\pi}$. Based on Algorithm 3, we start from the leaves, and initially $\overrightarrow{\pi} = []$ and $\delta = \mathbb{I}^{u_2}_{2,3} \cup \mathbb{I}^{u_3}_{2,2} \cup \mathbb{I}^{u_1}_{7,2}$. u_2 has no children, i.e. $\mathsf{children} = \emptyset$, so we skip the inner for-loop and concatenate the shortest action sequence that reaches the second petal from the first one in the inverted flower associated with u_2, so $\overrightarrow{\pi} = \overrightarrow{\pi}^{u_2}_{3 \mapsto 4} = [(x^{u_2}_3, x^{u_2}_1); (x^{u_2}_1, x^{u_2}_2); (x^{u_2}_2, x^{u_2}_4)]$. Similarly, since u_3 has no children, we concatenate $\overrightarrow{\pi}^{u_3}_{1 \mapsto 2}$ to $\overrightarrow{\pi}$, so $\overrightarrow{\pi} = \overrightarrow{\pi}^{u_2}_{3 \mapsto 4} \frown \overrightarrow{\pi}^{u_3}_{2 \mapsto 3} = [(x^{u_2}_3, x^{u_2}_1); (x^{u_2}_1, x^{u_2}_2); (x^{u_2}_2, x^{u_2}_4); (x^{u_2}_2, x^{u_3}_1); (x^{u_3}_1, x^{u_3}_3)]$.

For u_1, $\mathsf{children} = \mathbb{I}^{u_2}_{2,3} \cup \mathbb{I}^{u_3}_{2,2}$, which means that the inner for-loop executes, iterating over each action in $\overrightarrow{\pi}$. The first action in $\overrightarrow{\pi}$, $(x^{u_2}_3, x^{u_2}_1)$ comes from the inverted flower labelling u_2, which is a child of u_1. Accordingly, we remove $(x^{u_2}_3, x^{u_2}_1)$ from δ, add to its precondition the second petal $(x^{u_1}_3)$ in the inverted flower labelling u_1, which makes the action $(x^{u_2}_3 \uplus x^{u_1}_3, x^{u_2}_1)$ and add the augmented action to δ. Then we add before that action, the shortest action sequence joining the first and the second petals in the inverted flower of u_1, so $\overrightarrow{\pi} = \overrightarrow{\pi}^{u_1}_{2 \mapsto 3} \frown [(x^{u_1}_3 \uplus x^{u_2}_3, x^{u_2}_1); (x^{u_2}_1, x^{u_2}_2); (x^{u_2}_2, x^{u_2}_4); (x^{u_3}_2, x^{u_3}_1); (x^{u_3}_1, x^{u_3}_3)]$. This is repeated for the remaining actions, so $\overrightarrow{\pi} = \overrightarrow{\pi}^{u_1}_{2 \mapsto 3} \frown [(x^{u_1}_3 \uplus x^{u_2}_3, x^{u_2}_1)] \frown \overrightarrow{\pi}^{u_1}_{3 \mapsto 4} \frown [(x^{u_1}_4 \uplus x^{u_2}_1, x^{u_2}_2)] \frown \overrightarrow{\pi}^{u_1}_{4 \mapsto 5} \frown [(x^{u_1}_5 \uplus x^{u_2}_2, x^{u_2}_4)] \frown \overrightarrow{\pi}^{u_1}_{5 \mapsto 6} \frown [(x^{u_1}_6 \uplus x^{u_3}_2, x^{u_3}_1)] \frown \overrightarrow{\pi}^{u_1}_{6 \mapsto 7} \frown [(x^{u_1}_7 \uplus x^{u_3}_1, x^{u_3}_3)]$. All the modified actions were replaced in δ with themselves, but with the added preconditions. After the inner for loop ends, $\overrightarrow{\pi}^{u_1}_{7 \mapsto 8}$ is concatenated, so $\overrightarrow{\pi} = \overrightarrow{\pi}^{u_1}_{2 \mapsto 3} \frown [(x^{u_1}_3 \uplus$

[7] For lists l, l_1, and l_3, if $l = l_1 \frown a :: l_2$, then $\mathsf{pre}(l, a)$ denotes l_1 and $\mathsf{suf}(l, a)$ denotes l_2.

$x_3^{u_2}, x_1^{u_2})] \frown \overrightarrow{\pi}^{u_1}_{3\mapsto 4} \frown [(x_4^{u_1} \uplus x_1^{u_2}, x_2^{u_2})] \frown \overrightarrow{\pi}^{u_1}_{4\mapsto 5} \frown [(x_5^{u_1} \uplus x_2^{u_2}, x_4^{u_2})] \frown \overrightarrow{\pi}^{u_1}_{5\mapsto 6} \frown [(x_6^{u_1} \uplus x_2^{u_3}, x_1^{u_3})] \frown \overrightarrow{\pi}^{u_1}_{6\mapsto 7} \frown [(x_7^{u_1} \uplus x_1^{u_3}, x_3^{u_3})] \frown \overrightarrow{\pi}^{u_1}_{7\mapsto 8}$.

Consider the states $x_1 = x_2^{u_1} \uplus x_3^{u_2} \uplus x_2^{u_3} = \{\overline{v_1}, \overline{v_2}, v_3, \overline{v_4}, v_5, \overline{v_6}, v_7\}$ *and* $x_2 = x_8^{u_1} \uplus x_4^{u_2} \uplus x_3^{u_3} = \{v_1, v_2, \overline{v_3}, \overline{v_4}, v_5, \overline{v_6}, v_7\}$. $\mathsf{ex}(x_1, \overrightarrow{\pi}) = x_2$*, and the path traversed by* $\overrightarrow{\pi}$ *in* $\mathcal{G}(\delta)$ *is shown in Figure 3.11e, There is not an action sequence in* δ^* *shorter than* $\overrightarrow{\pi}$ *(17) that can reach* x_2 *from* x_1*. Accordingly the diameter of* δ *is at least 17. Letting* $\mathsf{N}(vs) = \mathsf{N}\langle rd\rangle(vs, \delta, A_{\mathrm{VS}})$*, we have* $\mathsf{N}(\{v_4, v_5\}) = rd(\delta\lfloor_{\{v_4,v_5\}}) = rd(\mathbb{I}^{u_2}_{2,3}) = 3$*,* $\mathsf{N}(\{v_4, v_5\}) = rd(\delta\lfloor_{\{v_6,v_7\}}) = rd(\mathbb{I}^{u_3}_{2,2}) = 2$*,* $\mathsf{N}(\{v_1, v_2, v_3\}) = rd(\delta\lfloor_{\{v_1,v_2,v_3\}})(1 + \mathsf{N}(\{v_4, v_5\}) + \mathsf{N}(\{v_6, v_7\})) = 2(1 + 3 + 2) = 12$*, and* $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta, A_{\mathrm{VS}}) = \mathsf{N}(\{v_1, v_2, v_3\}) +$ $\mathsf{N}(\{v_4, v_5\}) + \mathsf{N}(\{v_6, v_7\}) = 12 + 3 + 2 = 17$*, so* $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta, A_{\mathrm{VS}}) \leq d(\delta)$.

The domain of δ *is* $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ *and a partition of it is* $\{\mathcal{D}(\mathbb{I}^{u_1}_{7,2}), \mathcal{D}(\mathbb{I}^{u_2}_{2,3}), \mathcal{D}(\mathbb{I}^{u_3}_{2,2})\} = \{\{v_1, v_2, v_3\}, \{v_4, v_5\}, \{v_6, v_7\}\}$*. A lifted dependency graph of* δ*,* A_{VS}*, has the following labels:* $A_{\mathrm{VS}}(u_1) = \{v_1, v_2, v_3\}$*,* $A_{\mathrm{VS}}(u_2) = \{v_4, v_5\}$*, and* $A_{\mathrm{VS}}(u_3) = \{v_6, v_7\}$ *(shown in Figure 3.11d). It should be clear that* $A_{\mathrm{VS}} = \mathcal{D}(\!|A_\Delta|\!)$.

3.6 The Sublist Diameter

In this section we prove the validity of $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$ as an upper bound on the diameter in the case of an acyclic dependency graph. To do so we define the *sublist diameter*.

Definition 18 (Sublist Diameter)**.** *Recall that a list* l' *is a sublist of* l*, written* $l' \preceq\!\cdot\ l$*, iff all the members of* l' *occur in the same order in* l*. The sublist diameter,* $\ell(\delta)$*, is the length of the longest shortest equivalent sublist to any execution* $\overrightarrow{\pi} \in \delta^*$ *starting at any state* $x \in \mathbb{U}(\delta)$*. Formally,*

$$\ell(\delta) = \max_{x \in \mathbb{U}(\delta)} \max_{\overrightarrow{\pi} \in \delta^*} \min\{|\overrightarrow{\pi}'| \mid \overrightarrow{\pi}' \preceq \overrightarrow{\pi} \wedge \mathsf{ex}(x, \overrightarrow{\pi}) = \mathsf{ex}(x, \overrightarrow{\pi}')\}$$

It should be clear that the sublist diameter is an upper bound on the diameter and a lower bound on the recurrence diameter. We prove that it can be compositionally bounded by $\mathsf{N}_{\mathsf{sum}}\langle\ell\rangle$, Theorem 8 then follows. We also investigate the usage of the sublist diameter to compositionally upper-bound the diameter instead of using the recurence diameter (i.e. using $\mathsf{N}_{\mathsf{sum}}\langle\ell\rangle$ instead of $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$). This can be very advantageous, since we show that the sublist diameter is equal to the diameter in infinitely many systems, where the recurrence diameter is exponentially looser than the diameter (Theorem 10). However, computing the sublist the diameter is at least as hard as computing the recurrence diameter, which is NP-hard (Theorem 12).

3.6.1 Compositionally Bounding the Sublist Diameter Under Acyclic Dependency

Lemma 3. *For any factored system* δ *with an acyclic lifted dependency DAG* A_{VS}*,* $\ell(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle\ell\rangle(\delta, A_{\mathrm{VS}})$.

Proof Summary of Lemma 3. To prove Lemma 3 we use a construction which, given any action sequence $\overrightarrow{\pi} \in \delta^*$ violating the stated bound and a state $s \in \mathbb{U}(\delta)$, produces a sublist, $\overrightarrow{\pi}'$, of $\overrightarrow{\pi}$ satisfying that bound and $\mathsf{ex}(s, \overrightarrow{\pi}) = \mathsf{ex}(s, \overrightarrow{\pi}')$. The proof is by induction on the structure of A_{VS}, where we repeatedly consider every set of variables $P \in A_{\mathrm{VS}}$ and each of its children $C \in \mathsf{children}_{A_{\mathrm{VS}}}(P)$, and for each parent child pair we do the following. We first consider the projected action sequence $\overrightarrow{\pi}\lfloor_C$. From the definition of ℓ, there is a sublist $\overrightarrow{\pi}'_C \preceq\!\cdot\ \overrightarrow{\pi}\lfloor_C$ satisfying $|\overrightarrow{\pi}'_C| \leq \ell(\delta\lfloor_C)$. Moreover, this guarantees that $\overrightarrow{\pi}'_C$ is equivalent, in terms of the execution outcome, to

$\overrightarrow{\pi}\lfloor_C$. Hereupon, if for an action $\pi = (p, e)$, $e \subseteq vs$ is true, we call π a vs-action. The stitching function described in Figure 3.12a is then used to remove the C-actions in $\overrightarrow{\pi}$ whose projections on C are not in $\overrightarrow{\pi}'_C$. Thus our construction arrives at the action sequence $\overrightarrow{\pi}'' = \overrightarrow{\pi}'_C \underset{C}{\wr} \overrightarrow{\pi}$ with at most $\ell(\delta\lfloor_C)$ C-actions. Since $C \in \mathsf{children}_{A_{\mathrm{VS}}}(P)$, then $C \not\to P$ and accordingly, actions with variables from C in their effects never include P variables in their effects. Accordingly, $\mathsf{ex}(s, \overrightarrow{\pi}) = \mathsf{ex}(s, \overrightarrow{\pi}'')$, and in $\overrightarrow{\pi}''$, there will be a list, $\overrightarrow{\pi}_P$, of only P-actions between every consecutive C-actions. However, again, because of how ℓ is defined, there is a $\overrightarrow{\pi}'_P \preceq \cdot\ \overrightarrow{\pi}_P$, whose execution outcome is the same as $\overrightarrow{\pi}_P$ and whose length is no more than $\ell(\delta\lfloor_P)$. Repeating this for all sequences of only P-actions completes our construction. □

The above construction and the usage of the stitching function is illustrated in the following example.

Example 11. *Consider the following factored system, whose dependency graph is shown in Figure 3.12b.*

$$\left\{ \begin{array}{l} \pi_1 = (\emptyset, \{v_3\}), \pi_2 = (\{v_3\}, \{v_4\}), \pi_3 = (\{v_3\}, \{\overline{v_1}\}), \pi_4 = (\{v_3\}, \{v_2\}), \\ \pi_5 = (\{v_4\}, \{v_1\}), \pi_6 = (\{v_2, v_4\}, \{v_5\}), \pi_7 = (\{\overline{v_3}\}, \{v_4, v_5\}) \end{array} \right\}$$

$\mathcal{D}(\delta) = c \cup p$, where $c = \{v_1, v_4, v_5\}$ are called the "child" variables, and $p = \{v_2, v_3\}$, and $c \not\to p$ are called the "parent" variables. In δ, the actions π_2, π_3, π_5, π_6, π_7 are c-actions, and π_1, π_4 are p-actions. An action sequence $\overrightarrow{\pi} \in \delta^*$ is $[\pi_1; \pi_1; \pi_2; \pi_3; \pi_4; \pi_4; \pi_5; \pi_6]$ that reaches the state $\{v_1, v_2, v_3, v_4, v_5\}$ from $\{v_1, \overline{v_2}, \overline{v_3}, \overline{v_4}, \overline{v_5}\}$. When $\overrightarrow{\pi}$ is projected on c it becomes $[\pi_2\lfloor_c; \pi_3\lfloor_c; \pi_5\lfloor_c; \pi_6\lfloor_c]$, which is in $({}^*\delta\lfloor_c)$. A shorter action sequence, $\overrightarrow{\pi}_c$, achieving the same result as $\overrightarrow{\pi}\lfloor_c$ is $[\pi_2\lfloor_c; \pi_6\lfloor_c]$. Since $\overrightarrow{\pi}_c$ is a sublist of $\overrightarrow{\pi}\lfloor_c$, we can use the stitching function to obtain a shorter action sequence in δ^* that reaches the same state as $\overrightarrow{\pi}$. In this case, $\overrightarrow{\pi}_c \underset{c}{\wr} \overrightarrow{\pi}$ is $[\pi_1; \pi_1; \pi_2; \pi_4; \pi_4; \pi_6]$. The second step is to contract the pure p segments which are $[\pi_1; \pi_1]$ and $[\pi_4; \pi_4]$, which are contracted to $[\pi_1]$ and $[\pi_4]$, respectively. The final constructed action sequence is $[\pi_1; \pi_2; \pi_4; \pi_6]$, which achieves the same state as $\overrightarrow{\pi}$.

Theorem 8 follows directly from Lemma 3 and the fact that the sublist diameter is both: an upper bound on the diameter and a lower bound on the recurrence diameter.

Also because the recurrence dimater upper bounds the sublist diameter, and the sublist diameter upper-bounds the diameter, and since $\mathsf{N}_{\mathsf{sum}}$ is monotonic, the following lemma can be derived from Theorem 9 and Lemma 3.

Lemma 4. *For any $\mathbb{N}$-DAG (a DAG whose vertices are labelled with numbers), $A_{\mathbb{N}}$, there is a δ that has a lifted dependency DAG, A_{VS}, such that (i) $D_{\mathbb{N}} = \mathfrak{L}(\!|A_{\mathrm{VS}}|\!)$, where $\mathfrak{L}(vs) = \ell(\delta\lfloor_{vs})$ (i.e. $D_{\mathbb{N}}$ is a relabelling of A_{VS}, where every vertex labelled by vs in A_{VS}, is relabelled by the sublist diameter of the projection $\delta\lfloor_{vs}$), and (ii) $\ell(\delta) = \mathsf{N}_{\mathsf{sum}}\langle\ell\rangle(\delta, A_{\mathrm{VS}})$.*

3.6.2 The Sublist Diameter as a Compositional Upper Bound on the Diameter

Initially we defined the sublist diameter as a tool for proving Theorem 8. However, Lemma 3 shows that we can use the sublist diameter (instead of the recurrence diameter) as a base case function with the top-down approach to upper-bound the diameter. In this section we compare the sublist diameter to the recurrence diameter as a base case function for upper-bounding the diameter. We compare two aspects: their tightness as bounds, and the hardness of computing each of them.

Tightness of the Sublist Diameter

In this section we show that in addition to being a lower bound on the recurrence diameter, the sublist diameter can be exponentially smaller than the recurrence diameter. To do that, we exploit a rather interesting insight: the value of sublist diameter depends on the factored representation: two factored representations of the same state space can have different sublist diameters. Indeed minimising (or *factoring*) the size of a factored representation (i.e. the number of actions) used to represent the same system can minimise the sublist diameter. This is in contrast to the diameter, the recurrence diameter and the traversal diameter, which are agnostic to the factored representation: they only depend on the state space of the system. The following example demonstrates using factoring to reduce the sublist diameter.

Example 12. *Consider* $\delta\lfloor_{\{v_1,v_2\}}$ *from Example 2 and* $\delta_1 = \{k_1\lfloor_{\{v_1,v_2\}}, k_2\lfloor_{\{v_1,v_2\}}, k_3\lfloor_{\{v_1,v_2\}}, k_4\lfloor_{\{v_1,v_2\}}\}$. δ_1 *is a factoring of* $\delta\lfloor_{\{v_1,v_2\}}$ *and accordingly,* $\mathcal{G}(\delta_1)$ *and* $\mathcal{G}(\delta\lfloor_{\{v_1,v_2\}})$ *are identical. Accordingly* $d(\delta_1) = d(\delta\lfloor_{\{v_1,v_2\}}) = 1$*, and* $rd(\delta_1) = rd(\delta\lfloor_{\{v_1,v_2\}}) = 3$*.* $\ell(\delta_1) = 1$*, because for any non empty action sequence* $\vec{\pi} \in {\delta_1}^*$*, the last action* π *in* $\vec{\pi}$ *reaches the same state as* $\vec{\pi}$*, and* $[\pi] \preceq \vec{\pi}$*. In contrast,* $\ell(\delta\lfloor_{\{v_1,v_2\}}) = 3$*, since no shorter sublist of the action sequence* $[p_1\lfloor_{\{v_1,v_2\}}; p_2\lfloor_{\{v_1,v_2\}}; p_3\lfloor_{\{v_1,v_2\}}]$ *starts at* $\{\overline{v_1}, \overline{v_2}\}$ *and results in* $\{v_1, v_2\}$.

We now formally demonstrate the utility of factoring to reduce the sublist diameter.

Definition 19 (Factoring)**.** *Factored representation* δ_1 *is a factoring of representation* δ_2 *iff all the following is true:*

(i) $\mathcal{D}(\delta_1) \subseteq \mathcal{D}(\delta_2)$

(ii) $|\delta_1| \leq |\delta_2|$

(iii) $\forall \pi_2 \in \delta_2, x \in \mathbb{U}(\delta_2).\exists \pi_1 \in \delta_1.\ \mathsf{ex}(x, \pi_1) = \mathsf{ex}(x, \pi_2)$

(iv) there is a function $f : \delta_1 \Rightarrow 2^{\delta_2}$*, where*

(a) $\forall (p_1, e_1) \in \delta_1, (p_2, e_2) \in f(p_1, e_1).p_1 \subseteq p_2$*,*

(b) $\forall \pi_1 \in \delta_1, x \in \mathbb{U}(\delta_2).\exists \pi_2 \in f(\pi_1).\mathsf{ex}(x, \pi_1) = \mathsf{ex}(x, \pi_2)$*, and*

(c) $\forall \pi_1 \in \delta_1.f(\pi_1) \neq \emptyset$

Proposition 4. *If* δ_1 *is a factoring of* δ_2 *then the largest connected components of* $\mathcal{G}(\delta_1)$ *and* $\mathcal{G}(\delta_2)$ *are isomorphic.*

Corollary 1. *If* δ_1 *is a factoring of* δ_2 *then* $d(\delta_1) = d(\delta_2)$ *and* $rd(\delta_1) = rd(\delta_2)$.

Lemma 5. *There are infinitely many factored systems that have factorings with exponentially smaller sublist diameters, that are equal to the diameter.*

Before we begin our proof, let $K_n^i = \{(\emptyset, x_j^i) \mid 1 \leq j \leq n+1\}$, i.e. a factored system with the only non-singleton component of its state space being a clique with $n+1$ states.

Proof. We show that for every $n \in \mathbb{N}$ there is a system whose sublist diameter can be reduced from n to 1 through factoring. Consider the factored representations $K = K_n^1$, and $\gamma = \gamma_n^1$. $\ell(K) = 1$ and $\ell(\gamma) = n$ hold. Note that since in the definition of K and γ we used the same superscript, $\mathcal{D}(K) = \mathcal{D}(\gamma) = vs$.(*) Accordingly the state space of $K \cup \gamma$ will have only one non-singleton component that is a clique of order $n+1$. However, although $K \cup \gamma$ is a factored system whose

state space is a clique, its sublist diameter will be n, because there is not a shorter sublist of the action sequence $[(x_1^1, x_2^1); (x_2^1, x_3^1); ..; (x_n^1, x_{n+1}^1)]$ that can reach the state x_{n+1}^1, starting from the state x_1^1. Finally, from (*) and since γ does not add any more edges in the state space of K, then K is a factoring of $K \cup \gamma$, which finishes our proof. □

Theorem 10. *There are infinitely many factored systems whose sublist diameter is (i) equal to their diameter, and (ii) exponentially smaller (in the number of state variables) than their recurrence diameter. There are also infinitely many factored systems whose sublist diameter is (i) exponentially larger (in the number of state variables) than their diameter, and (ii) equal to their recurrence diameter.*

This theorem follows immediately from Corollary 1 and Lemma 5. Furthermore we show that factoring a representation can only reduce the value of the sublist diameter.

Theorem 11. *If δ_1 is a factoring of δ_2 then $\ell(\delta_1) \leq \ell(\delta_2)$.*

Proof. We prove the theorem by showing that for any $x \in \mathbb{U}(\delta_1)$ and $\overrightarrow{\pi} \in {\delta_1}^*$ there is a sublist of $\overrightarrow{\pi}$, $\overrightarrow{\pi}'$, such that $\mathsf{ex}(s, \overrightarrow{\pi}') = \mathsf{ex}(s, \overrightarrow{\pi})$ and $|\overrightarrow{\pi}'| \leq \ell(\delta_2)$. Using the mapping f from Definition 19, there must be an action sequence $\overrightarrow{\pi}_2 \in {\delta_2}^*$, such that $\mathsf{ex}(x, \overrightarrow{\pi}_2) = \mathsf{ex}(x, \overrightarrow{\pi})$. From Definition 18 we then obtain a sublist, $\overrightarrow{\pi}_2'$, of $\overrightarrow{\pi}_2$ whose length is bounded by $\ell(\delta_2)$ and, such that $\mathsf{ex}(x, \overrightarrow{\pi}_2') = \mathsf{ex}(x, \overrightarrow{\pi}_2)$. We reverse the mapping of actions to transform $\overrightarrow{\pi}_2'$ into a sublist $\overrightarrow{\pi}'$ of $\overrightarrow{\pi}$. From Definition 19, $\mathsf{ex}(x, \overrightarrow{\pi}) = \mathsf{ex}(x, \overrightarrow{\pi}')$, but $|\overrightarrow{\pi}'| = |\overrightarrow{\pi}_2'| \leq \ell(\delta_2)$, which finishes our proof. □

The Complexity of Computing the Sublist Diameter

Theorem 12. *Computing the Sublist Diameter is NP-hard.*

Proof. We prove this theorem by showing that for any digraph $\mathcal{G}$, one can compute the length of the longest path in $\mathcal{G}$ by computing the sublist diameter of a system, δ, that can be constructed in polynomial time from $\mathcal{G}$. Let $\mathcal{D} = \{v_1, v_2, .., v_{\lceil \log_2 |V(\mathcal{G})| \rceil}\}$ and let $2^{\mathcal{D}}$ denote the set of all states representable by $\mathcal{D}$. Since $|V(\mathcal{G})| \leq 2^{|\mathcal{D}|}$, then there is an injective function $x : V(\mathcal{G}) \Rightarrow 2^{\mathcal{D}}$. Consider the system $\delta = \{(x(u_1), x(u_2)) \mid (u_1, u_2) \in E(\mathcal{G})\}$. It should be clear that the largest connected component in $\mathcal{G}(\delta)$ is isomorphic to $\mathcal{G}$ and accordingly the length of the longest path in $\mathcal{G}$ is the same as the length of the longest path in $\mathcal{G}(\delta)$ which is equal to $rd(\delta)$. Now observe that for every $(p, e) \in \delta$, the precondition and the effect are both complete assignments of $\mathcal{D}$, i.e. $p \in \mathbb{U}(\delta)$ and $e \in \mathbb{U}(\delta)$. Accordingly every action in δ represents exactly one edge from $\mathcal{G}$. From that we have $rd(\delta) = \ell(\delta)$, which finishes our proof. □

3.7 Exploiting State Space Acyclicity

The practical utility of dependency graph based decompositions (like $\mathsf{N}_{\mathsf{sum}}$) provides a good motivation to pursue other structures, like state space acyclicity. State space acyclicity is independent of acyclicity in variable dependency. Thus, methods previously developed cannot be used to exploit the former in compositional upper-bounding. We demonstrate this using the well-studied hotel key protocol as a case-study.

3.7.1 Hotel Key Protocol

We now consider the hotel key protocol from Jackson (2006). Reasoning about safe and unsafe versions of this protocol is challenging for state-of-the-art AI planners and model-checkers. For example, a version of the protocol was shown unsafe for an instance with 1 room, 2 guests and 4 keys using a counterexample generator in Blanchette and Nipkow (2010). The problem becomes more challenging for the safe version of the protocol, where the only feasible approach is using interactive theorem provers, as in Nipkow (2006).

We describe the factored transition system corresponding to that protocol. The system models a hotel with R rooms, G guests, and K keys per room, which guests can use to enter rooms (Figure 3.13 shows an example with $R = 2$, $G = 2$ and $K = 3$). The state characterising propositions are: (i) $\texttt{lk}_{r,k}$, reception last issued key k for room r, for $0 < r \leq R$ and $(r-1)K < k \leq rK$; (ii) $\texttt{ck}_{r,k}$, room r can be accessed using key k, for $0 < r \leq R$ and $(r-1)k < k \leq rK$; (iii) $\texttt{gk}_{g,k}$, guest g has key k, for $0 < g \leq G$, $0 < k \leq RK$; and (iv) $\texttt{s}_r$, is an auxiliary variable that means that room r is "safely" delivered to some guest. The protocol actions are as follows: (i) guest g can check-in to room r, receiving key k—$(\{\texttt{lk}_{r,k_1}\}, \{\texttt{gk}_{g,k_2}, \texttt{lk}_{r,k_2}, \overline{\texttt{lk}_{r,k_1}}, \overline{\texttt{s}_r}\})$; and (ii) where room r was previously entered using key k, guest g can enter room r using key k'—$(\{\texttt{gk}_{g,k'}, \texttt{lk}_{r,k}\}, \{\texttt{ck}_{r,k'}, \overline{\texttt{ck}_{r,k}}, \texttt{s}_r\})$. Thus, guests can retain keys indefinitely, and there is no direct communication between rooms and reception.

For completeness, we note that this protocol was formulated in the context of checking safety properties. Safety is violated only if a guest enters a room occupied by another guest. Formally, the safety of this protocol is checked by querying if there exists a room r, guest g and keys $k \neq k'$, so that $\texttt{lk}_{r,k'} \wedge \texttt{ck}_{r,k} \wedge \texttt{gk}_{g,k'} \wedge \texttt{s}_r$. The initial state asserts that guests possess no keys, and the reception issued the first key for each room, and each room opens with its first key. Formally, this is represented by asserting $\texttt{lk}_{r,(r-1)K} \wedge \texttt{ck}_{r,(r-1)K}$ is true for $1 \leq r \leq R, (r-1)K < k \leq rK$, and that all other state variables are false.

We adopt some shorthand notations in order to provide examples of concepts in terms of the hotel key protocol. A variable name is written in upper case to refer to a particular assignment, where the only variable that is true is given by the indices. For example, the assignment $\{\overline{\texttt{ck}_{1,1}}, \texttt{ck}_{1,2}, \overline{\texttt{ck}_{1,3}}\}$—indicating room 1 can be accessed using key 2—is indicated by writing $\texttt{CK}_{1,2}$. We refer to sets of variables by omitting an index term. For example, $\texttt{lk}_1$ indicates the variables $\{\texttt{lk}_{1,i} \mid 1 \leq i \leq 3\}$. The following examples illustrate the concepts of projection and state variable dependency in the context of the hotel key protocol.

Example 13. *Consider the set of variables* ROOM1 $\equiv lk_1 \cup ck_1 \cup \{gk_{1,2}, gk_{1,3}, gk_{2,2}, gk_{2,3}\}$. *The variables* ROOM1 *model system state relevant to the 1st hotel room. Figure 3.13c shows the projected system* $\delta\lfloor_{\text{ROOM1}}$.

Example 14. *Figure 3.13b shows a dependency graph associated with the system from Figure 3.13a. Let* ROOM2 $\equiv lk_2 \cup ck_2 \cup \{gk_{1,5}, gk_{1,6}, gk_{2,5}, gk_{2,6}\}$. *Figure 3.13b depicts two connected components induced by the sets* ROOM1 *and* ROOM2*, respectively. One lifted dependency graph would have exactly two unconnected vertices, one being a contraction of the vertices from* ROOM1*, and the other a contraction of those from* ROOM2*. Due to the disconnected structure of the dependency graph, intuitively the sum of bounds for* $\delta\lfloor_{\text{ROOM1}}$ *and* $\delta\lfloor_{\text{ROOM2}}$ *can be used to upper-bound the diameter of the concrete system.*

3.7.2 State Space Acyclicity Compositional Bounding Constructs

To exploit state space acyclicity we formalise it as follows.

Definition 20 (Acyclic Transition System)**.** *δ is acyclic iff $\forall x, x' \in \mathbb{U}(\delta)$. $x \neq x'$ then $x \not\rightsquigarrow x'$ or $x' \not\rightsquigarrow x$.*

In the next example we show that state space acyclicity is independent of acyclicity in variable dependency, and thus $\mathsf{N}_{\mathsf{sum}}$ or other methods cannot be used to exploit state space acyclicity for compositional upper-bounding.

Example 15. *$\delta\lfloor_{\subset k_1}$ is acyclic. For example, no state satisfying $\subset K_{1,2}$ can be reached from a state satisfying $\subset K_{1,3}$. Now consider $\delta\lfloor_{\text{ROOM1}}$ from Example 13. The dependency graph of $\delta\lfloor_{\text{ROOM1}}$ is comprised of one strongly connected component (SCC). Thus, acyclicity in the assignments of $\subset k_1$ cannot be exploited in $\delta\lfloor_{\text{ROOM1}}$ by analysing its dependency graph.*

To be able to exploit state space acyclicity, we now introduce a new abstraction concept: *snapshot*. A snapshot models the system when we fix the assignment to a subset of the state variables, removing actions whose preconditions *or* effects contradict that assignment.

Definition 21 (Snapshot)**.** *We write $|X|$ to denote the cardinality of the set X. For states x and x', let $\mathsf{agree}(x, x')$ denote $|\mathcal{D}(x) \cap \mathcal{D}(x')| = |x \cap x'|$, i.e. a variable that is in the domains of both x and x' has the same assignment in x and x'. For δ and a state x, the snapshot of δ at x is*

$$\delta\restriction_x \equiv \{(p, e) \mid (p, e) \in \delta \wedge \mathsf{agree}(p, x) \wedge \mathsf{agree}(e, x)\}\lfloor_{\mathcal{D}(\delta)\setminus\mathcal{D}(x)}$$

Example 16. *$\delta\lfloor_{\text{ROOM1}}\restriction_{\subset K_{1,2}}$ is shown in Figure 3.13d.*

We now investigate how such acyclicity can be used for bounding. First, let $\Sigma(x) = \delta\restriction_x$, i.e. it is a function that maps a state x to the snapshot of δ on that state. Consider a system δ where for some variables vs we have that $\delta\lfloor_{vs}$ is acyclic – i.e. the state space of $\delta\lfloor_{vs}$ forms a directed acyclic graph. In that case, we define the following version of $\mathsf{S}_{\mathsf{max}}$ that operates on acyclic abstraction state spaces.

Definition 22 (Acyclic System Compositional Bound)**.** *For an arbitrary bounding function b and for a state $x \in \mathbb{U}(\delta\lfloor_{vs})$,*

$$\mathsf{S}\langle b\rangle(x, vs, \delta) = \mathsf{S}\langle b\rangle(x, \Sigma(\!|\mathcal{G}(\delta\lfloor_{vs})|\!))$$

Then, let $\mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta) = \max_{x \in \mathbb{U}(\delta\lfloor_{vs})} \mathsf{S}\langle b\rangle(x, vs, \delta)$.

This function returns the longest path in the state space of the projection $\delta\lfloor_{vs}$ weighted by the diameters of the snapshots of δ on different states in $\mathbb{U}(\delta)$. To make this concrete, consider the following example.

Example 17. *Since $\delta\lfloor_{\subset k_1}$ is acyclic, and $\subset K_{1,i} \in \mathbb{U}(\delta\lfloor_{\subset k_1})$, then $\mathsf{S}\langle d\rangle(\subset K_{1,i}, \subset k_1, \delta)$ is well-defined, for $i \in \{1, 2, 3\}$. Denoting $d(\delta\restriction_{\subset K_{1,i}})$ with $d_{1,i}$ and $\mathsf{S}\langle d\rangle(\subset K_{1,i}, \subset k_1, \delta)$ with $\mathsf{S}_{1,i}$, we have $\mathsf{S}_{1,3} = d_{1,3}$ because $\mathsf{succ}(\subset K_{1,3}, \delta\lfloor_{\subset k_1}) = \emptyset$. We also have $\mathsf{S}_{1,2} = d_{1,2}+1+S_{1,3} = d_{1,2}+1+d_{1,3}$ and $\mathsf{S}_{1,1} = d_{1,1} + 1 + S_{1,2} = d_{1,1} + 1 + d_{1,2} + 1 + d_{1,3} = d_{1,1} + d_{1,2} + d_{1,3} + 2$.*

Theorem 13. *If $\delta\lfloor_{vs}$ is acyclic and b bounds d, then $d(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.*

Proposition 5. *If $\delta\lfloor_{vs}$ is acyclic and $x \in \mathbb{U}(\delta\lfloor_{vs})$, if $x' \in \mathsf{succ}(x, \delta\lfloor_{vs})$, then $b(\delta\restriction_x) + 1 + \mathsf{S}\langle b\rangle(x', vs, \delta) \leq \mathsf{S}\langle b\rangle(x, vs, \delta)$, for a base case function b.*

Definition 23 (Subsystem Trace)**.** *For a state x, action sequence $\overrightarrow{\pi}$, and set of variables vs, let $\partial(x, \overrightarrow{\pi}, vs)$ be:*

$$\begin{aligned}\partial(x, [], vs) &= [] \\ \partial(x, \pi :: \overrightarrow{\pi}, vs) &= \begin{cases} x' :: \partial(x', \overrightarrow{\pi}, vs) & \text{if } x\lfloor_{vs} \neq x'\lfloor_{vs} \\ \partial(x', \overrightarrow{\pi}, vs) & \text{otherwise} \end{cases}\end{aligned}$$

where $x' = \mathsf{ex}(x, \pi)$.

Proposition 6. *For any* x, $\vec{\pi}$, *and* vs, *if* $\partial(x, \vec{\pi}, vs) = []$ *then: (i)* $x\downarrow_{vs} = \mathsf{ex}(x, \vec{\pi})\downarrow_{vs}$ *and (ii) there is* $\vec{\pi}'$ *where* $\mathsf{ex}(x, \vec{\pi}) = \mathsf{ex}(x, \vec{\pi}')$ *and* $|\vec{\pi}'| \leq d(\delta\downarrow_{x\downarrow_{vs}})$.

Proposition 7. *For two states* x *and* x', *a sequence of states* $\vec{x}$, *a set of variables* vs, *and an action sequence* $\vec{\pi}$, *if* $\partial(x, \vec{\pi}, vs) = x' :: \vec{x}$, *then there are* $\vec{\pi}_1$, π *and* $\vec{\pi}_2$ *such that (i)* $\vec{\pi} = \vec{\pi}_1 \frown \pi :: \vec{\pi}_2$, *(ii)* $\partial(x, \vec{\pi}_1, vs) = []$, *(iii)* $\mathsf{ex}(\mathsf{ex}(x, \vec{\pi}_1), \pi) = x'$, *and (iv)* $\mathsf{ex}(x', \vec{\pi}_2) = \mathsf{ex}(x, \vec{\pi})$.

Proposition 8. *For any* x, $\vec{\pi}_1$, $\vec{\pi}_2$, *and* vs, *we have that* $\partial(x, \vec{\pi}_1 \frown \vec{\pi}_2, vs) = \partial(x, \vec{\pi}_1, vs) \frown \partial(\mathsf{ex}(x, \vec{\pi}_1), \vec{\pi}_2, vs)$.

Lemma 6. *For any* δ *and* vs *where* $\delta\downarrow_{vs}$ *is acyclic,* $s \in \mathbb{U}(\delta)$, *and* $\vec{\pi} \in \delta^*$, *there is* $\vec{\pi}'$ *such that* $\mathsf{ex}(s, \vec{\pi}) = \mathsf{ex}(s, \vec{\pi}')$ *and* $|\vec{\pi}'| \leq \mathsf{S}\langle d\rangle(s\downarrow_{vs}, vs, \delta)$.[8]

Proof. The proof is by induction on $\partial(s, \vec{\pi})$. The base case, where, $\partial(s, \vec{\pi}) = []$, is trivial. In the step case we have that $\partial(s, \vec{\pi}) = s' :: \vec{x}$ and the induction hypothesis: for any $s^* \in \mathbb{U}(\delta)$, and $\vec{\pi}^* \in \delta^*$ if $\partial(s^*, \vec{\pi}^*) = \vec{x}$ then there is $\vec{\pi}^{*\prime}$ where $\mathsf{ex}(s^*, \vec{\pi}^*) = \mathsf{ex}(s^*, \vec{\pi}^{*\prime})$ and $|\vec{\pi}^{*\prime}| \leq \mathsf{S}\langle d\rangle(s^*\downarrow_{vs})$.

Since $\partial(s, \vec{\pi}) = s' :: \vec{x}$, we have $\vec{\pi}_1, \pi$ and $\vec{\pi}_2$ satisfying the conclusions of Proposition 7. Based on conclusion i, ii, and iii of Proposition 7 and Proposition 8 we have $\partial(s', \vec{\pi}_2) = \vec{x}$. Accordingly, letting s^*, and $\vec{\pi}^*$ from the inductive hypothesis be s', and $\vec{\pi}_2$, respectively, there is $\vec{\pi}'_2$ such that $\mathsf{ex}(s', \vec{\pi}_2) = \mathsf{ex}(s, \vec{\pi}'_2)$ and $|\vec{\pi}'_2| \leq \mathsf{S}\langle d\rangle(s'\downarrow_{vs})$.†

From conclusion ii of Proposition 7 and conclusion ii of Proposition 6 there is $\vec{\pi}'_1$ where $\mathsf{ex}(s, \vec{\pi}_1) = \mathsf{ex}(s, \vec{\pi}'_1)$ and $|\vec{\pi}'_1| \leq d(\delta\downarrow_{s\downarrow_{vs}})$. Letting $\vec{\pi}' = \vec{\pi}'_1 \# \pi :: \vec{\pi}'_2$, from conclusions iii and iv of Proposition 7 and † we have $\mathsf{ex}(s, \vec{\pi}) = \mathsf{ex}(s, \vec{\pi}')$ and $|\vec{\pi}'| \leq d(\delta\downarrow_{s\downarrow_{vs}}) + 1 + \mathsf{S}\langle d\rangle(s'\downarrow_{vs})$.‡

Lastly, from conclusion i of Proposition 6 and conclusion ii of Proposition 7 we have $s\downarrow_{vs} = \mathsf{ex}(s, \vec{\pi}_1)\downarrow_{vs} = \mathsf{ex}(s, \vec{\pi}'_1)\downarrow_{vs}$ and accordingly $\mathsf{ex}(s\downarrow_{vs}, \pi\downarrow_{vs}) = s'\downarrow_{vs}$. Based on that we have $s'\downarrow_{vs} \in \mathsf{children}_{\delta\downarrow_{vs}}(s\downarrow_{vs})$. Then from Proposition 5 and ‡ we have $|\vec{\pi}'| \leq \mathsf{S}\langle d\rangle(s\downarrow_{vs})$. □

Theorem 13 follows from Lemma 6 and Definitions 6 and 13.

In closing, it is worth noting that for the above example, the dependency graph of $\delta\downarrow_{\text{ROOM1}}$ is comprised of one SCC. Therefore there is no lifted dependency graph providing further decomposition. Indeed, exploiting acyclicity in dependency between variables alone, one cannot further decompose the subproblem $\delta\downarrow_{\text{ROOM1}}$. We were able to achieve a more fine grained decomposition of that component above, by exploiting state space acyclicity.

3.8 A Practical Algorithm for Upper-Bounding

Theorem 13 suggests the possibility of compositional upper-bounding of the diameter given the presence of acyclicity in a transition system's state space. In this section we investigate building a practical compositional algorithm to upper-bound the diameter based on Theorem 13. One straightforward approach is given by Algorithm 4.

In PUR, Ω is an oracle that returns a set of strict subsets of $\mathcal{D}(\delta)$, where $\forall vs \in \Omega(\delta).\delta\downarrow_{vs}$ is acyclic. PUR terminates because a snapshot has fewer variables than the concrete system. In PUR the function UPBND provides upper bounds for the diameters of "base-case" problems – i.e.

[8] In the rest of this proof we omit the vs and/or δ arguments from $\partial(,,)$ and S as they do not change.

Algorithm 4: PUR(δ)

$S = \mathsf{min}(\{\mathsf{S}_{\mathsf{max}}\langle\text{PUR}\rangle(vs, \delta) \mid vs \in \Omega(\delta)\} \cup \infty)$
if $S = \infty$ **return** UPBND(δ) **else return** S

problems that are not further decomposed. Given this assumption and Theorem 13, PUR itself computes valid upper bounds for the diameter of the whole problem.

A main question for a practical implementation of PUR is the choice of Ω. The trivial choice of all strict subsets of $\mathcal{D}(\delta)$ is impractical. A pragmatic solution which we have adopted, is to take the situation that elements in $\mathcal{D}(\delta)$ model individual assignments in the SAS+ model generated using Fast-Downward's preprocessing step Helmert (2006a). Each element in $\Omega(\delta)$ then corresponds to a set of elements from $\mathcal{D}(\delta)$ that model one multi-valued state variable whose domain transition graph is acyclic.

A source of intractability in PUR comes from the min operator. For a full evaluation, $\mathsf{S}_{\mathsf{max}}$ is recursively called as many as $|\Omega(\delta)|!$ times. In practice we only evaluate $\mathsf{S}_{\mathsf{max}}$ on one *arbitrarily* chosen element from $\Omega(\delta)$. Our experimentation never uncovered a problem where a full evaluation of the min, where computationally feasible, produced a better bound. A second source of computational expense comes from the definition of $\mathsf{S}_{\mathsf{max}}$: PUR can be recursively called a number of times that is linear in the size of the state space of δ. This happens if $\Omega(\delta)$ is a partition of $\mathcal{D}(\delta)$. Although this worst case scenario is contrived, in practice $\Omega(\delta)$ can cover sufficient elements from $\mathcal{D}(\delta)$ to render PUR impractical. This is demonstrated in the following example.

Example 18. *Taking $\mathcal{D}(\delta)$ associated with the hotel key protocol example, the Fast-Downward preprocessor Helmert (2006a) identifies partition:*

$$\left\{\begin{array}{l} ck_1, ck_2, lk_1, lk_2, \\ \{gk_{1,2}\}, \{gk_{1,3}\}, \{gk_{1,5}\}, \{gk_{1,6}\}, \\ \{gk_{2,2}\}, \{gk_{2,3}\}, \{gk_{2,5}\}, \{gk_{2,6}\}, \\ \{s_1\}, \{s_2\} \end{array}\right\}$$

as SAS+ variable assignments. Let $\Omega(\delta)$ denote that set, excluding $\{s_1\}$ and $\{s_2\}$. Note, $\forall vs \in \Omega(\delta)$ we have that $\delta\downarrow_{vs}$ is acyclic. Consequently, we have that PUR(δ) *evaluates after $\prod_{vs\in\Omega(\delta)}|vs|$,i.e. $3^4 = 81$, calls to $\mathsf{S}_{\mathsf{max}}$.*

3.8.1 Hybrid Algorithm

We have just observed a situation where PUR can exhibit a runtime that is linear in the size of the state space. That is favourable compared to exact calculations of diameter, which in our opening remarks we noted to have worse-than-quadratic runtime. Nevertheless this is unacceptable in our factored setting, and we now seek to alleviate this computational burden by applying $\mathsf{S}_{\mathsf{max}}$ to abstract sub-systems obtained using projections that motivated Definition 17. Such abstractions can be significantly smaller than the concrete systems, thus motivating a hybrid approach that can exponentially reduce bound computation times.

Example 19. *Consider applying the approach outlined in Example 14 to compute* PUR *only on the abstractions $\delta\downarrow_{\text{ROOM1}}$ and $\delta\downarrow_{\text{ROOM2}}$.* PUR($\delta\downarrow_{\text{ROOM1}}$) *can be evaluated in $\prod_{vs\in\Omega(\delta\downarrow_{\text{ROOM1}})}|vs|$ calls to $\mathsf{S}_{\mathsf{max}}$, where $\Omega(\delta\downarrow_{\text{ROOM1}})$=$\{ck_1, lk_1, \{gk_{1,2}\}, \{gk_{1,3}\}, \{gk_{2,2}\}, \{gk_{2,3}\}\}$. The same observation can be made for the evaluation time of* PUR($\delta\downarrow_{\text{ROOM2}}$). *Thus the product expression in Example 18 is split into a sum if* PUR *is called on projections.*

We now give an upper-bounding algorithm, HYB, that combines exploitation of acyclic variable dependency with exploitation of acyclicity in state spaces.

Algorithm 5: $\textsc{Hyb}(\delta)$

Compute the dependency graph $\mathcal{G}_{\mathcal{D}(\delta)}$ of δ and its SCCs
Compute the lifted dependency graph $\mathcal{G}_{\text{VS}}$
if $2 \leq |\mathcal{G}_{\text{VS}}.V|$ **return** $\mathsf{N}_{\mathsf{sum}}\langle\textsc{Hyb}\rangle(\delta, \mathcal{G}_{\text{VS}})$
else if $\Omega(\delta) \neq \emptyset$ **return** $\mathsf{S}_{\mathsf{max}}\langle\textsc{Hyb}\rangle(ch(\Omega(\delta)), \delta)$
else return $\textsc{UpBnd}(\delta)$

In HYB, ch is an arbitrary choice function. Note that in HYB, $\mathsf{S}_{\mathsf{max}}$ is only applied to the given transition system δ if there is no non-trivial projection (i.e.if $\mathcal{G}_{\mathcal{D}(\delta)}$ has one SCC), and UPBND is applied only to base-cases. Also note that $\mathcal{G}_{\mathcal{D}(\delta)}$ is constructed and analysed with every recursive call to HYB, as snapshotting in earlier calls can remove variable dependencies as a result of removing actions, leading to the breaking of the SCCs in $\mathcal{G}_{\mathcal{D}(\delta)}$, as shown in Example 20.

Example 20. *As shown in Figure 3.13b, the dependency graph of $\delta\downarrow_{\text{ROOM1}}$ has a single SCC, and thus not susceptible to dependency analysis. Taking a snapshot of $\delta\downarrow_{\text{ROOM1}}$ at the assignment $CK_{1,2}$ yields a system with one SCC in its dependency graph as well, as shown in Figure 3.13e. However, taking the snapshot of $\delta\downarrow_{\text{ROOM1}}{\restriction}_{CK_{1,2}}$ at the assignment $\{\overline{lk_{1,1}}, lk_{1,2}, \overline{lk_{1,3}}\}$, denoted by $LK_{1,2}$, yields a system with an acyclic dependency graph as shown in Figure 3.13f.*

We prove HYB is sound by proving it is sound for as tight a base function as possible. Then soundness for using UPBND as a base function follows. As discussed above, d cannot be used, because $\mathsf{N}_{\mathsf{sum}}\langle d\rangle$ is not a valid upper bound on d. However, using ℓ as a base-case function is sound. To prove that, we derive an analogue of Theorem 13 for ℓ, where $\mathsf{S}_{\mathsf{max}}\langle\ell\rangle(vs, \delta) = \mathsf{S}_{\mathsf{max}}\langle\ell\rangle(\Sigma(\!|\mathcal{G}(\delta\downarrow_{vs})|\!))$.

Theorem 14. *If $\delta\downarrow_{vs}$ is acyclic and b bounds ℓ, then $\ell(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.*

This theorem follows from an argument analogous to that provided for Theorem 13, taking ℓ to be d. Using this theorem, and Lemma 3, the validity of HYB as an upper bound on ℓ (and accordingly, the diameter) follows.

Proposition 9. *If* UPBND *bounds ℓ, then $\ell(\delta) \leq \textsc{Hyb}(\delta)$.*

3.9 Empirical Evaluations

We first discuss the practicalities of implementing HYB. Following Rintanen and Gretton (2013), we take a base-case function, UPBND, which gives the cardinality of the state space. This choice is pragmatic, taken in light of the fact that computation of alternatives, such as recurrence and sublist diameters, is NP-hard. To optimise computing $\mathsf{N}_{\mathsf{sum}}$ and $\mathsf{S}_{\mathsf{max}}$, we use memoisation, where we compute N or S once for every projection or snapshot, respectively, and store it in a look-up table. This reduced the bound computation time by 70% on average. Our evaluation considers problems from previous International Planning Competitions (IPC), and the unsolvablity IPC, and open Qualitative Preference Rovers benchmarks from IPC2006. Below, the latter are referred to as NEWOPEN.

3.9.1 Quality of HYB Bounds

Two measurements related to a compositional upper-bounding algorithm are indicative of its quality. First, we seek an indication of the degree of decompositionality provided by the algorithm. An indication is provided by comparing the size of the domain of the concrete problem—i.e. $|\mathcal{D}(\delta)|$—with that of the largest base-case. A strong decomposition is indicated when the domain of the base-case is small relative to the concrete problem. Second, we seek an approach that is able to produce bounds that grow sub-exponentiallly with the size of the problem, when they exist. Thus, we measure how the upper bounds scale in domains as the size of the problem instances grow. If the bounds scale gracefully, this indicates an effective compositional approach.

We report our measurements of the performance of HYB in these terms. Our experiments were conducted on a uniform cluster with time and memory limits of 30minutes and 4GB, respectively. Figure 3.14b shows the domain size of the largest base-case compared to the size of the concrete problem. IPC domains with instances remarkably susceptible to decomposition by HYB are: ROVERS (both solvable and unsolvable), STORAGE, TPP (both solvable and unsolvable), LOGISTICS, NEWOPEN, NOMYSTERY (both solvable and over-subscribed), UNSOLVABLE MYSTERY, VISITALL, SATELLITES, ZENO TRAVEL, and ELEVATORS. For those problems, the size of the largest base-case is significantly smaller than the size of the concrete problem, as shown in Figure 3.14b. One IPC domain that is particularly amenable to decomposition is the ROVERS domain, where many of its instances are decomposed to have largest base-cases modelling a single Boolean state-variable. Also, for domains susceptible to decomposition, the bounds computed by HYB grow sub-exponentially with the number of state variables, as shown in Figure 3.14a. We also note that out of those domains, LOGISTICS, NOMYSTERY, SATELLITES, ZENO TRAVEL and ELEVATORS, have linear (or almost linear) growth of the bounds with the size of the problem.

We also ran HYB on a PDDL McDermott et al. (1998) encoding of the hotel key protocol, with the parameters G, k, and R ranging between 1 and 10 (i.e. 1000 instances of the protocol). As shown in Figure 3.14b (and in the examples earlier), this protocol is particularly amenable to decomposition by HYB. All instances had a largest base-case modelling a single Boolean state-variable. Additionally, the bounds computed by HYB for this set of benchmarks are constant in the number of guests G, grow linearly in the number of rooms R, and quadratically in the number of keys per room K.

3.9.2 Comparison of HYB and $\mathsf{N}_{\mathsf{sum}}$

We compared the performance of the hybrid compositional bounding algorithm HYB with $\mathsf{N}_{\mathsf{sum}}$, which we showed earlier to dominate other state-of-the-art compositional bounding algorithms. Our experimental cluster and settings are as above. Our analysis and experimentation shows that HYB significantly outperforms $\mathsf{N}_{\mathsf{sum}}$, both in terms of decomposition quality and the tightness of computed bounds. This is particularly the case for the domains: NEWOPEN, NOMYSTERY, ROVERS, HYP, TPP, VISITALL, and BOTTLENECK. The success of HYB in our experimentation reveals something of an abundance of problems with acyclicity in their state space. Figure 3.15b indicates that HYB is more successful in decomposing problems compared to $\mathsf{N}_{\mathsf{sum}}$, where the largest base-cases for HYB are smaller than those for $\mathsf{N}_{\mathsf{sum}}$ in 71% of the IPC problems. This observation is reinforced, considering that the 1000th largest bound computed by HYB is $50,534$, while the 1000th largest bound computed by $\mathsf{N}_{\mathsf{sum}}$ is more than 10^6. In the HOTELKEY domain, the difference is even more pronounced. The bound computed by HYB is at most 990 for all the 1000 instances, while for $\mathsf{N}_{\mathsf{sum}}$ only 285 instances have bounds less than 10^6.

Figure 3.15a shows the computational cost of this improved bounding performance. HYB typically required more computation time than $\mathsf{N}_{\mathsf{sum}}$. However, HYB terminated in 60 seconds,

or less, for 93% of the benchmarks. Thus, we have not observed a significant time penalty. We note that the improved decompositionality over $\mathsf{N}_{\mathsf{sum}}$ exhibited here has further application yet to be explored. Should we take UPBND to be a more expensive operator, such as the NP-hard recurrence or sublist diameters, the stronger decomposition indicates that UPBND is invoked for relatively small instances when using HYB compared to $\mathsf{N}_{\mathsf{sum}}$. Thus, computing UPBND can be exponentially easier for decompositions computed by HYB compared to decompositions from $\mathsf{N}_{\mathsf{sum}}$.

3.9.3 Planning with HYB

To evaluate the practical utility of the bounds calculated using HYB, we take them as the queried horizon using the MP version of the SAT-based planner Madagascar Rintanen (2012). In our experiments we limited the time and memory for planners to 1 hour (inclusive of bound computation) and 4GB. The resulting planner proves the safety of 635 instances of the hotel key protocol, where the instance with 9 rooms, 7 guests, and 45 keys, takes the longest to prove safe – it took just under 30 minutes. This is a substantial improvement over the size of instances automatically proven safe in earlier work. We also ran AIDOS 1 Seipp et al. (2014) (unsolvability IPC winner) on the hotel key instances and it proved the safety of only 285 of them, where the instance with 2 rooms, 5 guests, and 10 keys, took the longest to prove safe – in 17 minutes. For the IPC benchmarks, our planner proved that 53 instances are unsolvable, 27 of which could not be proven unsolvable by AIDOS 1. The 27 instances are from BOTTLENECK (7 problems), 3UNSAT (4 problems), ELEVATORS (5 problems), and NEWOPEN (11 problems). We also note that compared to the system from Rintanen and Gretton (2013), we are additionally able to close the heretofore open 7th and 8th Qualitative Preference problem from IPC2006. We also found our bounds useful in solving satisfiable benchmarks. It allowed MP to solve 162 instances that it could not with its default query strategy. Those instances are from ELEVATORS (150 problems), DIAGNOSIS (8 problems), ROVERS (1 problem) and SLIDING-TILES (3 problems).

3.10 Conclusion and Open Questions

The practical incompleteness of SAT-based planning and bounded model checking has for some years been noted as a significant problem Clarke et al. (2004). It is perceived as a deficiency of SAT methods in making comparisons with state based methods. This is due to the absence of effective upper-bounding methods for topological properties of digraphs that model state spaces. We have addressed that deficiency by advancing the compositional approach to upper-bounding, virtually, the only practical approach. Although the compositional approach was intensely investigated to solve other graph theoretic questions concerning state spaces, most notably reachability, there is not as much work on compositionally upper-bounding topological properties of state spaces, except for Baumgartner et al. (2002); Rintanen and Gretton (2013). We considered a compositional approach based on a well studied abstraction, which is *projection*, and a well studied system structure, which is *state variable dependency*. We showed that projection and variable dependencies cannot be used to compositionally upper-bound two interesting topological properties: the diameter and the recurrence diameter.

To mitigate this, we defined new topological properties, the *sublist diameter* and the *traversal diameter*, which can be upper-bounded compositionally, and accordingly can be used to upper-bound the diameter and/or the recurrence diameter. In addition to being compositionally bounded, those new properties can be exponentially smaller than existing properties (like the size of the state space or the recurrence diameter) that were used in previous works to compositionally upper-

bound the diameter and the recurrence diameter. We also showed that those new parameters have interesting properties. For example, the sublist diameter exploits the factored representation: its value depends on the factored representation rather than the digraph modelling the state space. The traversal diameter on the other hand can be computed in linear time, and thus can be generally used as a completeness threshold instead of the NP-hard recurrence diameter.

To be able to deal with realistic problems we introduced a compositional procedure that exploits *acyclicity in the state space* and that is based on a new abstraction that we defined, *snapshotting*. An advantage of this new approach is that both, the diameter and the recurrence diameter can be compositonally bounded using it. Combined with projection based compositional bounding, the hybrid procedure produces very fine grained abstractions and very tight bounds compared to existing compositional upper-bounding procedures. Those benefits comes with the risk of an exponential explosion in the number of abstractions considered by the algorithm. The runtime measurements we made experimentally suggest that this theoretical risk is not realised in practice.

An important contribution was our methodology of defining new topological properties and new abstractions to indirectly compositionally upper-bound interesting topological properties, that themselves cannot be upper-bounded compositionally. We believe that the following questions are interesting for future research.

- Are there certain classes of temporal logic formulae, for which the sublist diameter or the traversal diameter is a tight completeness threshold?
- Does every digraph have a factored representation with a sublist diameter equal to the diameter? If this is possible, obtaining this factoring must be at least an NP-hard problem, since computing the sublist diameter is NP-hard, while computing the diameter is in P. If this is not the case, how close can the sublist diameter be, to the diameter, after factoring?
- $\mathsf{N}_{\mathsf{sum}}$ computes bounds on the diameter that *cannot* be improved, if the sublist diameters or the recurrence diameters of projections, and the dependencies are given. Would $\mathsf{N}_{\mathsf{sum}}$ still be tight given more knowledge, like the projections themselves instead of their sublist diameters or recurrence diameters.
- As we stated earlier, the polynomial returned by $\mathsf{N}_{\mathsf{sum}}$ depends only on the structure of the given DAG. We conjecture that for *any* polynomial, ρ, there is a DAG for which $\mathsf{N}_{\mathsf{sum}}$ returns a polynomial isomorphic with ρ.

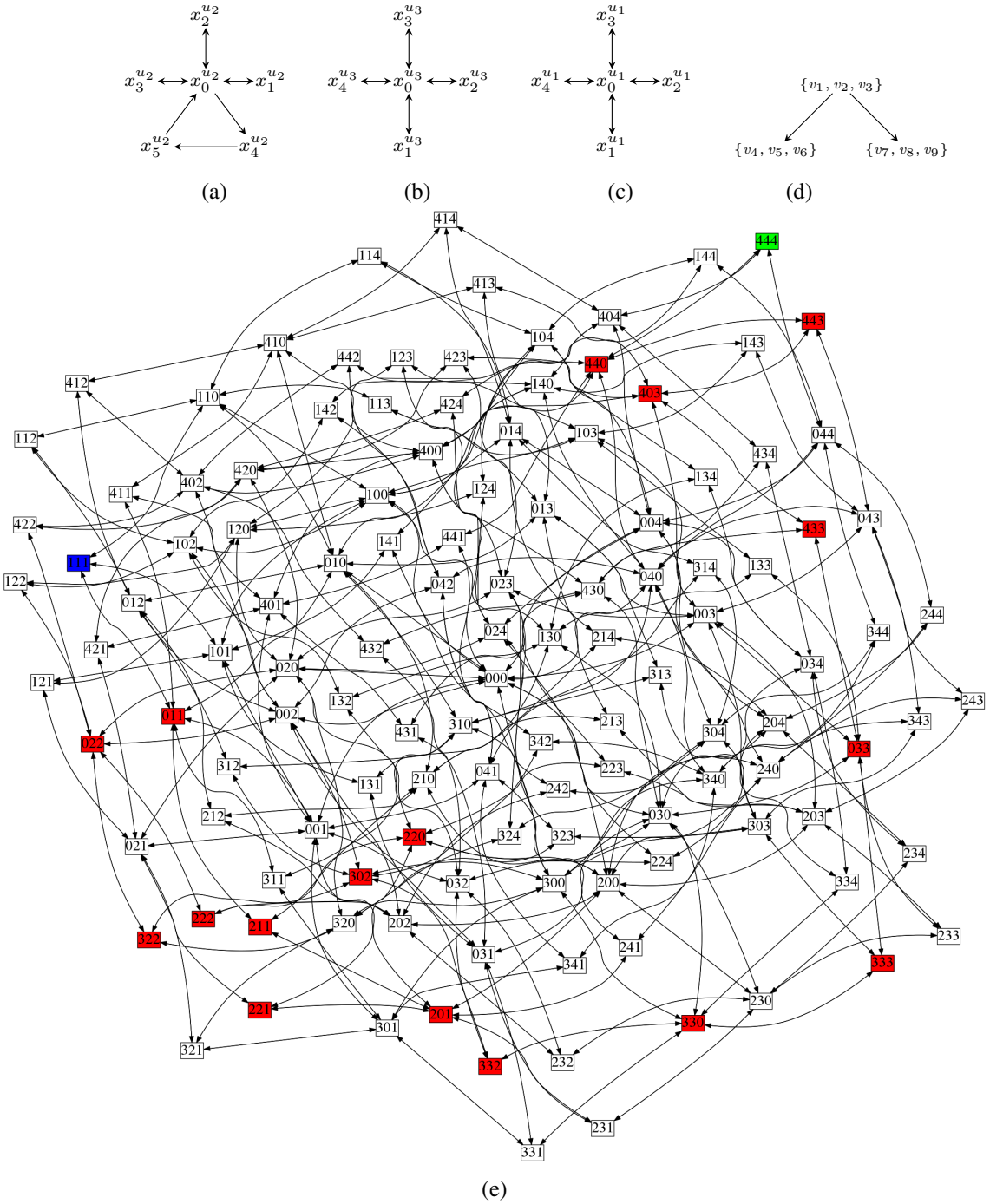


Figure 3.6: Referring to Example 5. (a), (b), and (c) are the largest connected components in the state spaces of the flowers $\mathbb{I}_{4,2}^{u_1}$, $\mathbb{I}_{4,3}^{u_2}$ and $\mathbb{I}_{4,2}^{u_3}$, respectively, (d) a lifted dependency graph of the factored system δ, and (e) is the largest connected component in the state space of δ. In this graph our notation is flattened, where for instance the state $x_i^{u_1} \uplus x_j^{u_2} \uplus x_k^{u_3}$ is denoted by ijk. States x_1 and x_2 are indicated by the blue and green vertices, respectively. The red vertices indicate the states traversed by $\vec{\pi}$ if executed on x_1. Note that (a), (b), and (c) are contractions of (e).

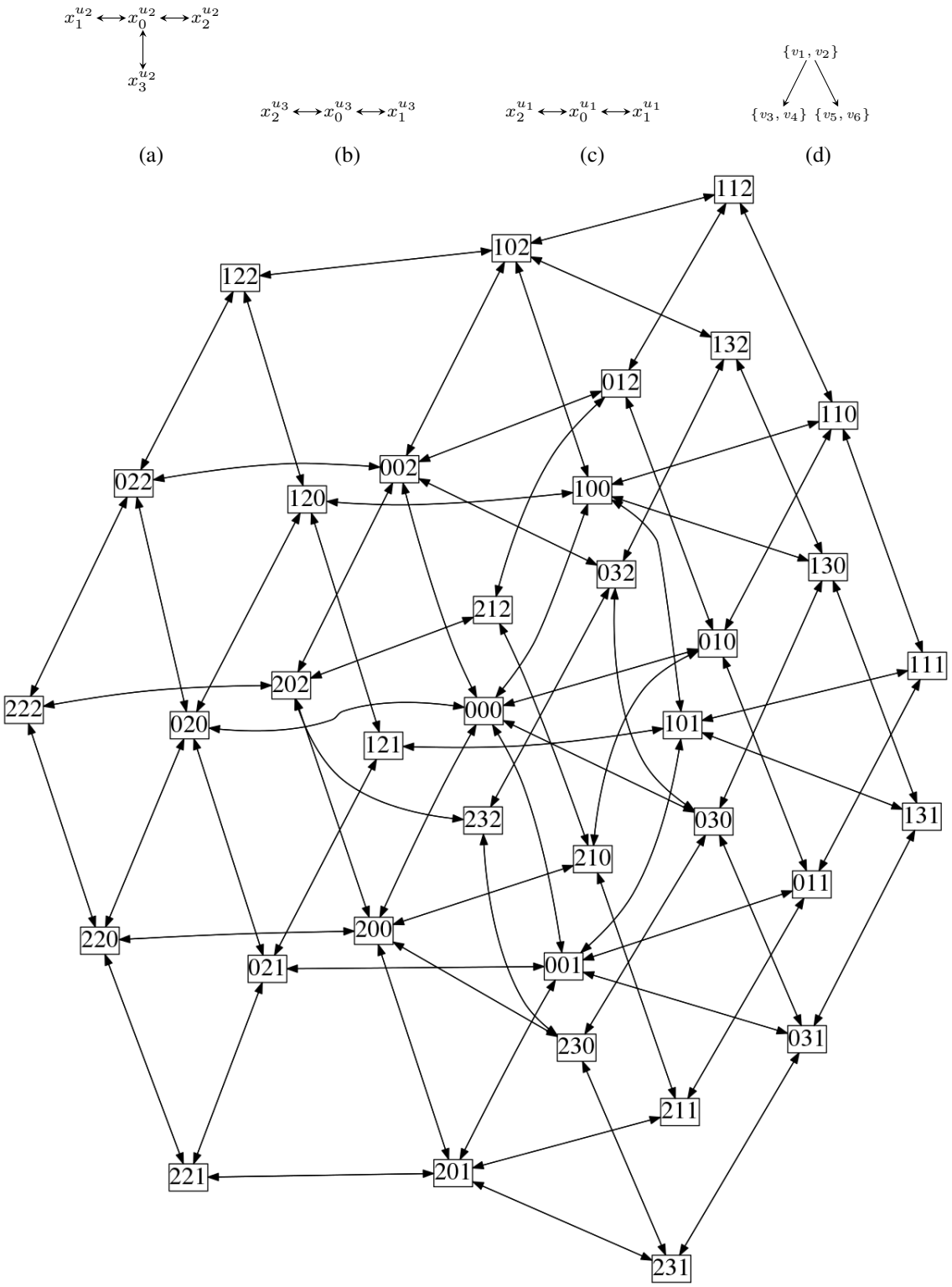


Figure 3.7: Referring to Example 7, (a), (b), and (c) are the largest connected components in the state spaces of the flowers $\mathbb{f}_{3,2}^{u_2}$, $\mathbb{f}_{2,2}^{u_3}$ and $\mathbb{f}_{2,2}^{u_1}$, respectively, (d) a lifted dependency graph of the factored system δ from Example 7, and (e) is the largest connected component in the state space of δ.

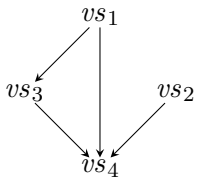


Figure 3.8: A lifted dependency graph for a factored digraph that has four sets of variables closed under mutual dependency.

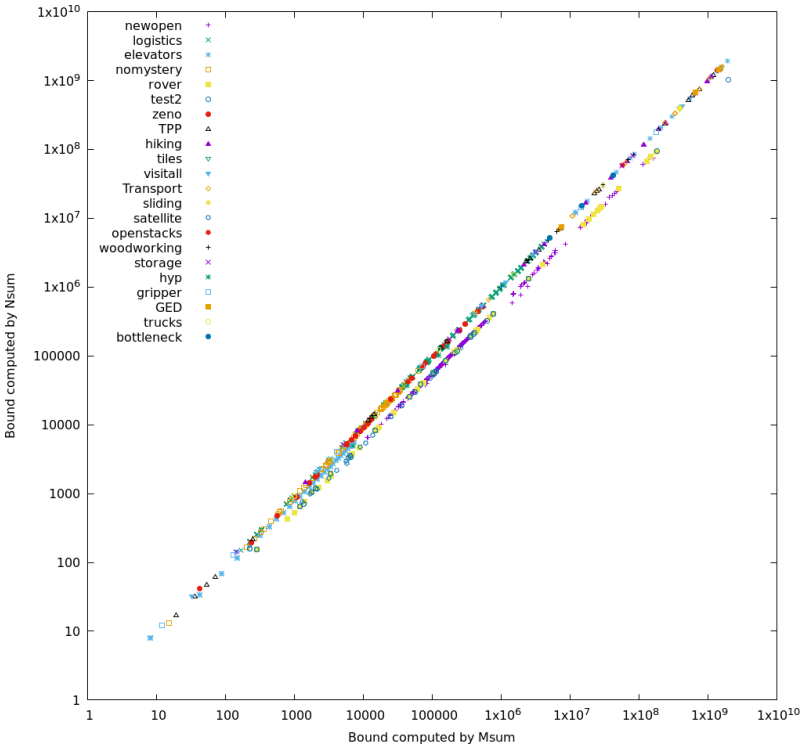


Figure 3.9: The bounds computed by $\mathsf{N}_{\mathsf{sum}}\langle b\rangle$ versus $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$ with $2^{|\mathcal{D}(\delta)|}-1$ as a base function.

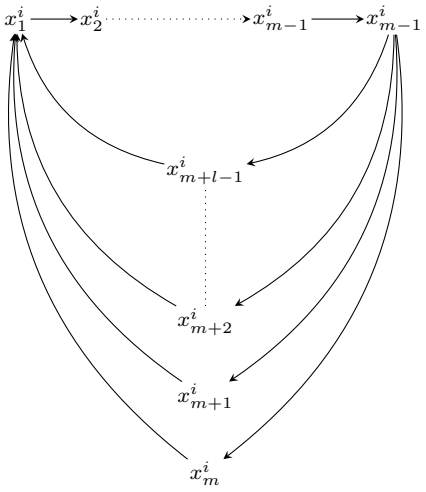


Figure 3.10: An inverted flower with $3 \leq m$.

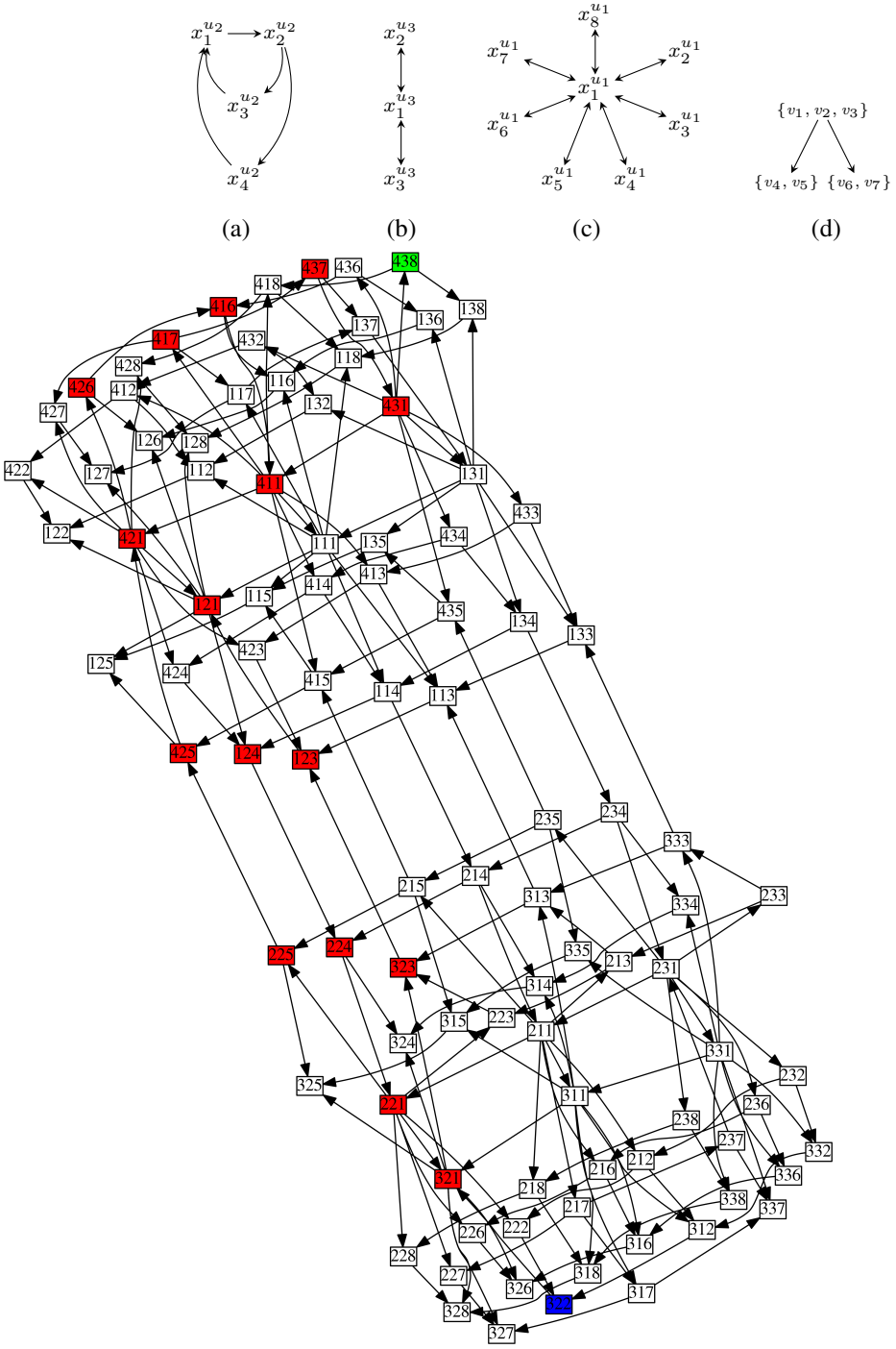
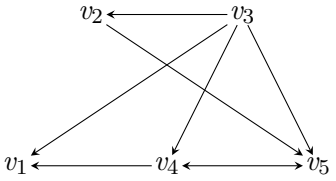


Figure 3.11: Referring to Example 10, (a), (b), and (c) are the largest connected components in the state spaces of the inverted flowers $\mathbb{l}^{u_2}_{2,3}$, $\mathbb{l}^{u_3}_{2,2}$ and $\mathbb{l}^{u_1}_{7,2}$, respectively (i.e. $\mathcal{G}(\mathbb{l}^{u_2}_{2,3}), \mathcal{G}(\mathbb{l}^{u_3}_{2,2})$, and $\mathcal{G}(\mathbb{l}^{u_1}_{7,2})$), (d) a lifted dependency graph of the factored system δ, and (e) is the largest connected component in the state space of δ. States x_1 and x_2 are indicated by the blue and green vertices, respectively. The red vertices indicate the states traversed by $\overrightarrow{\pi}$ if executed on x_1. Note that (a), (b), and (c) are contractions of (e).

$$
\begin{array}{rcl}
[\,] \underset{vs}{\between} \overrightarrow{\pi} & = & \overrightarrow{\pi}\lfloor_{\overline{vs}} \\
\overrightarrow{\pi}_c \underset{vs}{\between} [\,] & = & [\,] \\
\pi_c :: \overrightarrow{\pi}_c \underset{vs}{\between} \pi :: \overrightarrow{\pi} & = & \begin{cases} \pi :: (\overrightarrow{\pi}_c \underset{vs}{\between} \overrightarrow{\pi}) & \text{if } \pi\lfloor_{vs} = \pi_c \\ \pi_c :: \overrightarrow{\pi}_c \underset{vs}{\between} \overrightarrow{\pi} & \text{o/wise} \end{cases}
\end{array}
$$

(a) .



(b)

Figure 3.12: (a)The definition of the stitching function ($\between$), and (b) is the dependency graph of the system in Example 11.

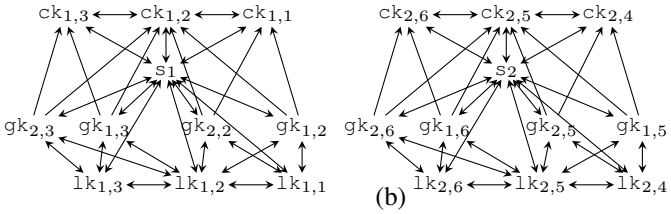
; check in to a room (at reception), receiving a new key

$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{1,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{1,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{2,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{2,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{lk}_{2,4}\},\{\mathtt{gk}_{1,5},\mathtt{lk}_{1,5},\overline{\mathtt{lk}_{1,4}},\overline{\mathtt{s}_2}\}),(\{\mathtt{lk}_{2,5}\},\{\mathtt{gk}_{1,6},\mathtt{lk}_{1,6},\overline{\mathtt{lk}_{1,5}},\overline{\mathtt{s}_2}\}),$
$(\{\mathtt{lk}_{2,4}\},\{\mathtt{gk}_{2,5},\mathtt{lk}_{1,5},\overline{\mathtt{lk}_{1,4}},\overline{\mathtt{s}_2}\}),(\{\mathtt{lk}_{2,5}\},\{\mathtt{gk}_{2,6},\mathtt{lk}_{1,6},\overline{\mathtt{lk}_{1,5}},\overline{\mathtt{s}_2}\})$

; enter a room with new key

$(\{\mathtt{gk}_{1,2}\},\{\mathtt{ck}_{1,2},\overline{\mathtt{ck}_{1,1}},\mathtt{s}_1\}),(\{\mathtt{gk}_{2,2}\},\{\mathtt{ck}_{1,2},\overline{\mathtt{ck}_{1,1}},\mathtt{s}_1\}),$
$(\{\mathtt{gk}_{1,3}\},\{\mathtt{ck}_{1,3},\overline{\mathtt{ck}_{1,2}},\mathtt{s}_1\}),(\{\mathtt{gk}_{2,3}\},\{\mathtt{ck}_{1,3},\overline{\mathtt{ck}_{1,2}},\mathtt{s}_1\}),$
$(\{\mathtt{gk}_{1,5}\},\{\mathtt{ck}_{2,5},\overline{\mathtt{ck}_{2,4}},\mathtt{s}_2\}),(\{\mathtt{gk}_{2,5}\},\{\mathtt{ck}_{2,5},\overline{\mathtt{ck}_{2,4}},\mathtt{s}_2\}),$
$(\{\mathtt{gk}_{1,6}\},\{\mathtt{ck}_{2,6},\overline{\mathtt{ck}_{2,5}},\mathtt{s}_2\}),(\{\mathtt{gk}_{2,6}\},\{\mathtt{ck}_{2,6},\overline{\mathtt{ck}_{2,5}},\mathtt{s}_2\})$

(a)



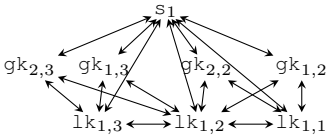
(b)

$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{1,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{1,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{2,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{2,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{gk}_{1,2}\},\{\mathtt{ck}_{1,2},\overline{\mathtt{ck}_{1,1}},\mathtt{s}_1\}),(\{\mathtt{gk}_{2,2}\},\{\mathtt{ck}_{1,2},\overline{\mathtt{ck}_{1,1}},\mathtt{s}_1\}),$
$(\{\mathtt{gk}_{1,3}\},\{\mathtt{ck}_{1,3},\overline{\mathtt{ck}_{1,2}},\mathtt{s}_1\}),(\{\mathtt{gk}_{2,3}\},\{\mathtt{ck}_{1,3},\overline{\mathtt{ck}_{1,2}},\mathtt{s}_1\})$

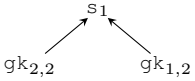
(c)

$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{1,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{1,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{lk}_{1,1}\},\{\mathtt{gk}_{2,2},\mathtt{lk}_{1,2},\overline{\mathtt{lk}_{1,1}},\overline{\mathtt{s}_1}\}),(\{\mathtt{lk}_{1,2}\},\{\mathtt{gk}_{2,3},\mathtt{lk}_{1,3},\overline{\mathtt{lk}_{1,2}},\overline{\mathtt{s}_1}\}),$
$(\{\mathtt{gk}_{1,2}\},\{\mathtt{s}_1\}),(\{\mathtt{gk}_{2,2}\},\{\mathtt{s}_1\})$

(d)

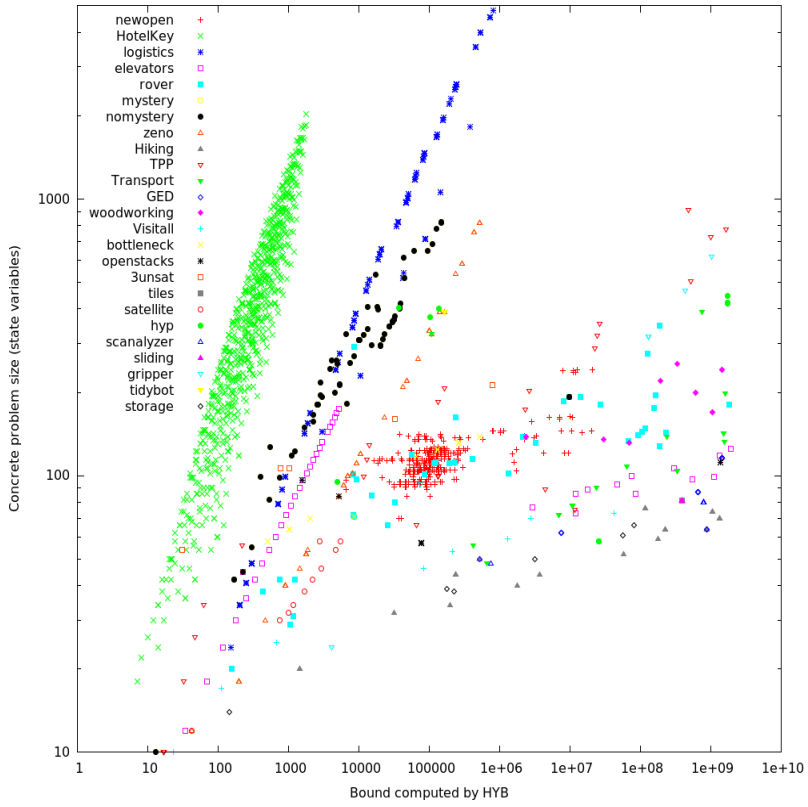


(e)

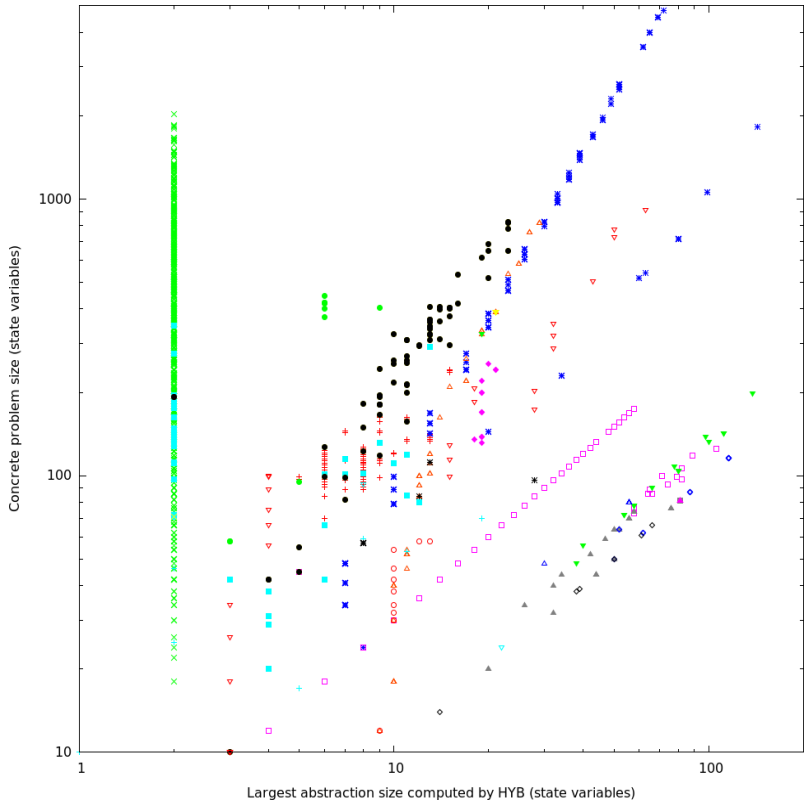


(f)

Figure 3.13: (a) shows the actions of a *transition system* δ representing the hotel key protocol with 2 rooms, 2 guests and 3 keys per room; room 1 is associated with keys 1–3; room 2 with keys 4–6. (b) is the *dependency graph* for that system. (c) is the *projection* of the system on an abstraction that models only the changes related to room 1. (d) is the *snapshot* of $\delta\lfloor_{\text{ROOM1}}$ on $CK_{1,2}$, an abstraction that only analyses the changes related to room 1 when its door recognises key 2 as the current key. (e) and (f) are the dependency graphs of snapshots that we use for illustrative purposes in the examples.

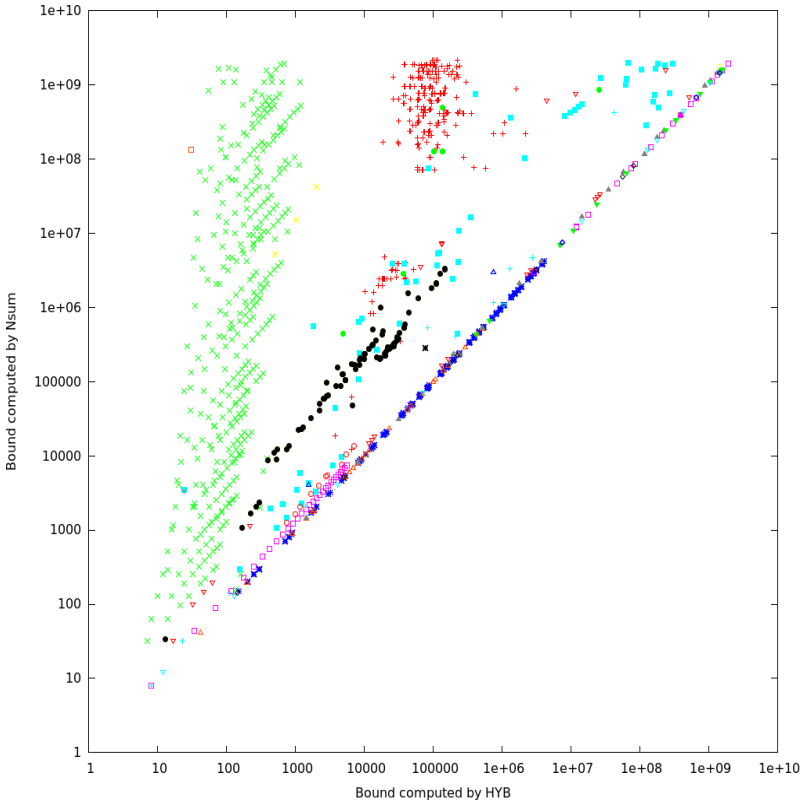


(a)

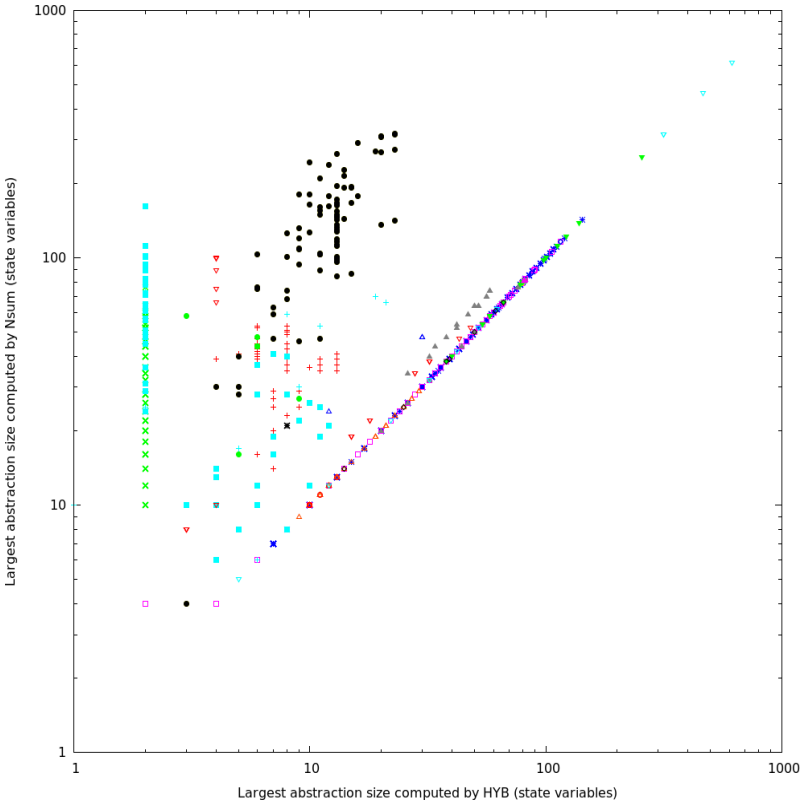


(b)

Figure 3.14: Measurements related to HYB on benchmarks. (a) Scatter plot of the bound (horizontal axis) computed by HYB, and the size (i.e. $|\mathcal{D}(\delta)|$) of the concrete problem (vertical). (b) Scatter plot of the size of the largest base-case (horizontal), and the size of the concrete problem (vertical).



(a)



(b)

Figure 3.15: Comparison of the bounding performance of $\mathsf{N}_{\mathsf{sum}}$ and HYB on benchmarks. Legend is provided in Figure 3.14a. (a) Scatter plot of the bounds computed by HYB (horizontal axis) and the state-of-the-art bounding algorithm $\mathsf{N}_{\mathsf{sum}}$ (vertical). (b) Scatter plot of the size (i.e. $|\mathcal{D}(\delta)|$) of the largest base-case using HYB (horizontal axis) and $\mathsf{N}_{\mathsf{sum}}$ (vertical).

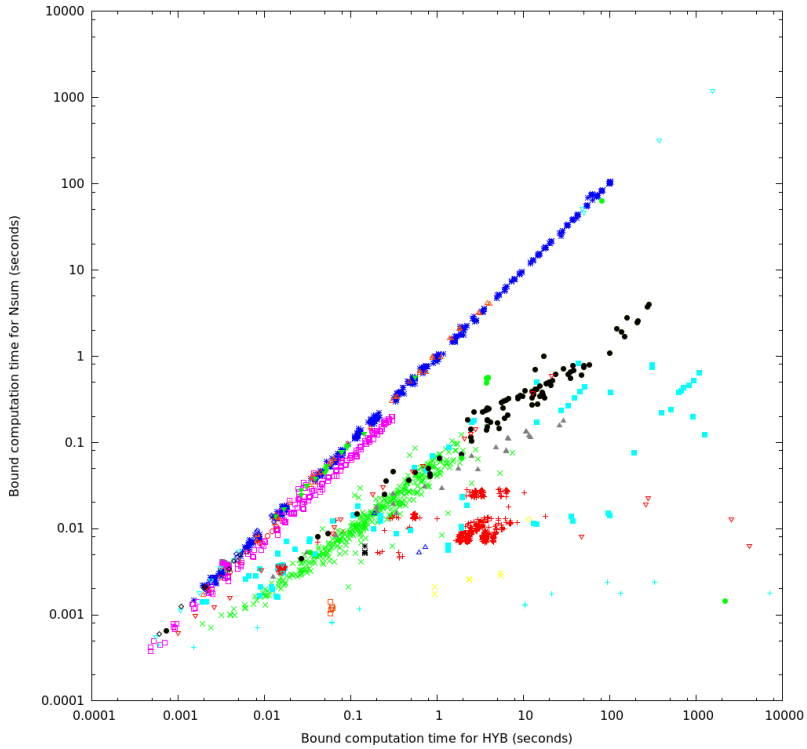


Figure 3.16: Scatter plot of computation time (in seconds) of HYB (horizontal axis) and N_{sum} (vertical) for benchmarks. Legend is provided in Figure 3.14a.

Chapter 4

Compositional Computation of Reachability in the Presence of Repetitive Symmetry

Testing whether one vertex (or a set of vertices) in a digraph can be reached from another vertex is one of the oldest and most well studied problems in computer science and graph theory. In AI planning and model checking, this problem is of immense importance, where the digraph is taken to model the state space of the problem, and the question is whether a goal state can be reached from a given initial state. In AI planning, the problem of reachability between states is the main problem whose solution is sought. An initial state and a set of goal states are given, and the quest is to find a valid action sequence (a *plan*) that if executed in the initial state will result in one of the goal states. Similarly, in model checking, the problem of reachability between states corresponds to the problem of checking safety properties of systems. An initial state of the system is given, along with a formula characterising desired safety properties, and a sequence of transitions (a bug trace) from the initial state to a state violating the given formula is searched for.

Although many polynomial time algorithms (e.g. Bellman-Ford algorithm) were introduced to solve the reachability problem for digraphs, they all assume that the digraph is explicit. Accordingly, for propositionally factored transition systems, if one uses any of those algorithms naively, he needs to construct the digraph modelling the state space to solve the reachability problem. Thus, the complexity of reachability in factored transition systems is PSPACE-complete Sistla and Clarke (1985); Bylander (1994), since the digraph modelling the state space can be exponentially larger than the factored representation. The compositional approach alleviates can alleviate the practical complexity of computing reachability, making the problem feasible to solve in many cases, where it otherwise may not be. In this approach, digraphs modelling state spaces of abstractions (which are minors of the original state space) are searched for paths, which are then used to synthesise a path in the digraph modelling the state space of the concrete system, a path that connects the required states. In this chapter we consider applying that approach to compute reachability, where we consider minors of the state space corresponding to the state space of an abstraction of the factored system that we refer to as a "descriptive quotient". In particular, we exploit a certain type of symmetry, which we call *repetitive symmetries*, in the factored system to obtain those abstractions.

State Variable Symmetry Informally, in a factored system δ, two state variables v_1 and v_2 are symmetric if they can be swapped by a permutation of the state variables of δ, and the resulting system is the same (up to isomorphism) as δ. This state variable symmetry relation induces a partition of the set of state variables and a partition of the actions. We find those partitions most

useful for use with quotienting within our work.

4.1 Related Work

Algorithms for finding a path or the shortest path between two vertices in a digraph were devised since the middle of last century. The earliest algorithm that we are aware of, the Bellman-Ford algorithm Bellman (1958), finds the shortest path between two vertices in a time linear in the product of the edges and vertices of a digraph. Many algorithms were introduced after that like the ones in Dijkstra (1959); Fredman and Tarjan (1987); Thorup (2003). All of those algorithms assume that the digraph is represented explicitly, and also they are all at least linear in both, the number of vertices and the number of edges. Accordingly a naive application of these algorithms to finding paths in state spaces of factored systems would be infeasible.

Compositional Planning and Model Checking

The compositional approach was employed to find solutions to AI planning and model checking problems in work like Knoblock (1994); Williams and Nayak (1997); Berezin et al. (1998); Case et al. (2009); Kroening (2006); Amir and Engelhardt (2003); Brafman and Domshlak (2006); Kelareva et al. (2007); Guere and Alami (2001); Helmert et al. (2014); Sievers et al. (2015). In those works, different types of abstractions and structures were used, and only the state space of a projected system is searched for a path. For example, abstractions based on projection were used in Knoblock (1994); Williams and Nayak (1997), which exploited acyclicity in variable dependencies. Another approach for compositional path search, known as *factored planning*, is used by Amir and Engelhardt (2003); Brafman and Domshlak (2006); Kelareva et al. (2007), in which the factored system δ is abstracted into multiple abstractions, referred to as "factors". These factors are obtained by an abstraction that, given a partition of the set of state variables $vs_{1..n}$, it produces a set of sets of actions, one per $vs_i \in vs_{1..n}$, such that the set of actions refers only to variables in vs_i. This partition of state variables is based on a tree decomposition of an undirected version of the variable dependency graph.

Exploiting Symmetries

Symmetries that occur in factored transition systems have for some time been exploited for efficient search for paths between states, both by AI planning and model checking communities. The quintessential planning scenario which exhibits symmetries is GRIPPER. This comprises a robot whose left and right grippers can be used interchangeably in the task of moving a set of N indistinguishable packages from a source location to a goal location. Intuitively the left and right grippers are symmetric because if we changed their names, by interchanging the terms left and right in the problem description, we are left with an identical problem. Packages are also interchangeable and symmetric.

One method to exploit symmetry is to perform search in a minor of the digraph modelling the state space, referred to as a *quotient system*, which corresponds to a (sometimes exponentially) smaller bisimulation of the system at hand. Planning in a quotient system, a state x is represented by a *canonical* element from its *orbit*, the set of states which are symmetric to x. Giving integer labels to packages in GRIPPER, when the search encounters a state where the robot is holding 1 package using 1 gripper, this is represented using the canonical state where, for example, the left gripper is holding package with identity "1". Orbit search explores the quotient system by

simulating actions from known canonical states, and then computing the canonical representations of resultant states. That canonicalisation step requires the solution to the *constructive orbit problem* Clarke et al. (1998) which is *NP-hard* Eugene (1993). A key weakness, is that for each state encountered by the search an intractable canonicalisation operation is performed. This is mitigated in practice by using approximate canonicalisation. By forgoing exact canonicalisation, one encounters a much larger search problem than necessary. For a GRIPPER instance with 42 packages, the breadth-first orbit search with approximate canonicalisation by Pochter et al. (2011) reportedly performs $60.5K$ state expansion operations, far more than necessary. This method of exploiting symmetry was firstly used in solving model checking problems, which is reviewed by Wahl and Donaldson (2010). Also, this method was recently adapted for planning and studied by Pochter et al. (2011), Domshlak et al. (2012, 2013) and Shleyfman et al. (2015). Related work about state-based planning in equivalence classes of symmetric states includes Guere and Alami (2001); Fox and Long (1999, 2002).

Following the seminal work by Crawford et al. (1996) and Brown et al. (1996), when searching for a path (in the context of AI planning) *via* constraint satisfaction, known symmetries are exploited by:

(i) including symmetry breaking constraints, either directly as part of the problem expression Joslin and Roy (1997); Rintanen (2003), or

(ii) otherwise dynamically as suggested by Miguel (2001) as part of a *nogood* mechanism.

In GRIPPER, we can statically require that if no package is yet retrieved, then any retrieval targets package 1 with the left gripper. Dynamically, having proved that no 3-step plan exists retrieving package 1 with the left gripper, then no plan of that length exists retrieving package $i \neq 1$ using either gripper. Searching using the proposed dynamic approach is quite weak, as symmetries are only broken as nogood information becomes available. A weakness of both approaches is that problems expressed as CSPs include variables describing the state of the transition system at different plan steps. Existing approaches do not break symmetries across steps, and can therefore waste effort exploring partial assignments that express executions which visit symmetric states at different steps.

Another method to exploit symmetries is via abstracting the factored system at hand. Some authors perform search in the state space of a *counter abstraction*, as surveyed by Wahl and Donaldson (2010). That approach treats a *transition system isomorphic* to the quotient system, but it avoids solving the NP-hard constructive orbit problem. Symmetries were also explored in factored planning in the context of *merge-and-shrink* heuristics Helmert et al. (2014). Sievers et al. (2015) developed a symmetry guided *merging* operation which yields relatively compact heuristic models, improving the scalability of that approach. Guere and Alami (2001) propose to plan via a *shape-graph*, a compact description of the problem state space in which states are represented by equivalence classes of symmetric states. As well as planning by searching in that graph, using the diameter of the *shape-graph* Guere and Alami are able to calculate tight upper bounds for the highly symmetric GRIPPER and BLOCKS-WORLD domains.

4.2 Results

We consider the AI planning problem defined on factored transition systems, i.e. given an initial state $I \in \mathbb{U}(\delta)$ and a set of goal states G, is there an action sequence that reaches one of the goal states if it is executed at the initial state.

Our contribution is a new abstraction of factored transition systems, the result of which we call a *descriptive quotient*. We provide conditions under which searching for a path between the initial state and a goal state (i.e. a plan) in the digraph modelling the state space of the descritptive quotient can be used to synthesise a path between the initial state and some goal state in the original system. The first condition is that the partition used to obtain the descriptive quotient is induced by the symmetry relation between state variables (a.k.a. orbits of a subgroup of the automorphism group). The second condition is that the descriptive quotient is isomorphic to a "sub-system" of the original factored system. Informally, those two conditions mean that planning via descriptive quotients is a way to exploit "repetitive symmetries" in a factored system, where by repetitive symmetries we mean that the factored system is constituted by a union of isomorphic sub-systems.

Based on that we provide a novel procedure for domain-independent planning with symmetries. Following, e.g., Pochter et al., in a first step we infer knowledge about the symmetries between state variable. Then departing from existing approaches, our second step uses that knowledge to obtain a quotient of the concrete factored system. Called the descriptive quotient, this describes any element in the set of isomorphic subsystems which abstractly model the concrete system. Third, we invoke a planner once to solve the small reachability problem posed by that descriptive quotient. In the fourth and final step, a concrete plan is synthesized by concatenating instantiations of that plan for each isomorphic subproblem.

The non-existence of a plan for the descriptive quotient does not exclude the possibility of a plan in the concrete system. Although sound, in that respect our approach is incomplete. Having an optimal plan for the quotient does not guarantee optimality in the concatenated plan. Aside from computing a plan for the descriptive quotient, the computationally challenging steps occur in preprocessing:

(i) Identification of state variable symmetries from the original description, a problem as hard as graph isomorphism, which is not known to be tractable, and

(ii) Computing an appropriate set of subsystems isomorphic to the quotient.

We introduce the general version of the latter problem: for an undirected graph $\mathcal{G}$ and a partition of its vertices P, is the quotient $\mathcal{G}/P$ isomorphic to a subgraph of $\mathcal{G}$, such that the morphism maps every set of vertices from P to one of its members? We show that this problem is NP-complete. We also show that if P is a set of *orbits* of a symmetry group for $\mathcal{G}$, then there is set of morphisms from $\mathcal{G}/P$ to $\mathcal{G}$ that covers all the vertices of $\mathcal{G}$, and whose size is logarithmic in the vertices of $\mathcal{G}$.

Unlike existing approaches, our search for a plan does not need to reason about symmetries between concrete states and the effects of actions on those. Plan search can be performed by an off-the-shelf SAT/CSP system, in which case *symmetry breaking* constraints are not required. Alternatively, using a state-based planner we avoid repeated (approximate) solution to the intractable canonicalisation problem, a clear bottleneck of recent planning algorithms. In this respect, our approach is similar to searching in a *counter abstraction*, as surveyed by Wahl and Donaldson (2010). Also, viewing our approach as one that decomposes a problem into subproblems, it is related to factored planning Amir and Engelhardt (2003); Brafman and Domshlak (2006); Kelareva et al. (2007).

The most important advantage of searching in the digraph modelling the state space of the descriptive quotient is that it is a minor that can be very compact relative to the digraph modelling the state space of the original factored system. For instance, a counter abstraction has a state space which can be exponentially larger than that of a descriptive quotient—i.e., the descriptive quotient will model 1 object, whereas the quotient transition system models N symmetric objects. Also,

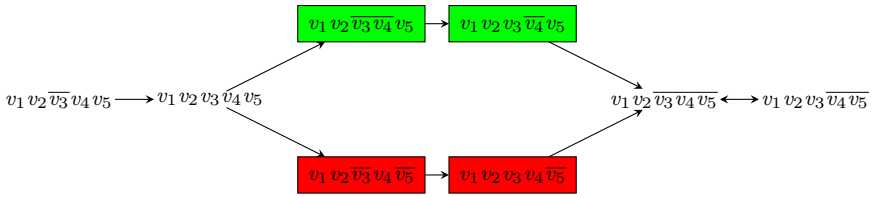


Figure 4.1: The largest connected component of the state space of the transition system underlying the problem from Example 21. It shows the presence of symmetries between different states.

existing state-based methods plan in that relatively large *quotient system*, and as just mentioned, face an additional intractable problem for every encountered state. By employing approximate canonicalisation, such methods face a state space much larger than that posed by the quotient system, which can be exponentially larger than that posed by a descriptive quotient. To give an idea of how compact can the size of the state space of the descriptive quotient, the descriptive quotient of the aforementioned 42-package GRIPPER instance is solved by breadth-first search expanding 6 states, and a concrete plan is obtained in under a second. On the other hand, the approach by Pochter et al. (2011) expands $60.5K$ states and takes about 28 seconds.

4.3 Planning Problems and Additional Notation

We formally define *planning problems*.

Definition 24 (Planning Problem)**.** *A planning problem* Π *is a 3-tuple* $\langle I, \delta, G \rangle$*, with* I *the initial state of the problem,* G *a description of goal states (another finite map from variables to Booleans), and* δ *a set of actions. We define the domain of the problem (*$\mathcal{D}(\Pi)$*) to be domain of the set of actions (*$\mathcal{D}(\delta)$*). The set of valid states with respect to a planning problem, which we write as* $\mathbb{U}(\Pi)$ *is* $\mathbb{U}(\delta)$*. Problem* Π *is* valid *if* $\mathcal{D}(G) \subseteq \mathcal{D}(I)$*, and* $I \in \mathbb{U}(\Pi)$*. We only consider valid problems. Hereafter we refer to the initial state, actions or goal of problem* Π *as* $\Pi.I$*,* $\Pi.\delta$ *or* $\Pi.G$ *respectively. We may also omit the* Π *if it is clear from the context, e.g.* I *for* $\Pi.I$ *and* δ_i *for* $\Pi_i.\delta$*. An action sequence* $\overrightarrow{\pi}$ *is a plan/solution for a planning problem* Π *iff* $\overrightarrow{\pi} \in \delta^*$ *and* $G \subseteq \mathsf{ex}(I, \overrightarrow{\pi})$*.*

Definition 25 (Subproblem)**.** *Problem* Π_1 *is a subproblem of* Π_2*, written* $\Pi_1 \subseteq \Pi_2$*, if* $I_1 \subseteq I_2$*, and if* $\delta_1 \subseteq \delta_2$*.*

Definition 25 purposefully only treats the description of system dynamics, and does not consider problem goals. We later consider subproblems with a variety of goals which do not all correspond to concrete problem goals. We form a concrete plan by concatenating plans for subproblems with such varieties of goals. For example, we will consider subproblems from GRIPPER with only one package and one gripper, where the goal is to: (i) relocate that package according to the concrete goal, (iii) additionally free the gripper, and (ii) additionally have the robot relocate to its starting position. In this case a concrete plan is formed by concatenating plans for subproblems given for each distinct package.

Lastly, we require a few notations. For a set of actions δ, we define the set of preconditions $\mathsf{pre}(\delta)$ as $\bigcup_{(p,e)\in\delta} p$. Let m be a finite map, e.g., a state x, *etc.*. Then, let $f(\!|m|\!)$ be the image of m under function f: the map $\bigcup_{(k\mapsto v)\in m}\{f(k) \mapsto v\}$. This is well-defined if $f(k_1) = f(k_2)$ implies that $m(k_1) = m(k_2)$. We lift this notion of image to other composite types. For example, we write $f(\!|\Pi|\!)$ for the image of Π under f, where all finite maps in Π are transformed by f.

Figure 4.2: The quotient state space of the system from Example 21 in which the traditional orbit search algorithm, ideally, would search for a path.

Figure 4.3: The largest connected component in the state space of the transition system underlying the descriptive quotient of the problem in Example 21.

4.4 Computing Problem Symmetries

To exploit problem symmetries we must first discover them. We follow the discovery approach from Pochter et al. (2011), restricting ourselves to Boolean-valued variables in $\mathcal{D}(\Pi)$. We assume familiarity with *groups, subgroups, group actions*, and *graph automorphism groups*. Symmetries in a problem description are defined by an *automorphism group*.

Definition 26 (Problem Automorphism Group)**.** *The automorphism group of* Π *is:* $Aut(\Pi) = \{\sigma \mid \sigma(\Pi) = \Pi\}$. *Members of* $Aut(\Pi)$ *are permutations on* $\mathcal{D}(\Pi)$ *and they induces a partition of* $\mathcal{D}(\Pi)$ *to which we refer as orbits.*

Example 21. *Consider the planning problem* Π_1 *with*

$$\begin{aligned} \Pi_1.I &= \{v_3, v_1, v_2, v_4, v_5\} \\ \Pi_1.\delta &= \{(\{v_3, v_1\}, \{\overline{v_4}, \overline{v_3}\}), (\{v_3, v_2\}, \{\overline{v_5}, \overline{v_3}\}), (\emptyset, \{v_3\})\} \\ \Pi_1.G &= \{\overline{v_4}, \overline{v_5}\} \end{aligned}$$

The largest connected component of $\mathcal{G}(\delta_1)$ *is shown in Figure 4.1.* $Aut(\Pi_1)$ *is the closure of this set of permutations under composition:* $\{\{v_1 \mapsto v_2, v_2 \mapsto v_1\}, \{v_4 \mapsto v_5, v_5 \mapsto v_4\}\}$. *Let* $vs_{1..n}$ *be* $\{p_1 = \{v_1, v_2\}, p_2 = \{v_3\}, p_3 = \{v_4, v_5\}\}$. *This partition of* $\mathcal{D}(\Pi_1)$ *is the set of orbits induced by* $Aut(\Pi_1)$. *This variable symmetry induces symmetries between the states and accordingly vertices of* $\mathcal{G}(\delta_1)$ *as shown in Figure 4.1, where state 1 is symmetric with 2, and 5 is symmetric with 6. The quotient system in which, ideally, traditional approaches would search for a path to exploit symmetry is shown in Figure 4.2, and it is clearly smaller in size than* $\mathcal{G}(\delta_1)$.

A graphical representation of Π is constructed so vertex symmetries in it correspond to variable symmetries in Π. We follow the graphical representation introduced in Pochter et al. (2011).

Definition 27 (Undirected Graph)**.** *An undirected graph* $\mathcal{G}$ *is a tuple* $\langle V, E\rangle$, *where* V *is the set of vertices of* $\mathcal{G}$ *and* E *is the set edges of* $\mathcal{G}$ *which is a set of unordered pairs from* V. *We write* $V(\mathcal{G})$ *for the set of vertices, and* $E(\mathcal{G})$ *for edges of a graph* $\mathcal{G}$.

Definition 28 (Problem Description Graph (PDG))**.** *The (undirected) graph* $\mathcal{G}(\Pi)$ *for a planning problem* Π, *is defined as follows:*

(i) $\mathcal{G}(\Pi)$ *has two vertices,* ${u_1}^\top$ *and* ${u_1}^\bot$, *for every variable* $v \in \mathcal{D}(\Pi)$; *two vertices,* a_p *and* a_e, *for every action* $(p, e) \in \delta$; *and vertex* u_{1I} *for* I *and* u_{1G} *for* G.

(ii) $\mathcal{G}(\Pi)$ *has an edge between* ${u_1}^\top$ *and* ${u_1}^\bot$ *for all* $v \in \mathcal{D}(\Pi)$; *between* a_p *and* a_e *for all* $(a, e) \in \delta$; *between* a_p *and* v^* *if* $(v \mapsto *) \in p$, *and between* a_e *and* v^* *if* $(v \mapsto *) \in e$; *between* v^* *and* v_I *if* $(v \mapsto *)$ *occurs in* I; *and between* v^* *and* v_G *if* $(v \mapsto *)$ *occurs in* G.

The automorphism group of the PDG, $Aut(\mathcal{G}(\Pi))$, is identified by solving an undirected graph isomorphism problem. The action of a subgroup of $Aut(\mathcal{G}(\Pi))$ on $V(\mathcal{G}(\Pi))$ induces a *partition*, called the *orbits*, of $V(\mathcal{G}(\Pi))$. We can now define our *quotient* structures based on partitions $vs_{1..n}$ of $\mathcal{D}(\Pi)$.

Definition 29 (Quotient Undirected Graph)**.** *For graph $\mathcal{G}$ and a partition P of its vertices, the quotient $\mathcal{G}/P$ is the digraph:*

(i) $V(\mathcal{G}/P) = P$*, and*

(ii) $E(\mathcal{G}/P) = \{\{U, W\} \mid U, W \in P \wedge \exists u_2 \in U, u_3 \in W.\{u_2, u_3\} \in E(\mathcal{G})\}$.

Definition 30 (Quotient Problem)**.** *Given partition $vs_{1..n}$ of $\mathcal{D}(\Pi)$, let $\mathcal{Q}$ map members of $\mathcal{D}(\Pi)$ to their equivalence class in $vs_{1..n}$. The descriptive quotient is $\Pi/vs_{1..n} = \mathcal{Q}(\!|\Pi|\!)$, the image of Π under $\mathcal{Q}$. This is well-defined if $vs_{1..n}$ is a set of orbits. We assume that quotient problems are well-defined.*

Example 22. *Consider the planning problem Π_1 from Example 21, and the partition $vs_{1..n}$. The descriptive quotient associated with $vs_{1..n}$ is as follows.*

$$\begin{aligned}(\Pi_1/vs_{1..n}).I &= \{p_1, p_2, p_3\}\\ (\Pi_1/vs_{1..n}).\delta &= \{(\{p_1, p_2\}, \{\overline{p_2}, \overline{p_3}\}), (\emptyset, \{p_2\})\}\\ (\Pi_1/vs_{1..n}).G &= \{\overline{p_3}\}\end{aligned}$$

The largest component of the minor $\mathcal{G}((\Pi_1/vs_{1..n}).\delta)$ is shown in Figure 4.3. It is clearly smaller than the orginal state space shown in Figure 4.1 as well as the quotient system shown in Figure 4.2.

To ensure correspondence between PDG symmetries and problem symmetries, we must ensure incompatible descriptive elements do not coalesce in the same orbit. For example, we cannot have action precondition vertices and state-variables in the same orbit.

Definition 31 (Well-formed Partitions)**.** *A partition of $V(\mathcal{G}(\Pi))$ is* well-formed *iff:*

(i) Positive ($u_1{}^{\top}$) and negative ($u_1{}^{\bot}$) variable assignment vertices only coalesce with ones of the same parity;

(ii) Precondition (a_p) and effect (a_e) vertices only coalesce with preconditions and effects respectively, and

(iii) Both u_{1I} and u_{1G} are always in a singleton.

A well-formed partition $\hat{P}$ defines a corresponding partition $vs_{1..n}$ of $\mathcal{D}(\Pi)$, so that $\mathcal{G}(\Pi)/\hat{P} = \mathcal{G}(\Pi/vs_{1..n})$

To ensure well-formedness, vertex symmetry is calculated using the *coloured* graph-isomorphism procedure (CGIP). Vertices of distinct colour cannot coalesce in the same orbit. Vertices of $\mathcal{G}(\Pi)$ are coloured to ensure the orbits correspond to a well-formed partition.

4.5 Computing the Set of Instantiations

Recapping, symmetries in Π are the basis of a partition $vs_{1..n}$ of its domain $\mathcal{D}(\Pi)$ into orbits. That exposes the *descriptive quotient*, $\Pi/vs_{1..n}$, an abstract problem whose variables correspond

to orbits of concrete symbols. Our task now is to compute a set of *instantiations* of the quotient which *cover* all the goal variables $\mathcal{D}(G)$. Called a covering set of isomorphic subproblems, by instantiating a quotient plan for each subproblem and concatenating the results we intend to arrive at a concrete plan. We describe a pragmatic approach to obtaining that covering set. We establish that a covering set is not guaranteed to exist, and give an approach to *refining* partitions to mitigate that fact. We derive a theoretical bound on the necessary size of a covering set, and prove that a general graph formulation of the problem of computing a covering instantiation is NP-complete.

Definition 32 (Transversals, Instantiations)**.** *A transversal $\pitchfork$ is an injective choice function over a set of sets S such that $\pitchfork(c) \in c$ for every equivalence class $c \in S$. A transversal* covers v *if $v \in \mathcal{R}(\pitchfork)$. If $vs_{1..n}$ is a partition of $\mathcal{D}(\Pi)$, we refer to a transversal $\pitchfork$ of $vs_{1..n}$ as an* instantiation *of $\Pi/vs_{1..n}$. An instantiation $\pitchfork$ is* consistent *with a concrete problem Π if $\pitchfork(\!|\Pi/vs_{1..n}|\!) \subseteq \Pi$. When we use the term "problem" discussing an instantiation $\pitchfork$, we intend $\pitchfork(\!|\Pi/vs_{1..n}|\!)$. Note that $\mathcal{D}(\pitchfork(\!|\Pi/vs_{1..n}|\!)) = \mathcal{R}(\pitchfork)$.*

Example 23. *Consider the set $s = \{a, b, c, d, e, f\}$, and the equivalence classes $c_1 = \{a, b\}$, $c_2 = \{c, d\}$ and $c_3 = \{e, f\}$ of its members. For the partition $P = \{c_1, c_2, c_3\}$ of s, $\pitchfork_1 = \{a, c, e\}$ and $\pitchfork_2 = \{b, c, f\}$ are two transversals.*

For each goal variable in the problem, our approach shall need to find a consistent instantiation of the quotient which covers that. We thus face the following problem.

Problem 1. *Given Π and a partition $vs_{1..n}$ of $\mathcal{D}(\Pi)$, is there an instantiation of $\Pi/vs_{1..n}$ consistent with Π that covers a variable $v \in \mathcal{D}(\Pi)$?*

Example 24. *Consider the planning problem Π_1 from Example 22. Let instantiation $\pitchfork$ be $\{p_1 \mapsto v_3, p_2 \mapsto v_1, p_3 \mapsto v_4\}$. Then, $\pitchfork$ covers v_4, and $\pitchfork(\!|\Pi_1/vs_{1..n}|\!)$ has $I = \{v_3, v_1, v_4\}$, $A = \{(\{v_3, v_1\}, \{\overline{v_4}, \overline{v_3}\}), (\emptyset, \{v_3\})\}$ and $G = \{\overline{v_4}\}$. Thus, $\pitchfork(\!|\Pi_1/vs_{1..n}|\!) \subseteq \Pi_1$, making $\pitchfork$ a consistent instantiation and so a solution to Problem 1 with inputs Π_1, $\mathcal{P}$ and v_4.*

4.5.1 Finding Instantiations: Practice

We now describe how we obtain a set of isomorphic subproblems of Π that covers the goal variables. We compute a set of instantiations, Δ, of the quotient $\Pi/vs_{1..n}$. Our algorithm first initialises that set of instantiations, $\Delta := \emptyset$. Every iteration of the *main loop* computes a consistent instantiation that covers at least one variable $v \in \mathcal{D}(\Pi.G)$. This is done as follows: we create a new (partial) instantiation $\pitchfork = p_v \mapsto v$, where p_v is the set in $vs_{1..n}$ containing v. Then we determine whether $\pitchfork$ can be completed, to instantiate every set in $vs_{1..n}$, while being consistent at the same time. This determination is achieved by posing the problem in the theory of uninterpreted functions, and using a satisfiability modulo theory (SMT) solver. Our encoding in SMT is constructive, and if a completion of $\pitchfork$ is possible the solver provides it. If successful, we set $\Delta := \Delta \cup \{\pitchfork\}$ and $G := G \setminus \mathcal{R}(\pitchfork)$, and loop. In the case the SMT solver reports failure, the concrete problem cannot be covered by instantiations of the descriptive quotient, and we report failure. In the worst case of $\mathcal{D}(\Pi.G) = \mathcal{D}(\Pi)$, our *main loop* executes $\Sigma_{p \in vs_{1..n}} |p| - |vs_{1..n}| + 1$ times, and hence we have that many instantiations.

Scenarios need not admit a consistent instantiation.

Example 25. *Take Π_2 with*

$$\begin{aligned} \Pi_2.I &= \{\overline{v_1}, \overline{v_2}, v_3\} \\ \Pi_2.\delta &= \{(\{\overline{v_1}\}, \{v_2, \overline{v_3}\}), (\{\overline{v_2}\}, \{v_1, \overline{v_3}\})\} \\ \Pi_2.G &= \{\overline{v_3}, \overline{v_1}, \overline{v_2}\} \end{aligned}$$

Let $vs_{1..n} = \{p_1 = \{v_1, v_2\}, p_2 = \{v_3\}\}$. *The quotient* $\Pi_2/vs_{1..n}$ *has*

$$\begin{aligned}(\Pi_2/vs_{1..n}).I &= \{\overline{p_1}, p_2\}\\ (\Pi_2/vs_{1..n}).\delta &= \{(\{\overline{p_1}\}, \{p_1, \overline{p_2}\})\}\\ (\Pi_2/vs_{1..n}).G &= \{\overline{p_1}, \overline{p_2}\}\end{aligned}$$

There are no consistent instantiations of $\Pi_2/vs_{1..n}$ *because* v_1 *and* v_2 *are in the same equivalence class and also occur together in the same action.*

Our example demonstrates a common scenario in the IPC benchmarks, where a partition $vs_{1..n}$ of $\mathcal{D}(\Pi)$ does not admit a consistent instantiation because variables that occur in the same action coalesce in the same member of $vs_{1..n}$. We resolve this situation by refining the partition produced using CGIP to avoid having such variables coalesce in the same orbit. For a $vs \in vs_{1..n}$, consider the graph $\mathcal{G}(vs)$ with vertices vs and edges $\{\{v_1, v_2\} \mid \exists \pi \in \Pi.\delta \wedge \{v_1, v_2\} \subseteq \mathcal{D}(\pi) \wedge v_1 \neq v_2\}$. The *chromatic number* N of $\mathcal{G}(vs)$ gives us the number of colours needed to colour the corresponding vertices $\hat{vs}$ in the PDG. Where two variables occur in the same action, their vertices in $\hat{vs}$ are coloured differently. For every $vs \in vs_{1..n}$, we use an SMT solver to calculate the required chromatic numbers N and graph colourings for $\mathcal{G}(vs)$, then we colour the corresponding vertices in the PDG according to the computed N-colouring of $\mathcal{G}(vs)$. Lastly, we again pass the PDG to a CGIP after it is recoloured. In 4 benchmark sets the thus-revised partition admits a consistent instantiation where the initial partition does not. Although the *chromatic number* problem is NP-complete, this step is not a bottleneck in practice because the size of the instances that need to be solved is bounded above by the largest number of literals stated in any action.

4.5.2 Finding Instantiations: Theory

We first develop a theoretical bound on the number of required instantiations—by treating abstract covering transversals—that is tight compared to our pragmatic solution above. To characterise the complexity of our instantiation problem, we then study the general problem of computing covering transversals.

Theorem 15 (B. McKay, 2014)**.** *Let* $-$ *be a group acting on a set* V *(e.g.,* $\mathcal{D}(\Pi)$*). Suppose* $\pitchfork$ *is a transversal of* O*, a set of orbits induced by the action of* $-$ *on* V*. Take* $S = \Sigma_{o \in O}|o|$ *and* $M = \mathsf{max}_{o \in O}|o|$*. Where* $\bullet$ *is composition, there will always be a set of transversals* $\mathcal{T}$ *with size* $\leq M \ln(S)$ *such that*

(i) Each element $\pitchfork' \in \mathcal{T}$ *satisfies* $\pitchfork' = \sigma \bullet \pitchfork$ *for some* $\sigma \in -$*, and*

(ii) For every $u_1 \in V$*, it has an element* $\pitchfork'$ *that covers* u_1*.*

Proof. Take $N = M \ln(S)$ and let H be a subset of $-$ obtained by drawing N permutations of V independently at random with replacement. For any orbit $o \in O$, the probability for a $v \in o$ is not in $\mathcal{R}(\sigma \bullet \pitchfork)$ for a randomly drawn $\sigma \in H$ is $1 - 1/|o|$. Let $\mathcal{T} = \{\sigma \bullet \pitchfork \mid \sigma \in H\}$ and $R = \bigcup(\mathcal{R}(\!|\mathcal{T}|\!))$. Drawing N times from $-$ to construct H, the probability that $v \notin R$ is $(1 - 1/|o|)^N$. Consider the random quantity $Z = |V \setminus R|$ with expected value $\mathbb{E}(Z) = \Sigma_{o \in O}|o|(1 - 1/|o|)^N$. Since $1 - x < e^{-x}$ for $x > 0$, we obtain $\mathbb{E}(Z) < \Sigma_{o \in O}|o|e^{(-N/|o|)} \leq Se^{-N/M} = Se^{-\ln(S)}$. From $xe^{-\ln(x)} = 1$, it follows that $\mathbb{E}(Z) < 1$. Since $Z \in \mathbb{N}$, then probability $Z = 0$ is more than 0, and thus N transversals of O suffice to cover V. □

We conjecture that a much smaller number of transversals is actually required, and in all our experimentation have found that the following conjecture is not violated:

Conjecture 1. *Let $-$ be a group acting on a set V (e.g., $\mathcal{D}(\Pi)$). Suppose $\pitchfork$ is a transversal of O, a set of orbits induced by the action of $-$ on V. Take $M = \mathsf{max}_{o \in O}|o|$. Where $\bullet$ is composition, there will always be a set of transversals $\mathcal{T}$ with size $\leq M$ such that*

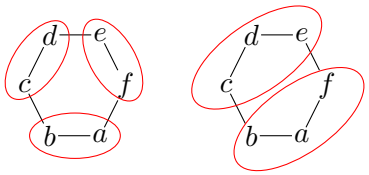
(i) Each element $\pitchfork' \in \mathcal{T}$ satisfies $\pitchfork' = \sigma \bullet \pitchfork$ for some $\sigma \in -$, and

(ii) For every $u_1 \in V$, it has an element $\pitchfork'$ that covers u_1.

We are left to formulate and study a general problem of instantiation, treating it as one of finding graph transversals.

Definition 33 (Consistent Graph Transversals)**.** *For a graph $\mathcal{G}$ and a partition P of $V(\mathcal{G})$, a transversal $\pitchfork$ of P is consistent with $\mathcal{G}$ when an edge between p_1 and p_2 in $E(\mathcal{G}/P)$ exists iff there is an edge between $\pitchfork(p_1)$ and $\pitchfork(p_2)$ in $E(\mathcal{G})$.*

Example 26. *Take $\mathcal{G}$ to be the hexagon that we have illustrated twice below in order to depict partitions $P_1 = \{\{a,b\},\{c,d\},\{e,f\}\}$ and $P_2 = \{p' = \{a,b,f\}, p'' = \{c,d,e\}\}$ on the left and right respectively.*



The vertices of $\mathcal{G}/P_1$ (a 3-clique) and $\mathcal{G}/P_2$ (a 2-clique) are indicated above by red outlines. There is no consistent transversal of P_1 (LHS) because there is no 3-clique subgraph of $\mathcal{G}$ with one vertex from each set in P_1. For $\mathcal{G}/P_2$ (RHS), $\pitchfork_1 = \{p' \mapsto f, p'' \mapsto e\}$ is a transversal of P_2 consistent with $\mathcal{G}$, because the subgraph of $\mathcal{G}$ induced by $\{e,f\}$ is a 2-clique with one vertex from each of p' and p''.

A transversal of a well-formed $\hat{P}$ consistent with $\mathcal{G}(\Pi)$ is isomorphic to an instantiation of $\Pi/vs_{1..n}$ consistent with Π. Accordingly, Problem 1 is an instance of the following.

Problem 2. *Given $\mathcal{G}$, a partition P of $V(\mathcal{G})$ and $u_1 \in V(\mathcal{G})$, is there a consistent transversal of P which covers u_1?*

We now derive the complexity of Problem 2. We first show that the following problem is NP-complete, and use that result to show that Problem 2 is also NP-complete.

Problem 3. *Given graph $\mathcal{G}$ and a partition P of $V(\mathcal{G})$, is there a transversal of P consistent with $\mathcal{G}$?*

Lemma 7. *Problem 3 is NP-complete.*

Proof. Membership in NP is given because a transversal's consistency can clearly be tested in polynomial-time. We then show the problem is NP-hard by demonstrating a polynomial-time reduction from SAT. Consider SAT problems given by formulae φ in conjunctive normal form. Assume every pair of clauses is mutually satisfiable—i.e., for clauses $c_1, c_2 \in \varphi$, for two literals $\ell_1 \in c_1$ and $\ell_2 \in c_2$ we have $\ell_1 \neq \neg\ell_2$ (when this assumption is violated, unsatisfiability can be decided in polynomial-time). Consider the graph $\mathcal{G}(\varphi)$, where $V(\mathcal{G}(\varphi)) = \{u_{1\ell c} \mid c \in \varphi,\ \ell \in c\}$,

and $E(\mathcal{G}(\varphi)) = \{\{u_{1\ell_1 c_1}, u_{1\ell_2 c_2}\} \mid c_1, c_2 \in \varphi, \ell_1 \in c_1, \ell_2 \in c_2, \ell_1 \neq \neg\ell_2\}$. Now let $P = \{\{u_{1\ell c} \mid \ell \in c\} \mid c \in \varphi\}$. Note that P is a partition of $V(\mathcal{G}(\varphi))$ and every set in P corresponds to a clause in φ. Because all the clauses in φ are mutually satisfiable, the quotient $\mathcal{G}(\varphi)/P$ is a clique. Now we prove there is a model for φ iff there is a transversal of P consistent with $\mathcal{G}(\varphi)$. ($\Rightarrow$) A model $\mathcal{M}$ has the property $\forall c \in \varphi.\ \exists \ell \in c.\ \mathcal{M} \models \ell$. Due to the correspondence between sets in P and clauses in φ, a transversal $\pitchfork$ of P can be constructed by selecting one satisfied literal from each clause. Based on a model, $\pitchfork$ will never select conflicting literals. All members of $\mathcal{R}(\pitchfork)$ are pairwise connected, so $\pitchfork$ is consistent with $\mathcal{G}(\varphi)$ as required. ($\Leftarrow$) By definition $\pitchfork$ is consistent with $\mathcal{G}(\varphi)$, so the subgraph of $\mathcal{G}$ induced by $\mathcal{R}(\pitchfork)$ is a clique. Let $\mathcal{L}$ be literals corresponding to $\mathcal{R}(\pitchfork)$, and note that its elements are pairwise consistent. A model $\mathcal{M}$ for φ is constructed by assigning u_1 to $\top$ where u_1 occurs positively in $\mathcal{L}$, and to $\bot$ otherwise. □

Theorem 16. *Problem 2 is NP-complete.*

Proof. Consistency of a transversal $\pitchfork$ is clearly polynomial-time testable, as is the coverage condition that $u_1 \in \mathcal{R}(\pitchfork)$. Thus, we have membership in NP. NP-hardness follows from the following reduction from Problem 3 (P3). Taking P and $\mathcal{G}$ as inputs to P3, construct the graph $\mathcal{G}'$, where: $V(\mathcal{G}') = V(\mathcal{G}) \cup \{u_1\}$, u_1 an auxiliary vertex; and $E(\mathcal{G}') = E(\mathcal{G}) \cup \{\{u_1, u_2\} \mid u_2 \in V(\mathcal{G})\}$. There is a solution to P3 with inputs $\mathcal{G}$ and P iff P2 is soluble with inputs $\mathcal{G}'$, $P \cup \{\{u_1\}\}$ (= P') and u_1. ($\Rightarrow$) If $\pitchfork$ is a transversal of P consistent with $\mathcal{G}$, then $\pitchfork \cup \{\{u_1\} \mapsto u_1\} (= \pitchfork')$ is a transversal of P'. As u_1 is fully connected, $\pitchfork'$ is consistent with $\mathcal{G}'$. $\pitchfork'$ is a solution to P2 because $u_1 \in \mathcal{R}(\pitchfork')$. ($\Leftarrow$) If $\pitchfork'$ is a solution to P2 with inputs $\mathcal{G}'$, P' and u_1, then $\pitchfork'$ is a transversal of P', and also P. All edges in $E(\mathcal{G}')$ are in $E(\mathcal{G})$, with the exception of those adjacent u_1, and since $\pitchfork'$ is consistent with $\mathcal{G}'$, then $\pitchfork'$ is consistent with $\mathcal{G}$. Thus, $\pitchfork'$ solves P3. □

In concluding, we note that the NP-hard canonicalisation problem—the optimisation problem posed at each state encountered by orbit search—is not known to be in NP. Our above results imply that exploitation of symmetry via instantiation poses a decision problem in NP.

4.6 Concrete Plan from Quotient Plan

Having computed isomorphic subproblems Δ that cover the goals of Π, a concrete plan is a concatenation of plans for members of Δ. However, this is not always straightforward.

Example 27. *Take Π_1, $vs_{1..n}$ and $\pitchfork$ from Example 24. Note $\pitchfork' = \{p_1 \mapsto v_3, p_2 \mapsto v_2, p_3 \mapsto v_5\}$ is also an instantiation of $\Pi_1/vs_{1..n}$ consistent with Π_1. Observe that $\{\pitchfork(\!|\Pi_1/vs_{1..n}|\!), \pitchfork'(\!|\Pi_1/vs_{1..n}|\!)\}$ covers the concrete goal $G = \{\overline{v_4}, \overline{v_5}\}$. A plan for $\Pi_1/vs_{1..n}$ is $\overrightarrow{\pi}' = (\{p_1, p_2\}, \{\overline{p}_3, \overline{p}_1\})$ and its two instantiations are $\pitchfork(\!|\overrightarrow{\pi}'|\!) = (\{v_3, v_1\}, \{\overline{v_4}, \overline{v_3}\})$ and $\pitchfork'(\!|\overrightarrow{\pi}'|\!) = (\{v_3, v_2\}, \{\overline{v_5}, \overline{v_3}\})$. Concatenating $\pitchfork(\!|\overrightarrow{\pi}'|\!)$ and $\pitchfork'(\!|\overrightarrow{\pi}'|\!)$ in any order does not solve Π_1 because both plans require v_3 initially, but do not establish it. To overcome this issue in practice, before we solve $\Pi_1/vs_{1..n}$ we augment its goal with the assignment $p_1 \mapsto \top$.*

We now give conditions under which concatenation is valid, and detail the quotient-goal augmentation step we use to ensure validity in practice. We will use the notion of projection, writing $Y\lfloor^X$ to mean $Y\lfloor_{\mathcal{D}(X)}$, for the common case where we wish to project with respect to all the variables in the domain of an object.

Definition 34 (Needed Assignments, Preceding Problem)**.** *Needed assignments, $\mathcal{N}(\Pi)$, are assignments in the preconditions of actions and goal conditions that also occur in I, i.e., $\mathcal{N}(\Pi) =$*

$(\mathsf{pre}(\delta) \cap I) \cup (G \cap I)$. *Problem* Π_1 *is said to precede* Π_2*, written* $\Pi_1 \lhd \Pi_2$*, iff*

$$G_1\lfloor^{\mathcal{N}(\Pi_2)} = (I_2\lfloor^{\Pi_1})\lfloor^{\mathcal{N}(\Pi_2)} \ \wedge \ G_1\lfloor^{\Pi_2} = G_2\lfloor^{\Pi_1}$$

In words,

(i) The needed assignments of Π_2 *which a plan for* Π_1 *could modify occur in* G_1*, and*

(ii) G_2 *contains all the assignments in* G_1 *which a plan for* Π_2 *might modify.*

Example 28. *Consider* Π_1 *and* Π_2 *from Examples 24 and 25 respectively.* $\mathcal{N}(\Pi_1) = \{v_3, v_1, v_2\}$ *and* $\mathcal{N}(\Pi_2) = G_1 = \{\overline{v_4}, \overline{v_5}\}$*. Since* $G_1\lfloor^{\mathcal{N}(\Pi_2)} = G_1$*,* $(I_2\lfloor^{\Pi_1})\lfloor^{\mathcal{N}(\Pi_2)} = G_1$*,* $G_1\lfloor^{\Pi_2} = G_1$ *and* $G_2\lfloor^{\Pi_1} = G_1$*, we have* $\Pi_1 \lhd \Pi_2$*.*

Writing $\vec{\pi}_i{}^\frown\vec{\pi}_j$ for concatenation of plans $\vec{\pi}_i, \vec{\pi}_{i+1} \ldots \vec{\pi}_j$, a simple inductive argument gives the following:

Lemma 8. *Let* $\Pi_1 \ldots \Pi_N$ *be a set of problems satisfying* $\Pi_j \lhd \Pi_k$ *for all* $j < k \leq N$*. If for all* $1 \leq i \leq N$*, state* x *satisfies* $I_i \subseteq x$*,* $\vec{\pi}_i$ *solves* Π_i *and* $\mathsf{sat\text{-}pre}(\mathcal{N}(\Pi_i), \vec{\pi}_i)$ *holds, then* $\mathsf{ex}(x, \vec{\pi}_1{}^\frown\vec{\pi}_N)\lfloor^{G_i} = G_i$*.*

Proof. By induction on N. First, note $\mathsf{ex}(x, \vec{\pi}_i)\lfloor^{G_i} = \mathsf{ex}(I_i, \vec{\pi}_i)\lfloor^{G_i}$. Our proof is by induction on N, and the base case $N = 1$ is trivial. We now show that if the theorem is true for N, then the theorem also holds for $N + 1$. Below, let $x' = \mathsf{ex}(x, \vec{\pi}_1{}^\frown\vec{\pi}_N)$.

Note $\mathcal{N}(\Pi_{N+1}) \subseteq x'$, because for each i :

(i) assignments modified by $\vec{\pi}_i$ whose variables are the subject of a needed assignment from Π_{N+1} are in G_i, because $\Pi_i \lhd \Pi_{N+1}$ and thus $G_i\lfloor^{\mathcal{N}(\Pi_{N+1})} = (I_{N+1}\lfloor^{\Pi_i})\lfloor^{\mathcal{N}(\Pi_{N+1})}$

(ii) the goals of any Π_i are in x' because the inductive hypothesis gives $x'\lfloor^{G_i} = G_i$

(iii) any assignments in I_{N+1} not modified by exeuction $\vec{\pi}_1{}^\frown\vec{\pi}_N$ from x are in x', because $I_{N+1} \subseteq x$.

Because $\vec{\pi}_{N+1}$ is a plan for Π_{N+1}, $\mathsf{ex}(x', \vec{\pi}_{N+1})\lfloor^{G_{N+1}} = G_{N+1}$.

Since $\Pi_i \lhd \Pi_{N+1}$, any assignment in x' that is in some G_i is also in $\mathsf{ex}(x', \vec{\pi}_{N+1})$. The inductive hypothesis gives $G_i = x'\lfloor^{G_i}$ thus $G_i = \mathsf{ex}(x', \vec{\pi}_{N+1})\lfloor^{G_i}$. Because $\mathsf{ex}(x, \vec{\pi}_1{}^\frown\vec{\pi}_{N+1}) = \mathsf{ex}(x', \vec{\pi}_{N+1})$ we have $\mathsf{ex}(x, \vec{\pi}_1{}^\frown\vec{\pi}_{N+1})\lfloor^{G_i} = G_i$. □

Take problem Π and a set of problems $\mathbf{\Pi}$, we say that $\mathbf{\Pi}$ covers Π iff $\forall g \in G.\ \exists \Pi' \in \mathbf{\Pi}.\ g \in G'$ and $\forall \Pi' \in \mathbf{\Pi}.\ \Pi' \subseteq \Pi$. I.e., every goal from Π is stated in one or more members of $\mathbf{\Pi}$, a set of subproblems of Π.

Theorem 17. *Consider a set* $\Pi_1 \ldots \Pi_N$ *of problems that covers* Π*, satisfying* $\Pi_j \lhd \Pi_k$ *for all* $j < k \leq N$*. For* $1 \leq i \leq N$ *let* $\vec{\pi}_i$ *be a plan for* Π_i*, then* $\mathsf{rem\text{-}condless}(\mathcal{N}(\Pi_1), \vec{\pi}_1)^\frown\mathsf{rem\text{-}condless}(\mathcal{N}(\Pi_N), \vec{\pi}_N)$ *is a plan for* Π*, where for an action sequence* $\vec{\pi}$ *and a state* x*,* $\mathsf{rem\text{-}condless}(x, \vec{\pi})$ *denotes* $\vec{\pi}$*, but without the actions whose preconditions are not satisfied during executing* $\vec{\pi}$ *from* x*.*

This theorem follows directly from the fact that the set of problems covers Π and from Lemma 8.

We now address the question: When can plans for a set of isomorphic subproblems be concatenated to provide a concrete plan? We provide sufficient conditions in terms of the concepts of *common* and *sustainable* variables.

Definition 35 (Common Variables)**.** *For a set of instantiations Δ, the set of* common variables*, written $\bigcap_v \Delta$, comprises variables in $\bigcup_{\pitchfork \in \Delta} \mathcal{R}(\pitchfork)$ that occur in the ranges of more than one member of Δ.*

Definition 36 (Sustainable Variables)**.** *A set of variables vs is sustainable in a problem Π iff $I\downarrow_{vs} = G\downarrow_{vs}$.*

Theorem 18. *Take problem Π, partition $vs_{1..n}$ of $\mathcal{D}(\Pi)$ where the quotient $\Pi/vs_{1..n} (= \Pi')$ is well-defined, with solution $\vec{\pi}'$, and consistent instantiations Δ. Suppose $\{\pitchfork(\!|\Pi'|\!) \mid \pitchfork \in \Delta\} (= \mathbf{\Pi})$ covers Π, and $\mathcal{Q}(\!|\bigcap_v \Delta|\!) \cap \mathcal{D}(\mathcal{N}(\Pi'))$—i.e., based on Definition 30, the orbits of common variables from needed assignments—are sustainable in Π'. Then any concatenation of the plans $\{\textsf{rem-condless}(\mathcal{N}(\pitchfork(\!|\Pi'|\!)), \pitchfork(\!|\vec{\pi}'|\!) \mid \pitchfork \in \Delta\}$ solves Π.*

Proof. Identify the elements in Δ by indices in $\{1..|\Delta|\}$. Let $k \in \{1..|\Delta|\}$ and $\Pi_k = \pitchfork_k(\!|\Pi'|\!)$, and note $\mathcal{R}(\pitchfork_k) = \mathcal{D}(\Pi_k)$. Take $\pitchfork_i, \pitchfork_j \in \Delta$, where $i, j \in \{1..|\Delta|\}$ and $i \neq j$. We first show that $\Pi_i \lhd \Pi_j$. For any $vs \in vs_{1..n}$, $\pitchfork_i(vs) = \pitchfork_j(vs)$ if $\pitchfork_i(vs) \in \mathcal{R}(\pitchfork_i) \cap \mathcal{R}(\pitchfork_j)$. Therefore, $I_i\downarrow^{\Pi_j} = I_j\downarrow^{\Pi_i}$ and $G_i\downarrow^{\Pi_j} = G_j\downarrow^{\Pi_i}$, providing the right conjunct in Definition 34. For the left conjunct, note $\mathcal{D}(\mathcal{N}(\Pi_j)) \subseteq \mathcal{D}(\Pi_j)$ and $\mathcal{D}(\Pi_i) \cap \mathcal{D}(\Pi_j) \subseteq \bigcap_v \Delta$. The sustainability premise—$\mathcal{Q}(\bigcap_v \Delta) \cap \mathcal{D}(\mathcal{N}(\Pi'))$ is sustainable in Π'—then provides that $\mathcal{D}(\Pi_i) \cap \mathcal{D}(\mathcal{N}(\Pi_j))$ is sustainable in Π_i—i.e., $I_i\downarrow^{\mathcal{N}(\Pi_j)} = G_i\downarrow^{\mathcal{N}(\Pi_j)}$. Thus $G_i\downarrow^{\mathcal{N}(\Pi_j)} = (I_j\downarrow^{\Pi_i})\downarrow^{\mathcal{N}(\Pi_j)}$, and we conclude $\Pi_i \lhd \Pi_j$. Since a plan $\pitchfork_k(\!|\vec{\pi}'|\!)$ solves $\pitchfork_k(\!|\Pi'|\!)$, $(\pitchfork_k(\!|\Pi'|\!)).I \subseteq \Pi.I$, therefore as per Theorem 17 a solution to Π is $\textsf{rem-condless}(\mathcal{N}(\pitchfork_1(\!|\Pi'|\!)), \pitchfork_1(\!|\vec{\pi}'|\!)) \frown \textsf{rem-condless}(\mathcal{N}(\pitchfork_N(\!|\Pi'|\!)), \pitchfork_N(\!|\vec{\pi}'|\!))$. □

In practice we take $vs^* = \mathcal{Q}(\!|\bigcap_v \Delta|\!) \cap \mathcal{D}(\mathcal{N}(\Pi/vs_{1..n}))$, and augment the goal of the quotient $\Pi/vs_{1..n}$ by adding $(\Pi/\mathcal{P}).I\downarrow_{vs^*}$. Call the resulting problem Π^q and its solution $\vec{\pi}^q$. Theorem 18 shows that any concatenation of the plans $\{\textsf{rem-condless}(\pitchfork(\!|\Pi^q|\!), \pitchfork(\!|\vec{\pi}^q|\!)) \mid \pitchfork \in \Delta\}$ solves Π. Thus, the proposed algorithm is sound.

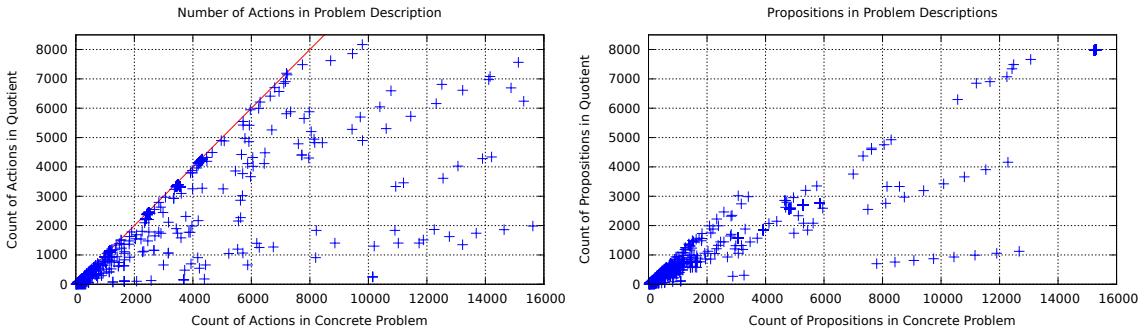
Example 29. *Take Π_1, $vs_{1..n}$, $\pitchfork$ and $\pitchfork'$ from Example 27. There is one common variable, $\{v_3\} = \bigcap_v \Delta$ from the orbit $\{p_1\} = \mathcal{Q}(\!|\bigcap_v \Delta|\!)$. Here $\mathcal{N}(\Pi_1/vs_{1..n}) = \{p_1, p_2\}$, and the orbit of the common needed variable is $\{p_1\} = \mathcal{Q}(\!|\bigcap_v \Delta|\!) \cap \mathcal{N}(\Pi_1)/vs_{1..n}$. To solve Π_1 via solving $\Pi_1/vs_{1..n}$, we augment the goal $(\Pi_1/vs_{1..n}).G$ with the assignment to p_1 in $(\Pi_1/vs_{1..n}).I$. The resulting problem, Π_1^q, is equal to $\Pi_1/vs_{1..n}$ except that it has the goals $\Pi_1^q.G = \{p_1, \overline{p}_3\}$. A plan for Π_1^q is $\vec{\pi}^q = (\{p_1, p_2\}, \{\overline{p}_3, \overline{p}_1\})(\emptyset, \{p_1\})$, and two instantiations of it are $\pitchfork(\!|\vec{\pi}^q|\!) = (\{v_3, v_1\}, \{\overline{v_4}, \overline{v_3}\})(\emptyset, \{v_3\})$ and $\pitchfork'(\!|\vec{\pi}^q|\!) = (\{v_3, v_2\}, \{\overline{v_5}, \overline{v_3}\})(\emptyset, \{v_3\})$. Concatenating $\pitchfork(\!|\vec{\pi}^q|\!)$ and $\pitchfork'(\!|\vec{\pi}^q|\!)$ in any order solves Π_1.*

4.7 Experimental Results

Implemented in C++,[1] our approach uses: NAUTY\TRACES by McKay and Piperno (2014) to calculate symmetries; Z3 by De Moura and Bjørner (2008) to find isomorphic subproblems; and the initial-plan search by LAMA by Richter and Westphal (2010) as the *base planner*. We limit *base planner* runtimes to 30 minutes.

By running our algorithm, we obtained a set of benchmarks with soluble descriptive quotients whose solutions can be instantiated to concrete plans. That set includes 439 problems, from 16 IPC benchmarks and 4 benchmarks from Porco et al. (2013). In all our experimentation we identified 120 instances where we were able to confirm that the descriptive quotient does not have a solution

[1] Found at the public repository `bitbucket.org/MohammadAbdulaziz/planning.git` in the directory `codeBase`.



(a) (b)

Figure 4.4: (a) Scatter-plot comparing the number of *actions* in problem posed by descriptive quotient *vs.* concrete problem, with red line plotting $f(x) = x$. (b) Scatter-plot comparing the number of *state variables* in problem posed by descriptive quotient *vs.* concrete problem, with red line plotting $f(x) = x$.

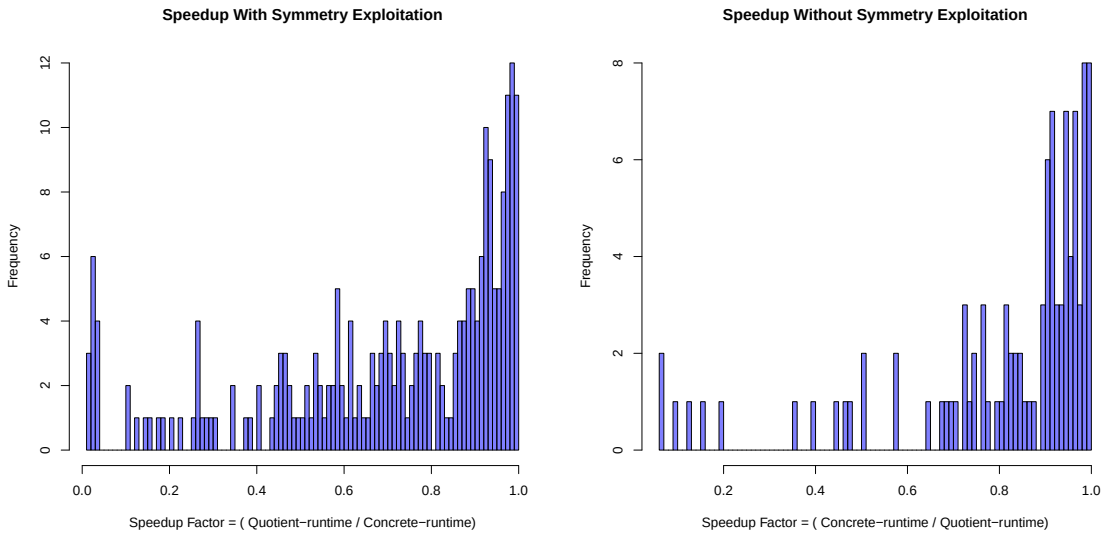
and the concrete problem does. Figure 4.4 plots the sizes in terms of both, the number of actions and the number of state variables, of concrete and quotient problem descriptions. The plotted data shows that the descriptive quotient can be much smaller than its concrete counterpart. In just over 15% of instances the quotient has less than half the number of actions. Here, we also analyzed what aspects of our approach are computationally expensive. In 79% of cases 99% of the runtime of our approach is executing the base planner. In 96% of cases 95% of the runtime of our approach is executing the base planner. Overall, 3% of time is spent in instantiation and finding chromatic numbers.

We examined where our approach is comparatively scalable and fast compared to the base planner. For 430 instances where the base planner and our approach are successful, Figure 4.5a displays the speedup factors where planning *via* the descriptive quotient is comparatively fast. Overall, for 68% of instances our approach is comparatively fast, and in 15% planning *via* the quotient is at least twice as fast. With few exceptions, instances where our approach is at least twice as fast are from GRIPPER, HIKING, MPRIME, MYSTERY, PARCPRINTER, PIPESWORLD, TPP and VISITALL. In 5 problems from HIKING, 2 from PARCPRINTER, 1 from TETRIS and 1 from KCOLOURABILITY, the base planner cannot solve the concrete problem, but can solve the quotient. Planning *via* the quotient can also be slow, primarily due to the extra cost of finding symmetries,[2] and because LAMA is heuristic and occasionally finds that the quotient poses a more challenging problem. Figure 4.5b provides the dual to Figure 4.5a, showing cases where planning directly for the concrete problem is comparatively fast. This is the case in 32% instances, and indeed in 2% our approach is at least twice as slow.

Finally, it is worth highlighting the difference between searching for a plan in the state space of the descriptive quotient versus approximate orbit search—i.e., searching for a plan in an approximation of the quotient state space—as is done in the state-of-the-art techniques for exploiting symmetries in planning.[3] In comparing those approaches, we measure the number of states expanded using a breadth-first search. We consider the IPC instances GRIPPER-20 and MPRIME-21, the largest instances from domains GRIPPER and MPRIME reported solved by Pochter et al. (2011) using breadth-first search. Those authors report the number of states expanded to be 60.5K and 438K, respectively. Using the same search to solve the problem posed by our descriptive quotient,

[2] If symmetries are given as part of the problem description, or if one resorts to more heuristic methods of discovering them, such as those described by Guere and Alami (2001); Fox and Long (2002), this burden is relieved.

[3] Such a comparison is admittedly unfair, because the problem posed by a descriptive quotient is not a bisimulation of the concrete problem.



(a) (b)

Figure 4.5: (a) Plots histogram of speedup factors experienced when planning **with** symmetry exploitation, reporting only for instances where planning via the quotient is faster, and only for instances where the base planner takes > 5 seconds. (b) Plots histogram of speedup factors experienced when planning **without** symmetry exploitation, reporting only for instances where planning for the concrete problem directly is faster, and only for instances where the base planner takes > 5 seconds.

the base planner expands 6 and 203K states, respectively.

4.8 Conclusions and Future Work

We introduced a new compositional approach for AI planning that is based on a new abstraction, the *descriptive quotient*, as well as exploiting state variable symmetries. A concrete plan is obtained by concatenating instantiations of the plan for the descriptive quotient. Compared to mainstream symmetry breaking techniques this has multiple advantages. Firstly, we do not have to solve the NP-hard constructive orbit problem for every orbit of symmetric states. Secondly, the size of the descriptive quotient's state space can be exponentially smaller than that of the quotient system, in which a plan is searched for, by other symmetry breaking algorithms. However, effectively this approach breaks a specific type of symmetries: *repetitive symmetries*, which is when the planning problem is constituted of multiple isomorphic subproblems. Also it is incomplete, as the descriptive quotient may not have a plan, while the concrete problem is soluble. Nonetheless, we experimentally show that this approach is successful in 70% of standard international planning competition benchmarks with symmetries, where it brings significant reduction in plan search time.

We suppose a fruitful research direction is exploring descriptive quotients: to develop heuristics, to characterise tractable classes, and to develop bounding methods. Also an interesting question is how much does the quality (in terms of length and cost) of the plans computed by this technique compare to that of plans computed by other symmetry breaking techniques. We suppose that this question is very strongly related to either proving or disproving Conjecture 1

Chapter 5

Formalisation

In order to formally verify the correctness of our compositional algorithms we build a library of formal proofs on factored transition systems in the Higher Order Logic (HOL) interactive theorem prover HOL4 by Slind and Norrish (2008). There is a rich body of previous formalisation of concepts related to transition systems in different logics and theorem proving systems. A lot of that work (e.g. Paulson (2015) and Esparza et al. (2013)) focuses on classical textbook results on finite automata and reachability of states within those automata in the setting of model checking LTL formulae. However, we believe that our work is the first to formalise transition systems in their factored representation, and to focus on the topological properties of their state spaces, in addition to reachability within them. In this chapter we discuss our formalisation of factored transition systems and how we used it to formally verify some of the main algorithms we discussed earlier. The general organisation of the library is shown in Figure 5.1 and the size and description of each theory is in Table 5.1.

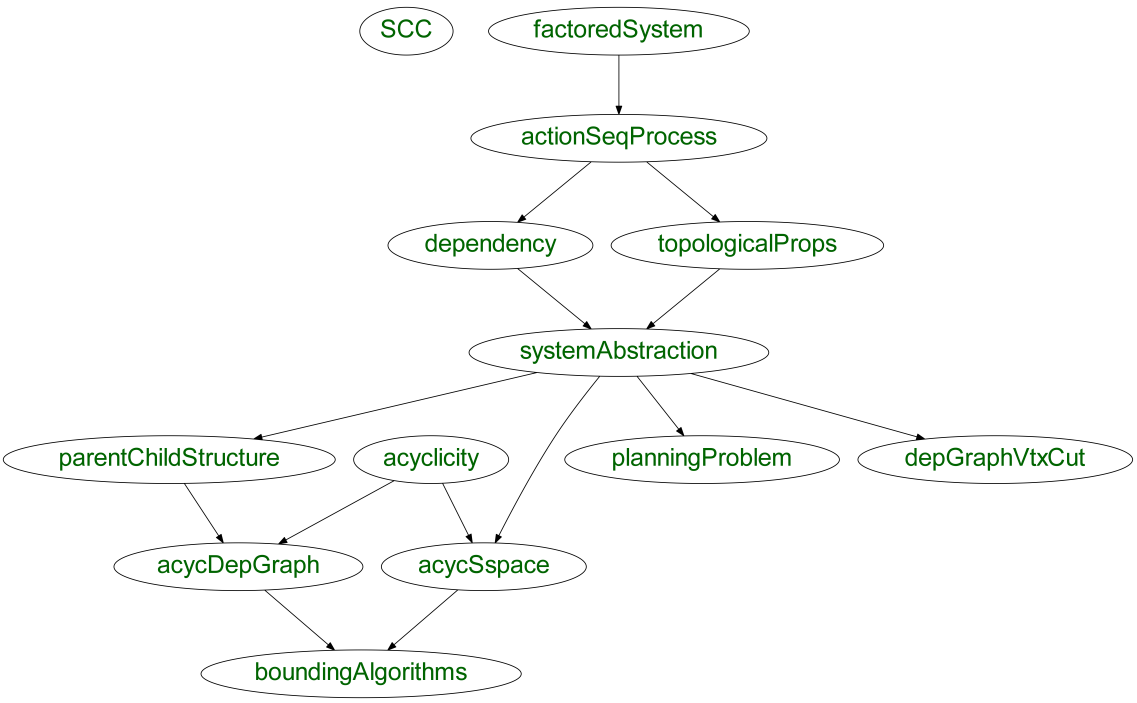


Figure 5.1: The organisation of the different theories concerning factored transition systems. An edge from one theory to another indicates the dependence of the latter on the former.

Theory	Size (LOC)	Description
actionSeqProcess	432	Theory on different functions that process action sequences, e.g. remove actions without effects.
acycDepGraph	1587	Theory related to the top-down algorithm.
acyclicity	130	Theory regarding acyclic digraphs represented as topologically sorted lists.
acycSspace	346	Theory related to the S-algorithm that exploits acyclicity in the state space.
boundingAlgorithms	644	Theory related to the hybrid algorithm.
dependency	83	Basic results related to variable dependency.
depGraphVtxCut	582	Results related to vertex cutting in the dependency graph.
factoredSystem	1335	Basic results and definitions related to factored transition systems.
instantiation	1007	Results related to instantiation of planning problems.
invariantsPlusOne	300	Theory related to the cardinality of sets of states with SAS+ like invariant properties.
invariantStateSpace	121	A bound on the size of a state space of a planing problem that has a SAS+ representation derived invariant property.
parentChildStructure	1304	Formalisation of bound compositionality in the parent-child structure and the stitching function.
planningProblem	994	Formalisation of planning problems on top of factored transition systems.
prodValidtd	451	Verification of the fact that the product of some projections' traversal diameters is an upper bound on the concrete system's traversal diameter.
SCC	46	Basic results related to strongly connected components.
SCCsystemAbstraction	1384	Results related to computing system abstractions based on computing strongly connected components of dependency graphs.
stateSpaceProduct	358	Defining the state space product operator and some basic facts about it.
systemAbstraction	1135	Definition of different abstraction concepts like projetion and snapshotting, and verifying some basic facts about them.
topologicalProps	879	Definition of the diameter, sublist diameter, recurrence diameter, and the traversal diameter and verifying basic theory about them.

Table 5.1: A table showing the sizes of different theories and their content.

HOL4 Notation All statements appearing with a turnstile ($\vdash$) are HOL4 theorems, automatically pretty-printed to LaTeX.

5.1 Related Work

Interactive theorem proving systems (e.g. Gordon and Melham (1993); Paulson (1994); Bertot and Castéran (2004)) have been used in mechanisation and verification in very diverse fields. Fundamental results in different branches of mathematics have been verified. For example Dufourd (2009) formalised a discretised form of the Jordan Curve theorem. In mathematical analysis, Harrison (2007) formalised Cauchy's integral theorem, and Abdulaziz and Paulson (2016) formalised Green's theorem. In Bezem and Hendriks (2008) a coherent logic based theorem prover was implemented and used to mechanise Hessenberg's theorem. Also the work by McKinna and Pollack (1999) shows a mechanisation of a simplified version of beta reduction in Lambda Calculus. Gamboa (2009) presents a mechanisation of the Powerlist Algebra, which is a data structure used for representing concurrent recursive algorithms as well as proving their correctness. Synthetic domain theory was formalised in Taylor (1991).

Many sohpisticated algorithms have also been mechanised in the literature. Techniques for theorem proving such as SLD-resolution have been mechanised in Jaume (1999), where the theorem prover Coq was used. The work in Kamareddine and Qiao (2003) presents a formalisation of the termination proof of two styles of substitution calculi. Manolios and Vroon (2005) provided libraries for doing basic tasks in ordinal arithmetic, and mechanised them using ACL2. For propositional satisfiability, Marić (2009); Blanchette et al. (2016) formalised SAT solvers from the basic DPLL solver up to modern solvers which have features such as clause learning or backjumping and other features.

Different work, like Klein et al. (2009); Hales et al. (2011); Gonthier (2008) , show that the scale of mechanisation projects is continuously growing. Bauer et al. (2013) provide a mechanisation of data flow analysis for the purpose of building more secure software systems. In Gonthier (2008) and Hales et al. (2011), the proof Kepler's conjecture and the four colour theorem, respectively, are verified, which are major mathematical results.

Most relevant to our work are formalisations of automata theory and their reachability properties, in the context of model checking. Textbook results in automata theory were formalised in many approaches. For example, the Myhill-Nerode theorem was formalised in intuitionistic logics by Constable et al. (2000) and Doczkal et al. (2013). It was also formalised by Paulson (2015) in Isabelle/HOL. He used hereditarily finite sets as the type of states, unlike our formalisation which uses finite maps. Furthermore, Wu et al. (2011) formally prove that theorem using a formalisation of regular expressions. Results and algorithms related to reachability in automata were formalised by Sprenger (1998); Schimpf et al. (2009); Esparza et al. (2013), where the main goal of those authors was to obtain formally verified model checking algorithms and implementations.

5.2 Factored Transition Systems in HOL4

We formalise propositionally factored transition systems by first defining states to be finite maps $\alpha \mapsto \beta$ from polymorphic type α to polymorphic type β.[1] An action is a pair of such states $(\alpha \mapsto \beta) \times (\alpha \mapsto \beta)$, and a factored transition system is a set of actions $(\alpha \mapsto \beta) \times (\alpha \mapsto \beta) \to \mathit{bool}$. We define the set of valid states and valid action sequences as follows:

[1] If we would model STRIPS or SMV transition systems, β would be instantiated with *bool*.

HOL4 Definition 1 (Factored System)**.**

$$\mathbb{U}(\delta) \texttt{ = } \{x \texttt{ | } \mathcal{D}(x) \texttt{ = } \mathcal{D}(\delta)\}$$

$$\delta^* \texttt{ = } \{\overrightarrow{\pi} \texttt{ | set } \overrightarrow{\pi} \texttt{ } \subseteq \texttt{ } \delta\}$$

Action execution and action sequence execution are defined as follows:

HOL4 Definition 2 (Execution)**.**

$$\textsf{state-succ } x \texttt{ } (p,e) \texttt{ = if } p \texttt{ } \sqsubseteq \texttt{ } x \texttt{ then } e \texttt{ } \uplus \texttt{ } x \texttt{ else } x$$

$$\textsf{ex}(x,\pi :: \overrightarrow{\pi}) \texttt{ = } \textsf{ex}(\textsf{state-succ } x \texttt{ } \pi, \overrightarrow{\pi})$$
$$\textsf{ex}(x,\texttt{[]}) \texttt{ = } x$$

The result of executing an action (p,e) on a state x depends on whether the preconditions of the action are satisfied by the state or not, which is modelled by the $p \sqsubseteq x$ relation. If the state satisfies the preconditions then the state resulting from the execution is the same as the original state, but amended by the effects of the executed action, otherwise the result of the execution is the same as the original state. The way we model amending the state by an action's effect is by using the finite map union operation $e \uplus x$. We note that this finite map union operation is not commutative, where it gives precedence to the assignments of its first argument. For an action sequence $\overrightarrow{\pi}$, the execution semantics are lifted in a straightforward way. A sanity check of our execution semantics is the following theorem, which states that the result of executing a valid action sequence on a valid state is also a valid state.

$$\vdash \overrightarrow{\pi} \in \delta^* \wedge x \in \mathbb{U}(\delta) \Rightarrow \textsf{ex}(x,\overrightarrow{\pi}) \in \mathbb{U}(\delta)$$

Note that we do not restrict the codomain of states to be *bool*. This is because a lot of the theory we develop applies to factored systems, regardless of the codomain of the state, so for example, a lot of the theory developed applies to hybrid systems.

5.2.1 Topological Properties

In HOL4, we define the diameter in the following way:

HOL4 Definition 3 (Diameter)**.**

$$d(\delta) \texttt{ = } \textsf{max } \{\textsf{min } (\Pi^d \texttt{ } (x,\overrightarrow{\pi},\delta)) \texttt{ | } x \in \mathbb{U}(\delta) \wedge \overrightarrow{\pi} \in \delta^*\}$$

where Π^d *is defined as*

$$\Pi^d \texttt{ } (x,\overrightarrow{\pi},\delta) \texttt{ = } \{|\overrightarrow{\pi'}| \texttt{ | } \textsf{ex}(x,\overrightarrow{\pi'}) \texttt{ = } \textsf{ex}(x,\overrightarrow{\pi}) \wedge \overrightarrow{\pi'} \in \delta^*\}$$

In the above definition, we first define the function Π^d which returns, for a state x, an action sequence $\overrightarrow{\pi}$, and a factored system δ, the set of lengths of all valid action sequences in δ^* that would yield the same execution result as $\overrightarrow{\pi}$, if executed on x. Accordingly, the smallest member of $\Pi^d \texttt{ } (x,\overrightarrow{\pi},\delta)$ would be the length of the shortest action sequence equivalent to $\overrightarrow{\pi}$ in terms of execution on x. The diameter then would the largest such length of the shortest equivalent action sequence for all pairs of states and valid action sequences. In a similar approach the sublist diameter is defined as follows:

HOL4 Definition 4 (Sublist Diameter)**.** *We first define the sublist relation between lists as follows:*

$$[\,] \preceq_{\cdot} l_1 \iff \mathsf{T}$$
$$h::t \preceq_{\cdot} [\,] \iff \mathsf{F}$$
$$x::l_1 \preceq_{\cdot} y::l_2 \iff x = y \wedge l_1 \preceq_{\cdot} l_2 \vee x::l_1 \preceq_{\cdot} l_2$$

Based on that the sublist diameter is defined as:

$$\ell(\delta) = \mathsf{max}\ \{\mathsf{min}\ \Pi^{\preceq_{\cdot}}(x,\vec{\pi}) \mid x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^*\}$$

where $\Pi^{\preceq_{\cdot}}$ *is defined as*

$$\Pi^{\preceq_{\cdot}}(x,\vec{\pi}) = \{|\vec{\pi}'| \mid \mathsf{ex}(x,\vec{\pi}') = \mathsf{ex}(x,\vec{\pi}) \wedge \vec{\pi}' \preceq_{\cdot} \vec{\pi}\}$$

The way we define the sublist diameter resembles the way we defined the diameter. We note however that in $\Pi^{\preceq_{\cdot}}$ we need not add the condition on the equivalent action sequences to be valid action sequences from δ^* since this is implied by the fact that they are also sublists of the given action sequence $\vec{\pi}$.

We define the recurrence diameter and the traversal diameter in a significantly different approach, since they are both best specified in terms of paths of states, and because in both of them we do not need to nest max and min. The recurrence diameter is defined as follows:

HOL4 Definition 5 (recurrence Diameter)**.**

$$rd(\delta) = \mathsf{max}\ \{|p| - 1 \mid \mathsf{valid\text{-}path}\ \delta\ p \wedge \mathsf{ALL\text{-}DISTINCT}\ p\}$$

where

$$\mathsf{valid\text{-}path}\ \delta\ [\,] \iff \mathsf{T}$$
$$\mathsf{valid\text{-}path}\ \delta\ [x] \iff x \in \mathbb{U}(\delta)$$
$$\mathsf{valid\text{-}path}\ \delta\ (x_1::x_2::rest) \iff$$
$$\quad x_1 \in \mathbb{U}(\delta) \wedge (\exists \pi.\ \pi \in \delta \wedge \mathsf{ex}(x_1,[\pi]) = x_2) \wedge$$
$$\quad \mathsf{valid\text{-}path}\ \delta\ (x_2::rest)$$

The predicate valid-path indicates that a certain list of states can be an execution trace of a valid action sequence in the given system, and the predicate ALL-DISTINCT indicates that all members of a certain list are distinct.

Lastly, we define the traversal diameter as follows:

HOL4 Definition 6 (Traversal Diameter)**.**

$$td(\delta) = \mathsf{max}\ \{|\mathsf{ss}\ (x,\vec{\pi})| - 1 \mid x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^*\}$$

where

$$\mathsf{ss}\ (x,\vec{\pi}) = \mathsf{set}\ (\vec{\mathsf{ss}}\ (x,\vec{\pi}))$$

and

$$\vec{\mathsf{ss}}\ (x,\pi::\vec{\pi}) = x::\vec{\mathsf{ss}}\ (\mathsf{state\text{-}succ}\ x\ \pi,\vec{\pi})$$
$$\vec{\mathsf{ss}}\ (x,[\,]) = [x]$$

In the above definition the functions $\vec{\mathsf{ss}}$ and ss return the list of states and the set of states, respectively, traversed by executing an action sequence on a state.

Since we are mainly interested in formally verifying the validity of HYB, we only formalised the different upper-bounding relations between different topological properties, leaving out the exponential separations between them. The formalised bounding relations are expressed in the following theorem.

HOL4 Theorem 1.

$$\vdash \mathsf{FINITE}\ \delta \Rightarrow d(\delta) \leq \ell(\delta) \wedge \ell(\delta) \leq rd(\delta)$$

We note that a sufficient condition for the different topological properties to exist, δ has to be finite and the codomain of the different states has to be *bool*, which is why those conditions exist in the theorem above. This guarantees that the argument sets to the functions max and min are finite. This is not the only approach to guarantee that the topological properties are well defined. However, it suits our needs since it models the factored systems on which we applied our algorithms. As sanity checks to verify that our definitions of the different topological properties have the desired semantics, we prove the following theorems for the sublist diameter and the traversal diameter. The first theorem says that any valid action sequence has a sublist of it that achieves the same execution outcome, and whose length is bounded by the sublist diameter. The second one says that executing any valid action sequence on a valid state traverses no more states than the traversal diameter.

$$\vdash \mathsf{FINITE}\ \delta \wedge x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^* \Rightarrow \exists \vec{\pi}'.\ \mathsf{ex}(x,\vec{\pi}) = \mathsf{ex}(x,\vec{\pi}') \wedge \vec{\pi}' \preceq \cdot\ \vec{\pi} \wedge |\vec{\pi}'| \leq \ell(\delta)$$

$$\vdash \mathsf{FINITE}\ \delta \wedge x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^* \Rightarrow |\mathsf{ss}\ (x,\vec{\pi})| - 1 \leq td(\delta)$$

Also, we prove the following theorems to aid us in deriving the compositional upper bounds of both the sublist diameter and the traversal diameter. Both theorems state sufficient conditions for upper-bounding the two topological properties. For the sublist diameter, if for some constant k, there is a sublist of every valid action sequence that achieves the same execution outcome whose length is bounded by k, then the sublist diameter is bounded by k. For the traversal diameter, if there is some constant k that bounds the number of states traversed by executing every valid action sequence, then the traversal diameter is bounded by one less than k.

$$\vdash \mathsf{FINITE}\ \delta \wedge (\forall \vec{\pi}\ x.\ x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^* \Rightarrow \exists \vec{\pi}'.\ \mathsf{ex}(x,\vec{\pi}) = \mathsf{ex}(x,\vec{\pi}') \wedge \vec{\pi}' \preceq \cdot\ \vec{\pi} \wedge |\vec{\pi}'| \leq k) \Rightarrow \ell(\delta) \leq k$$

$$\vdash \mathsf{FINITE}\ \delta \wedge (\forall x\ \vec{\pi}.\ \mathsf{sat\text{-}pre}\ (x,\vec{\pi}) \wedge x \in \mathbb{U}(\delta) \wedge \vec{\pi} \in \delta^* \Rightarrow |\mathsf{ss}\ (x,\vec{\pi})| \leq k) \Rightarrow td(\delta) \leq k - 1$$

Note that in the theorem for the traversal diameter, there is a requirement on the action sequence to have all of its preconditions satisfied (sat-pre $(x,\vec{\pi})$). This is because in our execution semantics, an action whose preconditions are not satisfied is executed, but with no effect. This means that there can be action sequences with unbounded lengths, that contain actions that do not change the state. This condition is formally defined as follows.

$$\mathsf{sat\text{-}pre}\ (x,(p,e)::\vec{\pi}) \iff p \sqsubseteq x \wedge \mathsf{sat\text{-}pre}\ (\mathsf{state\text{-}succ}\ x\ (p,e),\vec{\pi})$$
$$\mathsf{sat\text{-}pre}\ (x,[\,]) \iff \mathsf{T}$$

5.2.2 Abstraction

An important concept for our work is abstraction of a factored system. The first type of abstraction we consider is projection. To project a state x on a set of variables vs, we use the DRESTRICT function that restricts the domain of a finite map to a set of variables vs, and it is pretty printed as $x\lfloor_{vs}$. For actions, action sequences, and factored systems, we defined projection as follows.

HOL4 Definition 7 (Projection)**.** *For an action*

$$(p,e)\lfloor_{vs} \;=\; (p\lfloor_{vs}, e\lfloor_{vs})$$

For an action sequence

$$((p,e)::\overrightarrow{\pi})\lfloor_{vs} \;=\; \texttt{if}\ \mathcal{D}(e\lfloor_{vs}) \neq \emptyset\ \texttt{then}\ (p,e)\lfloor_{vs}::\overrightarrow{\pi}\lfloor_{vs}\ \texttt{else}\ \overrightarrow{\pi}\lfloor_{vs}$$
$$[]\lfloor_{vs} \;=\; []$$

Letting $f(\!|t|\!)$ denote the image of a function f on some set t, projection for a factored system is defined as

$$\delta\lfloor_{vs} \;=\; (\lambda\pi.\ \pi\lfloor_{vs})(\!|\delta|\!)$$

For a set of states

$$\overrightarrow{x}\lfloor_{vs} \;=\; (\lambda x.\ x\lfloor_{vs})(\!|\overrightarrow{x}|\!)$$

The following theorems show some of the basic properties of the projection operation that we defined.

$$\vdash\ x \in \mathbb{U}(\delta) \Rightarrow x\lfloor_{vs} \in \mathbb{U}(\delta\lfloor_{vs})$$

$$\vdash\ \overrightarrow{\pi} \in \delta^* \Rightarrow \overrightarrow{\pi}\lfloor_{vs} \in \delta\lfloor_{vs}{}^*$$

$$\vdash\ (\mathsf{ex}(x,\overrightarrow{\pi}\lfloor_{vs}))\lfloor_{vs} \;=\; \mathsf{ex}(x\lfloor_{vs}, \overrightarrow{\pi}\lfloor_{vs})$$

$$\vdash\ \textsf{sat-pre}\ (x,\overrightarrow{\pi}) \Rightarrow \mathsf{ex}(x\lfloor_{vs}, \overrightarrow{\pi}\lfloor_{vs}) \;=\; (\mathsf{ex}(x,\overrightarrow{\pi}))\lfloor_{vs}$$

Note that in the theorems above that relate executing a concrete action sequence and its projection, there is the condition sat-pre $(x,\overrightarrow{\pi})$. If this condition does not hold, some actions in the concrete action sequence whose preconditions are not satisfied can have their preconditions satisfied after projection; the projection would then be an incorrect presentation. This could lead to a different execution outcome than the projection of the concrete execution outcome, i.e. $\mathsf{ex}(x\lfloor_{vs}, \overrightarrow{\pi}\lfloor_{vs}) \;=\; (\mathsf{ex}(x,\overrightarrow{\pi}))\lfloor_{vs}$ does not hold unconditionally.

The second type of abstraction that we define is snapshotting.

HOL4 Definition 8 (Snapshotting)**.** *We first define the following relation between states:*

$$\textsf{agree}\ x_1\ x_2 \iff \forall v.\ v \in \mathcal{D}(x_1) \wedge v \in \mathcal{D}(x_2) \Rightarrow x_1\ \text{`}\ v \;=\; x_2\ \text{`}\ v$$

Based on that, a snapshot is defined as

$$\delta\!\upharpoonright_x \;=\; \{(p,e) \mid (p,e) \in \delta \wedge \textsf{agree}\ p\ x \wedge \textsf{agree}\ e\ x\}$$

The relation agree indicates that two states assign all variables in the intersection of their domains to the same values. A snapshot of a factored system δ on a state x is the set of actions from δ whose preconditions are enabled by x and that if executed they do not change the assignments of variables in the domain of x. The following properties of the agreement relation between states and the snapshot abstraction are sanity checks for the validity of our definitions.

$$\vdash\ f_1\ \sqsubseteq\ f_2\ \Rightarrow\ \mathsf{agree}\ f_1\ f_2$$

$$\vdash\ \overrightarrow{\pi}\ \in\ \delta^*\ \wedge\ x'\ \in\ \mathbb{U}(\delta)\ \wedge\ x\ \in\ \mathbb{U}(\delta)\ \wedge\ (\forall p\ e.\ \mathsf{MEM}\ (p,e)\ \overrightarrow{\pi}\ \Rightarrow\ \mathsf{agree}\ e\lfloor_{vs}\ x\lfloor_{vs})\ \wedge\ x'\lfloor_{vs}\ =\ x\lfloor_{vs}\ \Rightarrow\ (\mathsf{ex}(x',\overrightarrow{\pi}))\lfloor_{vs}\ =\ x\lfloor_{vs}$$

The second theorem above shows that if all actions in a sequence agree with a projection of an initial state, then the result of executing that action sequence on that state will have the same assignment to the initially agreed upon variables.

We lastly note that the diameter (resp. sublist diameter, recurrence diameter and traversal diameter) of an abstract system obtained using one of the above abstractions is in general incomparable to that of the concrete system.

5.3 Formalising Compositional Upper-Bounding Algorithms

In this section we review our approach to formally verify the validity of different compositioanl bounding approaches, including those computed by HYB.

5.3.1 Compositional Bounding in the General Case

We now discuss our formalisation of Theorem 3. That theorem states the validity of projection based compositional bounding using the traversal diameter, and it shows that it is always sound regardless of the dependency structure of the system under consideration. To formalise that theorem, we closely follow the pen and paper proof and formalise the propositions on which it depends. Formalising Proposition 1 and Proposition 2 is straightforward, unlike formalising Proposition 3 which warrants more detail. We first define the notion of a state space, which is a set of states that all have the same domain, defined as follows.

HOL4 Definition 9 (State Space)**.**

$$\mathsf{sspc}\ \overrightarrow{x}\ vs\ \Longleftrightarrow\ \forall x.\ x\ \in\ \overrightarrow{x}\ \Rightarrow\ \mathcal{D}(x)\ =\ vs$$

We then define the operation of multiplying two state spaces as follows.

HOL4 Definition 10 (State Space Multiplication)**.**

$$(ss_1\ \times\ ss_2)\ =\ (\lambda\,(x_1,x_2).\ x_1\ \uplus\ x_2)(\!|ss_1\ \times\ ss_2|\!)$$

We then extend that operation to sets of state spaces, following the approach by Nipkow and Paulson (2005). We first define that product as the following inductive relation:

HOL4 Definition 11 (Set of State Spaces Multiplication)**.**

$$\mathsf{PROD}\ \emptyset\ \overrightarrow{x}\ \overrightarrow{x}$$
$$\overrightarrow{x}\ \notin\ SS\ \wedge\ \mathsf{PROD}\ SS\ ss'\ ss''\ \Rightarrow\ \mathsf{PROD}\ (\overrightarrow{x}\ \mathsf{INSERT}\ SS)\ ss'\ (\overrightarrow{x}\ \times\ ss'')$$

Then we prove that this relation is unique, and use that uniqueness to redefine it as the function PRODf. The approach in Nipkow and Paulson (2005) requires the operation to be commutative and associative. The state space product as defined above is associative. However, to have commutativity we assumed that the domains of the different state spaces are disjoint.

$$\vdash\ (ss_1\ \times\ (ss_2\ \times\ ss_3))\ =\ ((ss_1\ \times\ ss_2)\ \times\ ss_3)$$

$$\vdash \mathsf{sspc}\ ss_1\ vs_1 \wedge \mathsf{sspc}\ ss_2\ vs_2 \wedge \mathsf{DISJOINT}\ vs_1\ vs_2 \Rightarrow (ss_1 \times ss_2) = (ss_2 \times ss_1)$$

Since the states traversed by executing any valid action sequence will always be a state space, the following formalised version of Proposition 3 follows.

$$\begin{array}{l}\vdash \mathsf{FINITE}\ VS \wedge (\forall vs'.\ vs' \in VS \Rightarrow \mathsf{DISJOINT}\ vs\ vs') \wedge \\ \quad (\forall vs\ vs'.\ vs \in VS \wedge vs' \in VS \wedge vs \neq vs' \Rightarrow \mathsf{DISJOINT}\ vs\ vs') \wedge \\ \quad \bigcup (vs\ \mathsf{INSERT}\ VS) = \mathcal{D}(\delta) \wedge x \in \mathbb{U}(\delta) \wedge \overrightarrow{\pi} \in \delta^* \wedge \\ \quad \mathsf{sat\text{-}pre}\ (x,\overrightarrow{\pi}) \Rightarrow \\ \quad\quad \mathsf{ss}\ (x,\overrightarrow{\pi}) \subseteq \mathsf{PRODf}\ (\lambda vs.\ (\mathsf{ss}\ (x,\overrightarrow{\pi}))\lfloor_{vs})(\langle\!|VS|\!\rangle)\ (\mathsf{ss}\ (x,\overrightarrow{\pi}))\lfloor_{vs}\end{array}$$

Lastly, the formalised version of Theorem 3 is as follows.

HOL4 Theorem 2.

$$\begin{array}{l}\vdash \mathsf{FINITE}\ VS \wedge \mathsf{FINITE}\ \delta \wedge vs \notin VS \wedge \\ \quad (\forall vs'.\ vs' \in VS \Rightarrow \mathsf{DISJOINT}\ vs\ vs') \wedge \\ \quad (\forall vs\ vs'.\ vs \in VS \wedge vs' \in VS \wedge vs \neq vs' \Rightarrow \mathsf{DISJOINT}\ vs\ vs') \wedge \\ \quad \bigcup (vs\ \mathsf{INSERT}\ VS) = \mathcal{D}(\delta) \wedge vs \notin VS \Rightarrow \\ \quad\quad td(\delta) \leq \Pi\ (\lambda vs.\ td(\delta\lfloor_{vs}) + 1)\ (vs\ \mathsf{INSERT}\ VS) - 1\end{array}$$

5.3.2 Compositional Bounds in the Presence of Acyclicity

We now review our formalisation of acyclicity, Lemma 3, and Theorem 14. In HOL we model a DAG with the predicate top-sorted-abs that means that l is a list of vertices of a digraph topologically sorted w.r.t. to the binary relation R, which is taken to be the edge relation in the digraph. This predicate is defined as follows in HOL:

HOL4 Definition 12 (DAG)**.**

$$\begin{array}{l}\mathsf{top\text{-}sorted\text{-}abs}\ R\ (u_1 :: A) \iff \\ \quad \mathsf{EVERY}\ (\lambda u_2.\ \neg R\ u_2\ u_1)\ A \wedge \mathsf{top\text{-}sorted\text{-}abs}\ R\ A \\ \mathsf{top\text{-}sorted\text{-}abs}\ R\ [\,] \iff \mathsf{T}\end{array}$$

In the definition above, EVERY is a high-order predicate that, given a predicate and a list, returns true if every member of the list satisfies the given predicate. The following basic properties hold for DAGs.

$$\vdash \mathsf{top\text{-}sorted\text{-}abs}\ R\ (u_1 :: A) \wedge u_2 \in \mathsf{set}\ A \Rightarrow \neg R\ u_2\ u_1$$

$$\vdash \mathsf{top\text{-}sorted\text{-}abs}\ R\ (u :: A) \Rightarrow \mathsf{top\text{-}sorted\text{-}abs}\ R\ A$$

Recall that one way to interpret both bounding functions N and S is as functions that compute some form of weightiest paths in an acyclic graph, where every path is given a weight that depends on the weights of the vertices and the number of edges that it traverses. However, both functions operate on a DAG induced by different relations (dependency for N versus successor state for S), weigh vertices with different functions (projection's sublist diameter for N versus snapshot's sublist diameter for S), and combine those weights in different ways (multiply them and add for for N versus add and take the maximum for S). To be able to factor what is common between the two bounding functions and yet accommodate the difference between them, we defined the following function.

HOL4 Definition 13 (Weightiest Longest Path)**.**

$$\begin{array}{l}\mathsf{wp}\ R\ w\ g\ f\ u_1\ (u_2 :: A)\ = \\ \quad \texttt{if}\ R\ u_1\ u_2\ \texttt{then}\ g\ (f\ (w\ u_1)\ (\mathsf{wp}\ R\ w\ g\ f\ u_2\ A))\ (\mathsf{wp}\ R\ w\ g\ f\ u_1\ A) \\ \quad \texttt{else}\ \mathsf{wp}\ R\ w\ g\ f\ u_1\ A \\ \mathsf{wp}\ R\ w\ g\ f\ u_1\ []\ =\ w\ u_1\end{array}$$

This function takes as an argument the relation that induces the DAG R, the weighing function w, and the two functions that combine the weights: g which combines weights of different paths; and f which combines the weights of vertices on one path, as well as the vertex of interest v and the DAG l. In order for wp to capture desired properties about S and N, we define the following properties on the weight combination function.

$$\mathsf{geq\text{-}arg}\ f \iff \forall x\ y.\ x \leq f\ x\ y \wedge y \leq f\ x\ y$$

$$\mathsf{increasing}\ f \iff \forall e\ b\ c\ d.\ e \leq c \wedge b \leq d \Rightarrow f\ e\ b \leq f\ c\ d$$

The previous definitions describe different notions of functions whose value is at least as big as their arguments and functions that are non-decreasing. If the weight combination functions f and g have those properties, the following basic properties hold for the abstract weighted longest path function, and accordingly for $\mathsf{N}_{\mathsf{sum}}$ and $\mathsf{S}_{\mathsf{max}}$.

$$\vdash \mathsf{geq\text{-}arg}\ g \Rightarrow w\ u \leq \mathsf{wp}\ R\ w\ g\ f\ u\ A$$

$$\begin{array}{l}\vdash \mathsf{geq\text{-}arg}\ f \wedge \mathsf{geq\text{-}arg}\ g \wedge (\forall u.\ u \in \mathsf{set}\ A \Rightarrow \neg R\ u\ u) \wedge R\ u_2\ u_1 \wedge \\ \quad u_1 \in \mathsf{set}\ A \wedge \mathsf{top\text{-}sorted\text{-}abs}\ R\ A \Rightarrow \\ \quad\quad f\ (w\ u_2)\ (\mathsf{wp}\ R\ w\ g\ f\ u_1\ A) \leq \mathsf{wp}\ R\ w\ g\ f\ u_2\ A\end{array}$$

$$\begin{array}{l}\vdash \mathsf{increasing}\ f \wedge \mathsf{increasing}\ g \wedge \\ \quad (\forall u.\ u \in \mathsf{set}\ A \Rightarrow w_1\ u \leq w_2\ u) \wedge w_1\ u \leq w_2\ u \Rightarrow \\ \quad\quad \mathsf{wp}\ R\ w_1\ g\ f\ u\ A \leq \mathsf{wp}\ R\ w_2\ g\ f\ u\ A\end{array}$$

Formalising the Validity of $\mathsf{N}_{\mathsf{sum}}$

We now review our formalisation of the compositional bounding of the sublist diameter, for the case of projections induced by an acyclic dependency graph. We defined dependency between variables and dependency between sets of variables as follows.

HOL4 Definition 14 (Dependency)**.** *For two variables v_1 and v_2, we define dependency as*[2]

$$\begin{array}{l}v_1 \rightarrow v_2 \iff \\ \quad (\exists p\ e. \\ \quad\quad (p,e) \in \delta \wedge \\ \quad\quad (v_1 \in \mathcal{D}(p) \wedge v_2 \in \mathcal{D}(e) \vee v_1 \in \mathcal{D}(e) \wedge v_2 \in \mathcal{D}(e))) \vee \\ \quad v_1 = v_2\end{array}$$

For sets of variables vs_1 and vs_1, we define

$$\begin{array}{l}vs_1 \rightarrow vs_2 \iff \\ \quad \exists v_1\ v_2.\ v_1 \in vs_1 \wedge v_2 \in vs_2 \wedge \mathsf{DISJOINT}\ vs_1\ vs_2 \wedge v_1 \rightarrow v_2\end{array}$$

[2]The definition of $\rightarrow$ has an implicit δ parameter; however, we hide it using the *ad hoc* overloading ability in HOL.

In the above defintion, DISJOINT is a predicate that, given two sets, returns true if they share no elements.

Our proof of Theorem 8 depends on Lemma 3 whose proof depends on the following fundamental argument. Consider a set of variables vs with only incoming dependencies from all the state variables in a system δ (a "parent-child" dependency structure). If for some $\overrightarrow{\pi} \in \delta^*$ one is to remove actions from the projection $\overrightarrow{\pi}\downarrow_{vs}$, while preserving the projection's execution outcome, the restitching function can be used to "stitch" back the shortened projected action sequence into $\overrightarrow{\pi}$, producing an action sequence with the same execution outcome as $\overrightarrow{\pi}$, but with the redundant actions removed. To formalise that, we formalise the parent-child dependency configuration and the restitching function as follows:

HOL4 Definition 15 (Parent-Child Structure)**.**

$$\textsf{child-parent-rel}\ (\delta, vs) \iff vs \not\rightarrow \overline{vs}$$

HOL4 Definition 16 (Restitching Function)**.**

$$\begin{array}{l}
(\pi' :: \overrightarrow{\pi'}) \underset{vs}{\wr\!\wr} (\pi :: \overrightarrow{\pi}) = \\
\quad \texttt{if}\ \textsf{varset-action}\ (\pi, vs)\ \texttt{then} \\
\quad\quad \texttt{if}\ \pi' = \pi\downarrow_{vs}\ \texttt{then}\ \pi :: \overrightarrow{\pi'} \underset{vs}{\wr\!\wr} \overrightarrow{\pi}\ \texttt{else}\ (\pi' :: \overrightarrow{\pi'}) \underset{vs}{\wr\!\wr} \overrightarrow{\pi} \\
\quad \texttt{else}\ \pi :: (\pi' :: \overrightarrow{\pi'}) \underset{vs}{\wr\!\wr} \overrightarrow{\pi} \\
[] \underset{vs}{\wr\!\wr} \overrightarrow{\pi} = \textsf{FILTER}\ (\lambda \pi .\ \neg \textsf{varset-action}\ (\pi, vs))\ \overrightarrow{\pi} \\
(\pi' :: \overrightarrow{\pi'}) \underset{vs}{\wr\!\wr} [] = []
\end{array}$$

where varset-action *is defined as follows*

$$\textsf{varset-action}\ ((p, e), vs) \iff \mathcal{D}(e) \subseteq vs$$

In words, the stitching function uses the first list as a guide: the second list of actions is filtered, with each action in the first list meant to have a corresponding action in the second list. The formalised theorem stating the aforementioned functionality of the restitching function is as follows. Its proof script is 1300 lines long, with comments, and it is our main tool to formally prove the validity of $\mathsf{N_{sum}}$ as a bound.

HOL4 Lemma 1.

$$\begin{array}{l}
\vdash \textsf{child-parent-rel}\ (\delta, vs) \wedge x \in \mathbb{U}(\delta) \wedge \textsf{no-effectless-act}\ \overrightarrow{\pi} \wedge \\
\quad \overrightarrow{\pi'} \preceq \cdot\ \overrightarrow{\pi}\downarrow_{vs} \wedge \overrightarrow{\pi} \in \delta^* \wedge (\textsf{ex}(x, \overrightarrow{\pi}))\downarrow_{vs} = \textsf{ex}(x\downarrow_{vs}, \overrightarrow{\pi'}) \wedge \\
\quad \textsf{sat-pre}\ (x, \overrightarrow{\pi}) \wedge \textsf{sat-pre}\ (x, \overrightarrow{\pi'}) \Rightarrow \\
\quad\quad \textsf{ex}(x, \overrightarrow{\pi'} \underset{vs}{\wr\!\wr} \overrightarrow{\pi}) = \textsf{ex}(x, \overrightarrow{\pi})
\end{array}$$

In the previous statement, the predicate no-effectless-act $\overrightarrow{\pi}$ asserts that the action sequence has no actions with empty effects. As we sketched earlier in the proof summary of Lemma 3 in Section 3.6, for a lifted dependency DAG A_{VS}, we take every set of variables $p \in A_{\mathrm{VS}}$, and remove all redundant actions whose effects are confined to p. As we describe in detail below, we use the stitching function as our main proof tool to perform the action removal. To formalise this, we define the following relation, which generalises the parent-child structure.

HOL4 Definition 17 (Generalised Parent-Child Structure)**.** *For a factored transition system δ and two sets of variables vs_1 and vs_2, the generalised parent-child relation holds between vs_1 and vs_2 iff (i) $vs_2 \not\rightarrow vs_1$, (ii) $vs_1 \not\rightarrow \overline{(vs_1 \cup vs_2)}$, and (iii) no bidirectional dependencies exist between any variable in vs_2 and $\overline{(vs_1 \cup vs_2)}$. Formally:*

$$\begin{array}{l}\textsf{gen-parent-child}\ (\delta,p,c) \iff \\ \quad \textsf{DISJOINT}\ p\ c \wedge c \not\rightarrow p \wedge p \not\rightarrow \overline{p \cup c} \wedge \\ \quad \forall v_1\ v_2 .\ v_1 \in c \wedge v_2 \in \overline{p \cup c} \Rightarrow v_1 \not\rightarrow v_2 \vee v_2 \not\rightarrow v_1\end{array}$$

Given the previous definition, the following lemma formally describes the process of removing redundant actions affecting a set of variables $p \in A_{\text{VS}}$.

Lemma 9. *Let $\mathsf{n}(vs,\overrightarrow{\pi})$ be the number of vs-actions contained within $\overrightarrow{\pi}$. Consider δ, in which the generalised parent-child relation holds between sets of variables p and c. Then, any action sequence $\overrightarrow{\pi}$ has a sublist $\overrightarrow{\pi}'$ that reaches the same state as $\overrightarrow{\pi}$ starting from any state such that: $\mathsf{n}(p,\overrightarrow{\pi}') \leq \ell(\delta\lfloor_p)(\mathsf{n}(c,\overrightarrow{\pi}')+1)$ and $\mathsf{n}(\overline{p},\overrightarrow{\pi}') \leq \mathsf{n}(\overline{p},\overrightarrow{\pi})$.*

A formal statement of that lemma follows:

$$\begin{array}{l}\vdash \textsf{FINITE}\ \delta \wedge x \in \mathbb{U}(\delta) \wedge \overrightarrow{\pi} \in \delta^* \wedge \textsf{gen-parent-child}\ (\delta,p,c) \Rightarrow \\ \quad \exists \overrightarrow{\pi}' . \\ \quad\quad \mathsf{n}(p,\overrightarrow{\pi}') \leq \ell(\delta\lfloor_p) \times (\mathsf{n}(c,\overrightarrow{\pi}') + 1) \wedge \overrightarrow{\pi}' \preceq \cdot\ \overrightarrow{\pi} \wedge \\ \quad\quad \mathsf{n}(\mathcal{D}(\delta) \setminus p,\overrightarrow{\pi}') \leq \mathsf{n}(\mathcal{D}(\delta) \setminus p,\overrightarrow{\pi}) \wedge \mathsf{ex}(x,\overrightarrow{\pi}') = \mathsf{ex}(x,\overrightarrow{\pi})\end{array}$$

where

$$\mathsf{n}(p,\overrightarrow{\pi}) = |\textsf{FILTER}\ (\lambda\pi .\ \textsf{varset-action}\ (\pi,p))\ \overrightarrow{\pi}|$$

Proof. The proof of Lemma 9 is constructive. Let $\overrightarrow{\pi}_{\overline{c}}$ be a contiguous fragment of $\overrightarrow{\pi}$ that has no c-actions in it. Then perform the following steps:

- By the definition of ℓ, there must be an action sequence $\overrightarrow{\pi}_p$ such that $\mathsf{ex}(s,\overrightarrow{\pi}_p) = \mathsf{ex}(s,\overrightarrow{\pi}_{\overline{c}}\lfloor_p)$, and satisfies $|\overrightarrow{\pi}_p| \leq \ell(\delta\lfloor_p)$ and $\overrightarrow{\pi}_p \preceq \overrightarrow{\pi}_{\overline{c}}\lfloor_p$.
- Letting $D \equiv \mathcal{D}(\delta)$, $p \not\rightarrow D \setminus p \setminus c$ holds. Accordingly, from HOL4 Lemma 1, $\overrightarrow{\pi}'_{\overline{c}} (= \overrightarrow{\pi}_p \underset{p}{\Big\}} \overrightarrow{\pi}_{\overline{c}}\lfloor_{D\setminus c})$ achieves the same $D \setminus c$ assignment as $\overrightarrow{\pi}_{\overline{c}}$ (i.e., $\mathsf{ex}(s,\overrightarrow{\pi}'_{\overline{c}})\lfloor_{D\setminus c} = \mathsf{ex}(s,\overrightarrow{\pi}_{\overline{c}})\lfloor_{D\setminus c}$), and it is a sublist of $\overrightarrow{\pi}_{\overline{c}}$. Also, $\mathsf{n}(p,\overrightarrow{\pi}'_{\overline{c}}) \leq \ell(\delta\lfloor_p)$ holds.
- Finally, because $\overrightarrow{\pi}_{\overline{c}}$ has no c-actions, no c variables change along the execution of $\overrightarrow{\pi}_{\overline{c}}$ and accordingly any c variables in preconditions of actions in $\overrightarrow{\pi}_{\overline{c}}$ always have the same assignment. This means that $\overrightarrow{\pi}'_{\overline{c}} \underset{D\setminus c}{\Big\}} \overrightarrow{\pi}_{\overline{c}}$ will achieve the same result as $\overrightarrow{\pi}_{\overline{c}}$, but with at most $\ell(\delta\lfloor_p)$ p-actions.

Repeating the previous steps for each $\overrightarrow{\pi}_{\overline{c}}$ fragment in $\overrightarrow{\pi}$ yields an action sequence $\overrightarrow{\pi}'$ that has at most $\ell(\delta\lfloor_p)(\mathsf{n}(c,\overrightarrow{\pi})+1)$ p-actions. This is justified by the following lemma, where in this lemma $\textsf{list-frag}\ (l_1,l_2)$ means that list l_2 is a contiguous sublist of l_1.

$$\begin{array}{l}\vdash |\textsf{FILTER}\ P_1\ l| \leq k_1 \wedge (\forall x .\ x \in \textsf{set}\ l \Rightarrow P_1\ x \Rightarrow \neg P_2\ x) \wedge \\ \quad (\forall l' . \\ \quad\quad \textsf{list-frag}\ (l,l') \wedge \textsf{EVERY}\ (\lambda x .\ \neg P_1\ x)\ l' \Rightarrow \\ \quad\quad\quad |\textsf{FILTER}\ P_2\ l'| \leq k_2) \Rightarrow \\ \quad |\textsf{FILTER}\ P_2\ l| \leq (k_1 + 1) \times k_2\end{array}$$

Because $\overrightarrow{\pi}'$ is the result of consecutive applications of the stitching function, it is a sublist of $\overrightarrow{\pi}$. Lastly, because during the previous steps, only p-actions were removed as necessary, the number of c-actions in $\overrightarrow{\pi}'$ is the same as their number in $\overrightarrow{\pi}$. □

We can now describe how we use that result to prove Lemma 3, the main lemma stating the validity of using $\mathsf{N}_{\mathsf{sum}}$ to compositionally upper-bound the sublist diameter. The main idea is to preform an induction on the acyclic lifted dependency graph, where for every node $p \in A_{\mathrm{VS}}$, the redundant actions of p are removed using Lemma 9 after removing the redundant actions of its children c. Then we use the stitching function to reconcile both shortened action sequences. We now describe our formalisation of that proof. Below are the formal definitions of a lifted dependency DAG, and the function N. They are instantiations of top-sorted-abs and wp, respectively, where the relation that induces the digraph is the variable dependency relation. Also for N, the functions for combining node weights g and f, are instantiated with addition and multiplication, respectively[3].

HOL4 Definition 9 (Lifted Dependency DAG)**.**

$$\begin{aligned}&\mathsf{dep\text{-}DAG}\ \delta\ A_{\mathrm{VS}} \iff \\ &\quad \mathcal{D}(\delta)\ =\ \bigcup\ (\mathsf{set}\ A_{\mathrm{VS}})\ \wedge\ \mathsf{ALL\text{-}DISTINCT}\ A_{\mathrm{VS}}\ \wedge \\ &\quad \mathsf{ALL\text{-}DISJOINT}\ A_{\mathrm{VS}}\ \wedge \\ &\quad \mathsf{top\text{-}sorted\text{-}abs}\ (\lambda\, vs_1\ \ vs_2\,.\ \ vs_1\ \rightarrow\ vs_2)\ A_{\mathrm{VS}}\end{aligned}$$

In the above definition the predicates ALL-DISTINCT and ALL-DISJOINT return true if all members of a given list are pairwise distinct and pairwise disjoint, respectively.

HOL4 Definition 10 (Acyclic Dependency Compositional Bound)**.**

$$\mathsf{N}\langle b\rangle(vs)\ =\ \mathsf{wp}\ (\lambda\, vs_1\ \ vs_2\,.\ \ vs_1\ \rightarrow\ vs_2)\ (\lambda\, vs\,.\ \ b(\delta\lfloor_{vs}))\ \ (+)\ \ (\times)\ \ vs\ \ A_{\mathrm{VS}}$$

This is the formal statement of Lemma 3.

HOL4 Theorem 3.

$$\vdash\ \mathsf{FINITE}\ \delta\ \wedge\ \mathsf{dep\text{-}DAG}\ \delta\ A_{\mathrm{VS}}\ \Rightarrow\ \ell(\delta)\ <\ \mathsf{SUM}\ (\mathsf{MAP}\ \mathsf{N}\langle\ell\rangle\ A_{\mathrm{VS}})\ +\ 1$$

Before we discuss the formal proof, we first introduce the following notation. Let $F(p, c, \overrightarrow{\pi})$ be the witness action sequence of Lemma 9. We know that:

- $\mathsf{ex}(s, F(p, c, \overrightarrow{\pi})) = \mathsf{ex}(s, \overrightarrow{\pi})$,
- $\mathsf{n}(p, F(p, c, \overrightarrow{\pi})) \leq \ell(\delta\lfloor_p)(\mathsf{n}(c, \overrightarrow{\pi}) + 1)$.
- $F(p, c, \overrightarrow{\pi}) \preceq \overrightarrow{\pi}$, and
- $\mathsf{n}(\overline{p}, F(p, c, \overrightarrow{\pi})) \leq \mathsf{n}(\overline{p}, \overrightarrow{\pi})$.

Proof of Lemma 3. Firstly, for brevity, for a set of variables vs, in the rest of this proof we write $\mathsf{N}\langle\ell\rangle(vs)$ as a short-hand for $\mathsf{N}\langle\ell\rangle(vs, \delta, A_{\mathrm{VS}})$.

Our proof of this lemma follows a constructive approach where we assume we have an action sequence $\overrightarrow{\pi} \in \delta^*$ and a state $s \in \mathbb{U}(\delta)$. The goal of the proof is to find a witness sublist, $\overrightarrow{\pi}'$, of $\overrightarrow{\pi}$ such that $\forall\ vs \in A_{\mathrm{VS}}.\ \mathsf{n}(vs, \overrightarrow{\pi}') \leq \mathsf{N}\langle\ell\rangle(vs)$ and $\mathsf{ex}(s, \overrightarrow{\pi}) = \mathsf{ex}(s, \overrightarrow{\pi}')$. We proceed by induction

[3] N has δ and A_{VS} parameters, but we hide them with HOL's *ad hoc* overloading ability.

on $V(A_{\mathrm{VS}})$ assuming it is topologically sorted in a list l_{VS} (without loss of generality since A_{VS} is a DAG). The base case is the empty list [], in which case $\mathcal{D}(\delta) = \emptyset$ and accordingly $\ell(\delta) = 0$.

In the step case, we assume the result holds for any factored system for which l'_{VS} is a topologically sorted vertices list of one of its lifted dependency graphs. We then show that it also holds for δ, a factored system whose dependency graph's vertices are topologically sorted into $vs :: l_{\mathrm{VS}}$. Let $\overline{vs} \equiv \bigcup l_{\mathrm{VS}}$. Since l_{VS} is a topologically sorted vertices list of a lifted dependency graph of $\delta\downharpoonright_{\overline{vs}}$, the induction hypothesis applies. Accordingly, there is $\vec{\pi}_{\overline{vs}} \in \delta\downharpoonright_{\overline{vs}}{}^*$ such that $\mathsf{ex}(s, \vec{\pi}_{\overline{vs}}) = \mathsf{ex}(s, \vec{\pi}\downharpoonright_{\overline{vs}})$, $\vec{\pi}_{\overline{vs}} \preceq \vec{\pi}\downharpoonright_{\overline{vs}}$, and $\forall\ vs' \in l_{\mathrm{VS}}.\ \mathsf{n}(vs', \vec{\pi}') \leq \mathsf{N}\langle\ell\rangle(vs', \delta\downharpoonright_{\overline{vs}}, A_{\mathrm{VS}}) \leq \mathsf{N}\langle\ell\rangle(vs')$. Since $vs :: l_{\mathrm{VS}}$ is topologically sorted, $\overline{vs} \not\rightarrow vs$ holds. Letting $\vec{\pi}'_{\overline{vs}} = \vec{\pi}_{\overline{vs}} \underset{vs}{\wr} \vec{\pi}$, from HOL4 Lemma 1 we have $\mathsf{ex}(s, \vec{\pi}'_{\overline{vs}}) = \mathsf{ex}(s, \vec{\pi})$. Furthermore, $\forall vs' \in l_{\mathrm{VS}}.\ \mathsf{n}(vs', \vec{\pi}'_{\overline{vs}}) \leq \mathsf{N}\langle\ell\rangle(vs')$ and $\vec{\pi}'_{\overline{vs}} \preceq \vec{\pi}$. Let $C \equiv \bigcup \mathsf{children}_{A_{\mathrm{VS}}}(vs)$. The last step in this proof is to show that $F(vs, C, \vec{\pi}'_{\overline{vs}})$ is the required witness, which is justified because the generalised parent-child relation holds for δ, vs and C. Since the relations $=$, $\leq$ and $\preceq$ are transitive, we have

- $\mathsf{ex}(s, \vec{\pi}) = \mathsf{ex}(s, F(vs, C, \vec{\pi}'_{\overline{vs}}))$,
- $\mathsf{n}(vs, F(vs, C, \vec{\pi}'_{\overline{vs}})) \leq \ell(\delta\downharpoonright_{vs})(\mathsf{n}(C, \vec{\pi}'_{\overline{vs}}) + 1) = \ell(\delta\downharpoonright_{vs})(\sum_{c \in C} \mathsf{n}(c, \vec{\pi}'_{\overline{vs}}) + 1)$,
- $F(vs, C, \vec{\pi}'_{\overline{vs}}) \preceq \vec{\pi}$, and
- $\mathsf{n}(\overline{vs}, F(vs, C, \vec{\pi}'_{\overline{vs}})) \leq \mathsf{n}(\overline{vs}, \vec{\pi}'_{\overline{vs}})$.

Since $\sum_{vs' \in l_{\mathrm{VS}}} \mathsf{n}(vs', \vec{\pi}'_{\overline{vs}}) = \mathsf{n}(\overline{vs}, \vec{\pi}'_{\overline{vs}})$, then $\forall\ vs' \in l_{\mathrm{VS}}.\ \mathsf{n}(vs', \vec{\pi}'_{\overline{vs}}) \leq \mathsf{N}\langle\ell\rangle(vs')$ and $\mathsf{n}(vs, F(vs, C, \vec{\pi}'_{\overline{vs}})) \leq \ell(\delta\downharpoonright_{vs})(\sum_{c \in C} \mathsf{N}\langle\ell\rangle(c))$ hold. Therefore $F(vs, C, \vec{\pi}'_{\overline{vs}})$ is an action sequence demonstrating the needed bound. □

Formalising the Validity of $\mathsf{S}_{\mathsf{max}}$

To formalise compositional upper-bounding under state space acyclicity, we again use top-sorted-abs and wp to formalise the digraph modelling the state space and the function S, respectively. In this case, the relation that induces the digraph is the successor relation on states. We note that in order for us to use the second property of wp, the relation the induces the digraph needs to be irreflexive, which is why the set of successors of a state is defined to not include that state. For S, the vertex weight combination functions g and f are instantiated by a function that chooses the maximum of two arguments and the addition function, respectively. Formally this is defined as follows[4]

HOL4 Definition 11 (Acyclic System State Space)**.**

$$\mathsf{sspace\text{-}DAG}\ \delta\ \vec{x} \iff \mathsf{set}\ \vec{x} = \mathbb{U}(\delta) \land \mathsf{top\text{-}sorted\text{-}abs}\ (\mathsf{succ}\ \delta)\ \vec{x}$$

where

$$\mathsf{succ}\ \delta\ x = \mathsf{state\text{-}succ}\ x(\!|\delta|\!) \setminus \{x\}$$

HOL4 Definition 12 (Acyclic System Compositional Bound)**.**

$$\mathsf{S}\langle b\rangle(x) = \mathsf{wp}\ (\mathsf{succ}\ \delta\downharpoonright_{vs})\ (\lambda x.\ b(\delta\!\restriction_x))\ \mathsf{MAX}\ (\lambda x\ y.\ x + y + 1)\ x\ \vec{x}$$

[4]S has vs, δ and $\vec{x}$ parameters, but we hide them with HOL's *ad hoc* overloading ability.

Formalising Theorem 14 is a straightforward implementation of the proof discussed in Section 3.6.2. Proposition 5 follows from the properties of wp shown earlier as well as HOL4 Definition 12. To formalise the other propositions we define the function ∂, that gives the number of changes in the assignments of a set of variables vs if an action sequence $\overrightarrow{\pi}$ is executed on a state x, as follows

HOL4 Definition 13 (Subsystem Trace)**.**

$$\begin{array}{l}\partial\ (\pi::\overrightarrow{\pi})\ vs\ x\ =\\ \quad \texttt{if}\ (\mathsf{state\text{-}succ}\ x\ \pi)\lfloor_{vs}\ \neq\ x\lfloor_{vs}\ \texttt{then}\\ \quad\quad \mathsf{state\text{-}succ}\ x\ \pi::\partial\ \overrightarrow{\pi}\ vs\ (\mathsf{state\text{-}succ}\ x\ \pi)\\ \quad \texttt{else}\ \partial\ \overrightarrow{\pi}\ vs\ (\mathsf{state\text{-}succ}\ x\ \pi)\\ \partial\ \texttt{[]}\ vs\ x\ =\ \texttt{[]}\end{array}$$

Proposition 6, Proposition 7 are formalised as follows.

HOL4 Proposition 1.

$$\begin{array}{l}\vdash\ \mathsf{FINITE}\ \delta\ \wedge\ \partial\ \overrightarrow{\pi}\ vs\ x\ =\ \texttt{[]}\ \wedge\ \mathsf{sat\text{-}pre}\ (x,\overrightarrow{\pi})\ \wedge\ x\ \in\ \mathbb{U}(\delta)\ \wedge\\ \quad \overrightarrow{\pi}\ \in\ \delta^*\ \Rightarrow\\ \quad\quad \exists\overrightarrow{\pi'}.\\ \quad\quad\quad \mathsf{ex}(x\lfloor_{\mathcal{D}(\delta\downarrow_{x\lfloor vs})},\overrightarrow{\pi})\ =\ \mathsf{ex}(x\lfloor_{\mathcal{D}(\delta\downarrow_{x\lfloor vs})},\overrightarrow{\pi'})\ \wedge\ \overrightarrow{\pi'}\ \preceq\cdot\ \overrightarrow{\pi}\ \wedge\\ \quad\quad\quad |\overrightarrow{\pi'}|\ \leq\ \ell(\delta\downarrow_{x\lfloor vs})\end{array}$$

HOL4 Proposition 2.

$$\begin{array}{l}\vdash\ \partial\ \overrightarrow{\pi}\ vs\ x\ =\ x'::\overrightarrow{x}\ \Rightarrow\\ \quad\quad \exists\overrightarrow{\pi}_1\ \pi\ \overrightarrow{\pi}_2.\\ \quad\quad\quad \overrightarrow{\pi}\ =\ \overrightarrow{\pi}_1\ +\!\!+\ \pi::\overrightarrow{\pi}_2\ \wedge\ \partial\ \overrightarrow{\pi}_1\ vs\ x\ =\ \texttt{[]}\ \wedge\\ \quad\quad\quad \mathsf{state\text{-}succ}\ (\mathsf{ex}(x,\overrightarrow{\pi}_1))\ \pi\ =\ x'\ \wedge\\ \quad\quad\quad \partial\ \overrightarrow{\pi}_2\ vs\ (\mathsf{state\text{-}succ}\ (\mathsf{ex}(x,\overrightarrow{\pi}_1))\ \pi)\ =\ \overrightarrow{x}\end{array}$$

Note that in the conclusions of the previous formalised statements there is an extra requirement on the witness action sequence: that it is a sublist of the original sequence. This is one difference from the proof discussed of Theorem 13 in Section 3.6.2 and it is because in the earlier proof the aim is to prove the validity of $\mathsf{S}_{\mathsf{max}}$ as a compositional bound on the diameter, while here we prove its validity as an upper bound on the sublist diameter. Other than this extra conclusion, the proofs of Theorem 13 and Theorem 14 are the same. Lemma 6 is then formalised as follows:

HOL4 Lemma 2.

$$\begin{array}{l}\vdash\ \mathsf{FINITE}\ \delta\ \wedge\ \mathsf{sspace\text{-}DAG}\ \delta\lfloor_{vs}\ \overrightarrow{x}\ \wedge\ x\ \in\ \mathbb{U}(\delta)\ \wedge\ \overrightarrow{\pi}\ \in\ \delta^*\ \Rightarrow\\ \quad\quad \exists\overrightarrow{\pi'}.\ \mathsf{ex}(x,\overrightarrow{\pi'})\ =\ \mathsf{ex}(x,\overrightarrow{\pi})\ \wedge\ \overrightarrow{\pi'}\ \preceq\cdot\ \overrightarrow{\pi}\ \wedge\ |\overrightarrow{\pi'}|\ \leq\ \mathsf{S}\langle\ell\rangle(x\lfloor_{vs})\end{array}$$

Finally, based on HOL4 Lemma 2 the formalised statement of Theorem 14 is as follows:

HOL4 Theorem 4.

$$\begin{array}{l}\vdash\ \mathsf{FINITE}\ \delta\ \wedge\ \mathsf{sspace\text{-}DAG}\ \delta\lfloor_{vs}\ \overrightarrow{x}\ \Rightarrow\\ \quad\quad \ell(\delta)\ \leq\ \mathsf{max}\ \{\mathsf{S}\langle\ell\rangle(x')\ \mid\ x'\ \in\ \mathbb{U}(\delta\lfloor_{vs})\}\end{array}$$

Formalising the Validity of HYB

We formalise HYB as follows:

HOL4 Definition 14 (The Hybrid Algorithm)**.**

```
HYB f1 f2 δ =
  if FINITE δ then
    if ∀vs. vs ∈ set (f1 δ) ⇒ vs ⊂ 𝒟(δ) then
      (let
         A_VS = f1 δ
       in
         SUM
           (MAP
              (N
                 (λδ'.
                    if ∃vs. vs ∈ set A_VS ∧ δ' = δ⌊vs then
                      HYB f1 f2 δ'
                    else 0) δ A_VS) A_VS))
    else if
      f2 δ ≠ ∅ ∧ ∀vs x⃗ x. (vs,x⃗) ∈ f2 δ ∧ x ∈ set x⃗ ⇒ δ↾x ⊂ δ
    then
      (let
         (vs,x⃗) = CHOICE (f2 δ)
       in
         max
           {S-gen
              (λδ'.
                 if ∃x. x ∈ set x⃗ ∧ δ' = δ↾x then HYB f1 f2 δ'
                 else 0) vs x⃗ δ x' |
            x' ∈ 𝕌(δ⌊vs)})
    else ℓ(δ)
  else 0
```

It takes two functions f_1 and f_2 as arguments. The first function is an oracle that given a transition system, returns a lifted dependency DAG of that system. The second function is an oracle that returns a set of pairs, each of which has a strict subset of the state variables of the given system, and the state space of the projection of the system on that subset of the state variables. We note that proving the termination of HYB is not trivial. In order to guarantee termination, we add the conditions that δ is finite as well as that all the lifted dependency DAGs computed by f_1 and the subsets of the domain of δ computed by f_2 provide non trivial decompositions of δ, i.e. they are strict subsets. Also, in order to guide HOL4 to extract the right termination conditions, we add the redundant if-then-else statements in the functional parameters passed to N and S. After proving termination, we are able to obtain the following characterisation that resembles Algorithm 5.

HOL4 Lemma 3 (The Hybrid Algorithm)**.**

```
FINITE δ ∧ (∀vs x⃗. (vs,x⃗) ∈ f2 δ ⇒ set x⃗ = 𝕌(δ⌊vs)) ⇒
  HYB f1 f2 δ =
    if ∀vs. vs ∈ set (f1 δ) ⇒ vs ⊂ 𝒟(δ) then
      (let A_VS = f1 δ in SUM (MAP N⟨HYB⟩ A_VS))
```

$$
\begin{array}{l}
\texttt{else if} \\
\quad f_2\ \delta \neq \emptyset \wedge \forall vs\ \vec{x}\ x.\ (vs,\vec{x}) \in f_2\ \delta \wedge x \in \mathsf{set}\ \vec{x} \Rightarrow \delta\!\downarrow_x \subset \delta \\
\texttt{then} \\
\quad (\texttt{let}\ (vs,\vec{x}) = \mathsf{CHOICE}\ (f_2\ \delta)\ \texttt{in}\ \mathsf{max}\ \{\mathsf{S}\langle \textsc{Hyb}\rangle(x') \mid x' \in \mathbb{U}(\delta\lfloor_{vs})\}) \\
\texttt{else}\ \ell(\delta)
\end{array}
$$

Lastly, proving its validity as an upper bound on the sublist diameter follows directly from HOL4 Theorem 3 and HOL4 Theorem 4.

HOL4 Theorem 5.

$$
\begin{array}{l}
\vdash\ \mathsf{FINITE}\ \delta\ \wedge \\
\quad (\forall \delta'. \\
\qquad \mathsf{dep\text{-}DAG}\ \delta'\ (f_1\ \delta') \wedge \\
\qquad \forall vs\ \vec{x}.\ (vs,\vec{x}) \in f_2\ \delta' \Rightarrow \mathsf{sspace\text{-}DAG}\ \delta'\lfloor_{vs}\ \vec{x}) \Rightarrow \\
\quad \ell(\delta) \leq \textsc{Hyb}\ f_1\ f_2\ \delta
\end{array}
$$

5.4 Formalising the Compositional Reachability Algorithm

In this section we review our formalisation of our algorithm to compute plans via planning for the descriptive quotient. Precisely, we formalise Theorem 17, Theorem 18 and the validity of concatenating instantiations of the descriptive quotient's plan after its goal is augmented. However, in our approach we prove the validity of the algorithm after generalising it in two different ways. Firstly, we prove its validity for a more general class of abstractions: abstractions that are isomorphic to subproblems of the concrete problem, which are more general than the descriptive quotients. Secondly, we prove that the algorithm works for planning problems whose state characterising variables are of arbitrary types, and not necessarily Boolean. This shows that the algorithm has potential application to numerical planning as well as verification of hybrid systems.

An important point to state is that as a result of the formalisation, we found some subtle but non-trivial mistakes in our original algorithm and framework. We will point out those mistakes within our description of our formalisation. We also found that to verify that algorithm, there were other arguments that are easier to formalise than the ones we presented in our earlier proofs, arguments that we shall elaborate on in the next sections.

We first review how we formalise the basic concepts related to planning problems. First we formalise a planning problem as a record type in HOL4 as follows:

$$
\begin{array}{l}
(\alpha,\ \beta)\ \textit{planningProblem}\ = \\
\quad <|\ \mathsf{I}\ :\ (\alpha \mapsto \beta); \\
\qquad \delta\ :\ ((\alpha \mapsto \beta) \times (\alpha \mapsto \beta) \to \textit{bool}); \\
\qquad A\ :\ (\alpha \mapsto \beta)\ |>
\end{array}
$$

Note that this record type extends the sets of actions that we used in our treatment of upper-bounding topological properties. In the above definition $\Pi.\mathsf{I}$ is the initial state, $\Pi.\delta$ is the set of actions a problem has, and $\Pi.A$ is the goal, which is a partial state. Note that this formalisation of planning problems does not require that state characterising variables to be Boolean.

An action sequence is a plan for a planning problem if it satisfies the following predicate:

HOL4 Definition 15 (Plan)**.**

$$\Pi\ \mathsf{solved\_by}\ \vec{\pi} \iff \vec{\pi} \in \Pi.\delta^* \wedge \Pi.A \sqsubseteq \mathsf{ex}(\Pi.\mathsf{I},\vec{\pi})$$

In some cases we require the planning problem to be "valid", which is defined as follows:

HOL4 Definition 16 (Planning Problem)**.**

$$\mathsf{planning\_problem}\ \Pi \iff \Pi.\mathsf{I} \in \mathbb{U}(\Pi.\delta) \wedge \mathcal{D}(\Pi.A) \subseteq \mathcal{D}(\Pi)$$

Also we define the following union operation on planning problems and a lifted union operation for lists of planning problems.

HOL4 Definition 17 (Planning Problem Union)**.**

$$(\Pi_1 \cup \Pi_2) = \\ <|\mathsf{I} := \Pi_1.\mathsf{I} \uplus \Pi_2.\mathsf{I};\ \delta := \Pi_1.\delta \cup \Pi_2.\delta;\ A := \Pi_1.A \uplus \Pi_2.A|>$$

$$\bigcup \Pi_l = \mathsf{FOLDR}\ \mathsf{planning\_prob\_union}\ \Pi_\emptyset\ \Pi_l$$

$\Pi_\emptyset$ *is the empty problem defined as follows*

$$\Pi_\emptyset = <|\mathsf{I} := x_\emptyset;\ \delta := \emptyset;\ A := x_\emptyset|>$$

where $x_\emptyset$ denotes a state with an empty domain, i.e. a state that maps nothing to nothing.

The following theorems show that the semantics of the planning problem union operations are as intended.

$$\vdash \mathsf{planning\_problem}\ \Pi_1 \wedge \mathsf{planning\_problem}\ \Pi_2 \Rightarrow \\ \mathsf{planning\_problem}\ (\Pi_1 \cup \Pi_2)$$

$$\vdash (\forall \Pi.\ \mathsf{MEM}\ \Pi\ \Pi_l \Rightarrow \mathsf{planning\_problem}\ \Pi) \Rightarrow \\ \mathsf{planning\_problem}\ (\bigcup \Pi_l)$$

$$\vdash \mathcal{D}(\Pi_1 \cup \Pi_2) = \mathcal{D}(\Pi_1) \cup \mathcal{D}(\Pi_2)$$

5.4.1 Formalising Soundness of Sub-solution Concatenation (Theorem 17)

We now discuss our formalisation of Lemma 8 from page 60. Before we go into the details of our formalisation, we would like to highlight that in the process of formalisation, we found a mistake in the statement of that lemma as we formulated and published it our International Joint Conference on Artificial Intelligence paper. In the original formulation, we left out the assumption that for every problem Π_i, the corresponding plan $\vec{\pi}_i$ needs to satisfy the condition $\mathsf{sat\text{-}pre}(\mathcal{N}(\Pi_i), \vec{\pi}_i)$, i.e. all the actions in $\vec{\pi}_i$ have to have their preconditions satisfied during the execution of $\vec{\pi}_i$ from the needed assignments of Π_i. Without having this assumption, concatenating the plans for the different problems in the problem list does not necessarily satisfy all the goals for all the problems in the list. This is because there may be some action in a plan $\vec{\pi}_i$, whose conditions are not satisfied by $\mathcal{N}(\Pi_i)$, that gets activated in the concatenated plan, thus jeopardising the outcome of the concatenated plan. Implications of missing this assumption in our original published treatment propagated to the theorems that depend on Lemma 8 and eventually to the main algorithm. In some cases this mistake could have led to the computation of buggy plans, i.e. plans that do not reach the goals they are supposed to reach. Interestingly, this case never showed up in any of the thousands of standard planning benchmarks on which we conducted our experiments. Bugs in such corner cases cannot be afforded if the AI algorithm was to be deployed in a safety sensitive application, and the existence of such bugs further strengthens the argument for using formal verification for AI algorithms.

We now go to the details of the formalisation. To formalise the concept of needed assignments, we took a different approach from the one we took in our pen and paper proof. We first define the needed variables as follows, which are equivalent to $\mathcal{D}(\mathcal{N}(\Pi))$ in the original formulation, i.e. the domain of the needed assignments.

HOL4 Definition 18. *Needed Variables*

$$\begin{array}{l}\mathcal{N}_{\mathcal{D}}(\Pi)\ =\\ \ \{v\ \mid\\ \quad v\ \in\ \mathcal{D}(\Pi\,.\,\mathsf{I})\ \wedge\\ \quad ((\exists p\ e\,.\ (p,e)\ \in\ \Pi\,.\,\delta\ \wedge\ v\ \in\ \mathcal{D}(p)\ \wedge\ p\ `\ v\ =\ \Pi\,.\,\mathsf{I}\ `\ v)\ \vee\\ \quad\ v\ \in\ \mathcal{D}(\Pi\,.\,A)\ \wedge\ \Pi\,.\,\mathsf{I}\ `\ v\ =\ \Pi\,.\,A\ `\ v)\}\end{array}$$

We then formalise needed assignments as follows:

HOL4 Definition 19 (Needed Assignments)**.**

$$\mathcal{N}(\Pi)\ =\ \Pi\,.\,\mathsf{I}\lfloor_{\mathcal{N}_{\mathcal{D}}(\Pi)}$$

The following theorem shows that the definition of the needed assignments has the right semantics. It effectively says that for a problem, a plan will work from any state x that provides the needed assignments of that problem, even if x contradicts the initial state of the problem on the assignments to some state variables. Note the assumption $\mathsf{sat\text{-}pre}\ (\mathcal{N}(\Pi),\overrightarrow{\pi})$: it guarantees that the state x does not unintentionally activate actions that are not active when $\overrightarrow{\pi}$ is executed from $\Pi\,.\,\mathsf{I}$.

HOL4 Lemma 4.

$$\begin{array}{l}\vdash\ \mathsf{planning\_problem}\ \Pi\ \wedge\ \mathcal{N}(\Pi)\ \sqsubseteq\ x\ \wedge\ \mathsf{sat\text{-}pre}\ (\mathcal{N}(\Pi),\overrightarrow{\pi})\ \Rightarrow\\ \qquad \Pi\ \mathsf{solved\_by}\ \overrightarrow{\pi}\ \Rightarrow\ \Pi\,.\,A\ \sqsubseteq\ \mathsf{ex}(x,\overrightarrow{\pi})\end{array}$$

Based on the concept of needed assignments and needed variables, defining the concept of a planning problem that "precedes" another planning problem is straight forward, as follows.

HOL4 Definition 20 (Preceding Problems)**.**

$$\Pi_1\ \lhd\ \Pi_2\ \iff\ \Pi_1\,.\,A\lfloor_{\mathcal{N}_{\mathcal{D}}(\Pi_2)}\ =\ \mathcal{N}(\Pi_2)\lfloor_{\mathcal{D}(\Pi_1)}\ \wedge\ \Pi_1\,.\,A\lfloor_{\mathcal{D}(\Pi_2)}\ =\ \Pi_2\,.\,A\lfloor_{\mathcal{D}(\Pi_1)}$$

The following HOL4 lemmas show that our definition of the precede relation has the right semantics. It guarantees that if a planning problem Π_1 precedes another planning problem Π_2, then the a plan for Π_1 always preserves the needed assignments for Π_2, and a plan for Π_2 does not invalidate a goal for Π_1. This is crucial to show that the concatenation of the plans of the problems will solve the union of the two problems.

HOL4 Lemma 5.

$$\begin{array}{l}\vdash\ \Pi_1\ \lhd\ \Pi_2\ \wedge\ \Pi_1\,.\,A\ \sqsubseteq\ \mathsf{ex}(x,\overrightarrow{\pi})\ \wedge\ \overrightarrow{\pi}\ \in\ \Pi_1.\delta^*\ \wedge\ \mathcal{N}(\Pi_2)\ \sqsubseteq\ x\ \wedge\\ \quad \mathsf{planning\_problem}\ \Pi_1\ \Rightarrow\\ \qquad \mathcal{N}(\Pi_2)\ \sqsubseteq\ \mathsf{ex}(x,\overrightarrow{\pi})\end{array}$$

HOL4 Lemma 6.

$$\begin{array}{l}\vdash\ \Pi_1\ \lhd\ \Pi_2\ \wedge\ \Pi_2\,.\,A\ \sqsubseteq\ \mathsf{ex}(x,\overrightarrow{\pi})\ \wedge\ \overrightarrow{\pi}\ \in\ \Pi_2.\delta^*\ \wedge\ \mathsf{planning\_problem}\ \Pi_2\ \wedge\\ \quad \Pi_1\,.\,A\ \sqsubseteq\ x\ \Rightarrow\\ \qquad \Pi_1\,.\,A\ \sqsubseteq\ \mathsf{ex}(x,\overrightarrow{\pi})\end{array}$$

Another way the formalisation departs from our pen and paper proof, is in the way we specify the situation of a list of planning problems which all precede each other in the order of the list. We do that in the following recursive way.

HOL4 Definition 21 (List of Preceding Problems)**.**

$$\lhd_l\ (\Pi :: \Pi_l) \iff (\forall \Pi'.\ \mathsf{MEM}\ \Pi'\ \Pi_l \Rightarrow \Pi \lhd \Pi') \wedge \lhd_l\ \Pi_l$$
$$\lhd_l\ [\,] \iff \mathsf{T}$$

We are now ready to state our formalised statement of Lemma 8 from page 60, which states sufficient conditions for the concatenation of sub-problem plans to solve the goals for each of the sub-problems.

HOL4 Lemma 7.

$$\begin{array}{l}
\vdash \lhd_l\ \Pi_l \Rightarrow \\
\quad \forall x. \\
\quad\quad (\forall \Pi. \\
\quad\quad\quad \mathsf{MEM}\ \Pi\ \Pi_l \Rightarrow \\
\quad\quad\quad\quad \mathsf{planning_problem}\ \Pi \wedge \Pi.\mathsf{I} \sqsubseteq x \wedge \Pi\ \mathsf{solved_by}\ f\ \Pi \wedge \\
\quad\quad\quad\quad \mathsf{sat\text{-}pre}\ (\mathcal{N}(\Pi), f\ \Pi)) \Rightarrow \\
\quad\quad\quad \forall \Pi.\ \mathsf{MEM}\ \Pi\ \Pi_l \Rightarrow \Pi.A \sqsubseteq \mathsf{ex}(x, \mathsf{FLAT}\ (\mathsf{MAP}\ f\ \Pi_l))
\end{array}$$

Note that in the statement above f is a function that maps every planning problem to a plan that solves it. A significant difference between the formalisation of this lemma and the pen and paper treatment is the proof. An informal version of the proof that we formalised follows.

Proof. In the pen and paper proof we proceed by induction on the size of the list of problems, and effectively add a new member to the end of list of problems, which is equivalent to induction on the reverse of Π_l. In the formalisation we proceed in the opposite direction, where the induction adds a new planning problem as a head to the list of problems Π_l. The base case is trivial. In the step case we have the theorem for list of problems Π_l, and we need to show that it applies to Π_l with the problem Π pre-pended to it. The key idea of the proof is to deal with $\bigcup\ \Pi_l$ as one planning problem. Since Π precedes every problem in Π_l, we have that Π precedes $\bigcup\ \Pi_l$. From this, the inductive hypothesis, HOL4 Lemma 4, HOL4 Lemma 5, and HOL4 Lemma 6, the result follows. □

The formalisation of Theorem 17 follows directly from HOL4 Lemma 7. However, we review our formalisation of concepts necessary to state Theorem 17. First we formalise the concepts of a subproblem and based on it the concept of coverage, whose formalisations are straightforward as follows:

HOL4 Definition 22 (Sub-Problem)**.**

$$\Pi_1 \subseteq \Pi_2 \iff \Pi_1.\mathsf{I} \sqsubseteq \Pi_2.\mathsf{I} \wedge \Pi_1.\delta \subseteq \Pi_2.\delta$$

HOL4 Definition 23. *Covering Problems*

$$\begin{array}{l}
\mathsf{covers}\ \Pi_l\ \Pi \iff \\
\quad (\forall x. \\
\quad\quad x \in \mathcal{D}(\Pi.A) \Rightarrow \\
\quad\quad\quad \exists \Pi'.\ \mathsf{MEM}\ \Pi'\ \Pi_l \wedge x \in \mathcal{D}(\Pi'.A) \wedge \Pi.A\ \text{`}\ x = \Pi'.A\ \text{`}\ x) \wedge \\
\quad \forall \Pi'.\ \mathsf{MEM}\ \Pi'\ \Pi_l \Rightarrow \Pi' \subseteq \Pi
\end{array}$$

During formalisation, we discovered a mistake in our definition of a subproblem: in the original definition we omitted the requirement that $\Pi_1.\mathsf{I} \sqsubseteq \Pi_2.\mathsf{I}$ holds, and instead required that $\mathcal{D}(\Pi_1) \subseteq \mathcal{D}(\Pi_2)$ holds.

Secondly we formalise the function rem-condless as follows. This function removes the actions whose preconditions are not satisfied during execution from a certain state

HOL4 Definition 24 (Remove Condition-less Actions)**.**

$$\begin{array}{l}\mathsf{rem\text{-}condless}\ (x,pfx\_a,(p,e)::\overrightarrow{\pi})\ =\\ \quad \mathtt{if}\ p \sqsubseteq \mathsf{ex}(x,pfx\_a)\ \mathtt{then}\ \mathsf{rem\text{-}condless}\ (x,pfx\_a \mathbin{+\!\!+} [(p,e)],\overrightarrow{\pi})\\ \quad \mathtt{else}\ \mathsf{rem\text{-}condless}\ (x,pfx\_a,\overrightarrow{\pi})\\ \mathsf{rem\text{-}condless}\ (x,pfx\_a,[])\ =\ pfx\_a\end{array}$$

The following theorems show that our formalisation of that function has the desired properties, where the preconditions of all the actions in the resulting action sequence are satisfied, and the resulting action sequence produces the same execution outcome.

$$\vdash\ \mathsf{sat\text{-}pre}\ (x,\mathsf{rem\text{-}condless}\ (x,[],\overrightarrow{\pi}))$$

$$\vdash\ \mathsf{ex}(x,\overrightarrow{\pi})\ =\ \mathsf{ex}(x,\mathsf{rem\text{-}condless}\ (x,[],\overrightarrow{\pi}))$$

Given those definitions, the following formalised statement of Theorem 17 follows directly from HOL4 Lemma 7, and it formalises sufficient conditions for the concatenation of sub-problem plans to solve the union of those sub-problems.

HOL4 Theorem 6.

$$\begin{array}{l}\vdash\ \mathsf{covers}\ \Pi_l\ \Pi\ \wedge\ \lhd_l\ \Pi_l\ \Rightarrow\\ \quad (\forall \Pi.\ \mathsf{MEM}\ \Pi\ \Pi_l\ \Rightarrow\ \mathsf{planning\_problem}\ \Pi\ \wedge\ \Pi\ \mathsf{solved\_by}\ f\ \Pi)\ \Rightarrow\\ \quad\quad \Pi\ \mathsf{solved\_by}\\ \quad\quad \mathsf{FLAT}\ (\mathsf{MAP}\ (\lambda \Pi'.\ \mathsf{rem\text{-}condless}\ (\mathcal{N}(\Pi'),[],f\ \Pi'))\ \Pi_l)\end{array}$$

5.4.2 Formalising Instantiation

The most challenging part of formalising Theorem 18 and the augmentation algorithm is in capturing the instantiation of states and building on top of the instantiation of actions, planning problems, and action sequences. Because we formalise states as finite maps, the instantiation $\pitchfork(\!|x|\!)$ of a state x corresponds to applying an image of the instantiantion $\pitchfork$ to the domain of the finite map, which is why we denote the instantiation operation as a function image application. For example, for a state $\{o_1 \mapsto T, o_2 \mapsto F\}$ and an instantiation function $\pitchfork$, the instantiation of that state using that function is the state $\{\pitchfork(o_1) \mapsto T, \pitchfork(o_2) \mapsto F\}$. This is equivalent to composing the inverse of the instantiation function with the state. The first step in formalising this is to define the inverse of a function.

HOL4 Definition 25 (Function Inverse)**.**

$$f^{-1}\ x\ =\ \mathsf{CHOICE}\ \{y\ |\ f\ y\ =\ x\}$$

In order for the inverse of a function to behave as we desire, that function needs to be a bijection, in which case the following theorems follow.

$$\vdash\ \mathsf{BIJ}\ f\ \mathcal{U}(:\alpha)\ \mathcal{U}(:\beta)\ \Rightarrow\ f\ (f^{-1}\ x)\ =\ x$$

$\vdash$ BIJ f $\mathcal{U}(:\alpha)$ $\mathcal{U}(:\beta)$ $\Rightarrow$ f^{-1} $(f\ x)$ = x

$\vdash$ BIJ f $\mathcal{U}(:\alpha)$ $\mathcal{U}(:\beta)$ $\Rightarrow$ $f^{-1\,-1}$ = f

Note that this bijectivity condition holds for instantiations that are transversals of variable orbits which we considered in Chapter 4. This is because the set of orbits forms a partition of the domain of the planning problem, and transversals map every orbit to one of its members.

Based on this characterisation of an inverse function, the way we formalise state instantiation in HOL4 is as follows:

HOL4 Definition 26 (State Instantiation)**.**

$\pitchfork(\!|x|\!)$ = x f_o $\pitchfork^{-1}$

In HOL4 the term fm f_o f is the composition of a finite map fm with a function f. Based on that, we define the instantiation operation as follows for an action, a factored system, a planning problem and an action sequence.

HOL4 Definition 27 (Action Instantiation)**.**

$\pitchfork(\!|(p,e)|\!)$ = $(\pitchfork(\!|p|\!),\pitchfork(\!|e|\!))$

HOL4 Definition 28 (System Instantiation)**.**

$\pitchfork(\!|\delta|\!)$ = $(\lambda\,\pi.\ \pitchfork(\!|\pi|\!))(\!|\delta|\!)$

HOL4 Definition 29 (Planning Problem Instantiation)**.**

$\pitchfork(\!|\Pi|\!)$ = Π with <| I := $\pitchfork(\!|\Pi.\mathsf{I}|\!)$; δ := $\pitchfork(\!|\Pi.\delta|\!)$; A := $\pitchfork(\!|\Pi.A|\!)$ |>

HOL4 Definition 30 (Action Sequence Instantiation)**.**

$\pitchfork(\!|\vec{\pi}|\!)$ = MAP $(\lambda\,\pi.\ \pitchfork(\!|\pi|\!))$ $\vec{\pi}$

Below are some HOL4 theorems that show some of the properties of instantiations, where valid_inst f means that f is a bijection.

$\vdash$ valid_inst $\pitchfork$ $\Rightarrow$ $\pitchfork(\!|x\downharpoonright_{vs}|\!)$ = $\pitchfork(\!|x|\!)\downharpoonright_{\pitchfork(\!|vs|\!)}$

$\vdash$ valid_inst $\pitchfork$ $\Rightarrow$ ex$(\pitchfork(\!|x|\!),\pitchfork(\!|\vec{\pi}|\!))$ = $\pitchfork(\!|\mathsf{ex}(x,\vec{\pi})|\!)$

$\vdash$ valid_inst $\pitchfork$ $\wedge$ planning_problem Π $\Rightarrow$ planning_problem $\pitchfork(\!|\Pi|\!)$

$\vdash$ valid_inst $\pitchfork$ $\wedge$ Π solved_by $\vec{\pi}$ $\Rightarrow$ $\pitchfork(\!|\Pi|\!)$ solved_by $\pitchfork(\!|\vec{\pi}|\!)$

$\vdash$ valid_inst $\pitchfork$ $\Rightarrow$ $\mathcal{N}_{\mathcal{D}}(\pitchfork(\!|\Pi|\!))$ = $\pitchfork(\!|\mathcal{N}_{\mathcal{D}}(\Pi)|\!)$

In addition to being bijections, an additional property needs to hold for a set of instantiations of a planning problem in order for the algorithm of planning via the quotient to work. That property is that two different instantiations from a set of instantiations Δ should not map different variables from the domain of the quotient to the same variable in their range. This is formally stated as follows.

HOL4 Definition 31 (Valid Instantiation)**.**

$$\begin{array}{l}\mathsf{valid\_instantiations}\ \Delta\ vs\ \Longleftrightarrow \\ \quad \forall \pitchfork_1\ \pitchfork_2\ v_1\ v_2\ . \\ \quad\quad \pitchfork_1 \in \Delta \wedge \pitchfork_2 \in \Delta \wedge v_1 \in vs \wedge v_2 \in vs \wedge v_1 \neq v_2 \Rightarrow \\ \quad\quad\quad \pitchfork_1\ v_1 \neq \pitchfork_2\ v_2\end{array}$$

This condition guarantees that different instantiations of the same state are consistent with each other, i.e. a certain variable is mapped to the same value in all the instantiated states, which is formally stated in the following theorem.

$$\begin{array}{l}\vdash\ \mathsf{valid\_inst}\ \pitchfork_1\ \wedge\ \mathsf{valid\_inst}\ \pitchfork_2\ \wedge \\ \quad \mathsf{valid\_instantiations}\ \{\pitchfork_1\,;\ \pitchfork_2\}\ \mathcal{D}(vs)\ \Rightarrow \\ \quad\quad \mathsf{agree}\ \pitchfork_1(\!|vs|\!)\ \pitchfork_2(\!|vs|\!)\end{array}$$

Note that this condition also holds for instantiations that are transversals of variable orbits, again, since the set of orbits forms a partition of the domain of the planning problem.

5.4.3 Formalising Theorem 18 and the Validity of Goal Augmentation

Now that we formalised instantiations, we describe how we formalised Theorem 18. Below are the formalisations of the concepts of common variables between instantiations and the sustainability relation between planning problems and sets of state variables.

HOL4 Definition 32 (Sustainable Variables)**.**

$$\mathsf{sustainable\_vars}\ \Pi\ vs\ \Longleftrightarrow\ \Pi.I\downarrow_{vs}\ =\ \Pi.A\downarrow_{vs}$$

HOL4 Definition 33 (Common Variables)**.**

$$\begin{array}{l}\bigcap_v\ \Delta\ vs\ = \\ \quad \{\, v \mid \exists \pitchfork_1\ \pitchfork_2\ .\ \pitchfork_1 \in \Delta \wedge \pitchfork_2 \in \Delta \wedge \pitchfork_1 \neq \pitchfork_2 \wedge v \in vs \wedge \pitchfork_1\ v\ =\ \pitchfork_2\ v \,\}\end{array}$$

The following HOL4 lemma summarises the main argument in the proof of Theorem 18, and it shows that the semantics of the definitions above are as intended. It shows that if the needed variables of the quotient are sustained then any two instantiations of that quotient precede each other.

$$\begin{array}{l}\vdash\ \mathsf{valid\_inst}\ \pitchfork_1\ \wedge\ \mathsf{valid\_inst}\ \pitchfork_2\ \wedge \\ \quad \mathsf{valid\_instantiations}\ \Delta\ \mathcal{D}(\Pi)\ \wedge\ \mathsf{planning\_problem}\ \Pi\ \wedge\ \pitchfork_1 \in \Delta\ \wedge \\ \quad \pitchfork_2 \in \Delta \wedge \pitchfork_1 \neq \pitchfork_2 \wedge \mathsf{sustainable\_vars}\ \Pi\ (\bigcap_v\ \Delta\ \mathcal{D}(\Pi) \cap \mathcal{N}_{\mathcal{D}}(\Pi))\ \Rightarrow \\ \quad\quad \pitchfork_1(\!|\Pi|\!)\ \lhd\ \pitchfork_2(\!|\Pi|\!)\end{array}$$

That lemma can then be generalised to a list of instantiations as follows, where similar sufficient conditions on lists of planning problems that precede each other are formalised.

$$\begin{array}{l}\vdash\ \mathsf{ALL\text{-}DISTINCT}\ \Delta\ \wedge\ (\forall \pitchfork .\ \mathsf{MEM}\ \pitchfork\ \Delta\ \Rightarrow\ \mathsf{valid\_inst}\ \pitchfork)\ \wedge \\ \quad \mathsf{valid\_instantiations}\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi)\ \wedge\ \mathsf{planning\_problem}\ \Pi\ \wedge \\ \quad \mathsf{sustainable\_vars}\ \Pi\ (\bigcap_v\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi) \cap \mathcal{N}_{\mathcal{D}}(\Pi))\ \Rightarrow \\ \quad\quad \lhd_l\ (\mathsf{MAP}\ (\lambda \pitchfork .\ \pitchfork(\!|\Pi|\!))\ \Delta)\end{array}$$

From this and from HOL4 Theorem 6, we can derive the following theorem. It is a formalisation of Theorem 18, the main theorem that states sufficient soundness conditions for planning via the descriptive quotient.

HOL4 Theorem 7.

$$
\begin{array}{l}
\vdash \mathsf{ALL\text{-}DISTINCT}\ \Delta\ \wedge \\
\quad \mathsf{INJ}\ (\lambda \pitchfork.\ \pitchfork(\!|\Pi'|\!))\ (\mathsf{set}\ \Delta)\ \mathcal{U}(:(\alpha,\ \beta)\ \mathit{planningProblem})\ \wedge \\
\quad (\forall \pitchfork.\ \mathsf{MEM}\ \pitchfork\ \Delta\ \Rightarrow\ \mathsf{valid\_inst}\ \pitchfork)\ \wedge \\
\quad \mathsf{valid\_instantiations}\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi')\ \wedge\ \mathsf{planning\_problem}\ \Pi'\ \wedge \\
\quad \mathsf{sustainable\_vars}\ \Pi'\ (\bigcap_v\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi')\ \cap\ \mathcal{N}_{\mathcal{D}}(\Pi'))\ \wedge \\
\quad \mathsf{covers}\ (\mathsf{MAP}\ (\lambda \pitchfork.\ \pitchfork(\!|\Pi'|\!))\ \Delta)\ \Pi\ \wedge\ \Pi'\ \mathsf{solved\_by}\ \vec{\pi}'\ \Rightarrow \\
\quad\quad \Pi\ \mathsf{solved\_by} \\
\quad\quad \mathsf{FLAT}\ (\mathsf{MAP}\ (\lambda \pitchfork.\ \mathsf{rem\text{-}condless}\ (\mathcal{N}(\pitchfork(\!|\Pi'|\!)), [\,], \pitchfork(\!|\vec{\pi}'|\!)))\ \Delta)
\end{array}
$$

The formalised version of Theorem 18 is more general in that it does not assume that the quotient problem is necessarily a descriptive quotient obtained through symmetry analysis. It rather assumes that the quotient can be instantiated into subproblems of the concrete planning problem that cover its goals, i.e. the quotient is isomorphic to subproblems of the given planning problem. It also should be clear that the concrete planning problem and the quotient are not necessarily propositionally factored systems, i.e. the state variables are not necessarily Boolean and are not required to have finite or countable domains. This makes the planning via descriptive quotients a candidate approach to be used with hybrid systems.

The last step in our formalisation is to verify that instantiations of a plan for a quotient whose goal is augmented, as per our approach described in Section 4.6, can be concatenated into a plan for the concrete planning problem. The first step in doing so is to show that augmenting the quotient's goal with the common needed assignments guarantees that it sustains those assignments. This is shown in the following theorem.

$$
\begin{array}{l}
\vdash \mathtt{let} \\
\quad\quad \Pi^q\ \mathtt{=}\ \Pi'\ \mathtt{with}\ A\ \mathtt{:=}\ \Pi'.\mathsf{I}|_{\bigcap_v\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi')\ \cap\ \mathcal{N}_{\mathcal{D}}(\Pi')}\ \uplus\ \Pi'.A \\
\quad \mathtt{in} \\
\quad\quad \mathsf{sustainable\_vars}\ \Pi^q\ (\bigcap_v\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi^q)\ \cap\ \mathcal{N}_{\mathcal{D}}(\Pi^q))
\end{array}
$$

It is worth noting that HOL4 is able to prove the above goal entirely automatically.

Now that we proved that goal augmentation works as intended, the following theorem follows, which verifies the validity of our approach of planning using the descriptive quotient. The main idea in that theorem is that, since a planning problem with an augmented goal sustains the common needed variables, as we show in the lemma above, then the same algorithm as in HOL4 Theorem 7 can always be used the augmented quotient problem.

$$
\begin{array}{l}
\vdash \mathtt{let} \\
\quad\quad \Pi^q\ \mathtt{=}\ \Pi'\ \mathtt{with}\ A\ \mathtt{:=}\ \Pi'.\mathsf{I}|_{\bigcap_v\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi')\ \cap\ \mathcal{N}_{\mathcal{D}}(\Pi')}\ \uplus\ \Pi'.A \\
\quad \mathtt{in} \\
\quad\quad \mathsf{ALL\text{-}DISTINCT}\ \Delta\ \wedge\ (\forall \pitchfork.\ \mathsf{MEM}\ \pitchfork\ \Delta\ \Rightarrow\ \mathsf{valid\_inst}\ \pitchfork)\ \wedge \\
\quad\quad \mathsf{planning\_problem}\ \Pi'\ \Rightarrow \\
\quad\quad\quad \mathsf{INJ}\ (\lambda \pitchfork.\ \pitchfork(\!|\Pi^q|\!))\ (\mathsf{set}\ \Delta)\ \mathcal{U}(:(\alpha,\ \beta)\ \mathit{planningProblem})\ \wedge \\
\quad\quad\quad \mathsf{valid\_instantiations}\ (\mathsf{set}\ \Delta)\ \mathcal{D}(\Pi^q)\ \wedge \\
\quad\quad\quad \mathsf{covers}\ (\mathsf{MAP}\ (\lambda \pitchfork.\ \pitchfork(\!|\Pi^q|\!))\ \Delta)\ \Pi\ \wedge\ \Pi^q\ \mathsf{solved\_by}\ \vec{\pi}^q\ \Rightarrow \\
\quad\quad\quad\quad \Pi\ \mathsf{solved\_by} \\
\quad\quad\quad\quad \mathsf{FLAT}\ (\mathsf{MAP}\ (\lambda \pitchfork.\ \mathsf{rem\text{-}condless}\ (\mathcal{N}(\pitchfork(\!|\Pi^q|\!)), [\,], \pitchfork(\!|\vec{\pi}^q|\!)))\ \Delta)
\end{array}
$$

5.5 Concluding Remarks

Much of the previous work on formalised algorithms related to transition systems, like Esparza et al. (2013), focus on algorithms that target model checking applications. Although the algorithms

that we introduce and formally verify here apply to *propositionally factored transition systems* in their full generality, we focus more on AI planning applications. Thus, with this work, we believe we have launched a fruitful inter-disciplinary collaboration between the fields of AI planning and mechanised mathematics. From the interactive theorem-proving community's point of view, it is gratifying to be able to find and fix errors in the modern research literature. For example, the insights which led us to develop the sublist diameter and the top-down algorithm followed from attempting to formalise the results in Rintanen and Gretton (2013) . That effort helped us find a bug in their algorithm, where they incorrectly theorise that the diameter can be compositionally upper-bounded by projections' diameters, if the projections are induced by a partition of state variables whose members are closed under mutual dependency.

Also during formalisation, we found mistakes in our own algorithms that are subtle but non-trivial. For instance, in our published version and implementation of the algorithm for planning via the descriptive quotient, we missed an important step in the algorithm. That step is removing actions from the quotient's plan whose preconditions are not satisfied, before the instantiation and concatentation steps of the quotient's plan, which can possibly produce an invalid plan. Interestingly, this error never showed up during experimentation, although we ran our algorithm on thousands of diverse standard planning benchmarks. Although such errors can be practically rare, they cannot be tolerated in safety critical applications, thus, their existence makes a strong case for the utility of mechanical verification.

It is vital that we give AI researchers assurances that their algorithms, theory and systems are correct. For AI planning systems to be deployed in safety critical applications and for autonomous exploration of space, they must not only be efficient, and provably conservative in their resource consumption, but also correct. The upper-bounding algorithms like the ones we formalise here underpin fixed-horizon planning; indeed, they provide the fixed horizon past which an algorithm need not search. If an autonomous vehicle exploring outer space implemented diameter-based compositional bounding suggested by Rintanen and Gretton (2013) to do its plan-search, that system could incorrectly conclude that no plan exists.

Another point we would like to raise is one concerning the difference between the formal proofs and the informal proofs. We oftentimes found that some proofs were more intuitive and accordingly easier to explain to deliver the gist of how a proof works. In contrast, we sometimes found it very hard to formalise those proofs and found that substantially different approaches were more natural and more fit for formalisation. An example of that is the proof of Lemma 8 and its formalised equivalent HOL4 Lemma 7. We found that to deliver the argument, doing the induction on the tail of the list is more natural and clearer, while it was substantially much easier to formalise the proof by doing the induction on the head of the list.

Furthermore, our formalisation made it much easier to generalise some of our results and algorithms from propositionally factored systems to more general systems, systems in which the codomains of states are not necessarily Boolean, finite or even countable. This raises the possibility of applying the algorithms that we developed to hybrid systems. Also, we would like to note that for our formalisation we developed a large formalised library on factored transition systems. Since a lot of theory in that library applies to factored systems that are not propositionally factored, it can be used for verifying algorithms on hybrid systems, for instance. In the case of hybrid systems, an interesting challenge would be extending the theory we developed to be capable of representing actions whose preconditions and effects are functions in state variables, versus assignments to state variables.

Our experience mechanising results in AI planning allows us to provide insights regarding the scalability of formalising AI planning algorithms. To formalise the compositional algorithms we developed a library of HOL4 proof scripts that is around $14k$ lines long, including comments.

Interestingly, the size of developed code does not necessarily scale linearly with the number of algorithms that we formalised. The first algorithm that we formalised was $\mathsf{N}_{\mathsf{sum}}$, and to do so we developed around $10k$ lines of proof script. That script was developed in approximately six months. Our subsequent formalisation of $\mathsf{S}_{\mathsf{max}}$ and HYB required an additional $2k$ lines of proof scripts each. Each of those algorithms took around two and half weeks to formalise. This productivity improvement follows because when we formalised $\mathsf{N}_{\mathsf{sum}}$, we developed the majority of the needed formal background theory. That theory is leveraged in our formalisation of other algorithms on factored transition systems.

We made a number of observations in our efforts that we believe provide insight into how HOL4 can be improved. The feature of HOL4 that we would cite as the most positive, is the ability to quickly modify existing tactics, or add new tactics, since the entire system is completely implemented in SML. Also, automation tactics in general are reasonable. Nonetheless, we think that other aspects of automation can still be improved. Tasks that can be automated which we found cumbersome include: (i) searching for theorems in the library, (ii) the generation of termination conditions, (iii) the definition of relations, (iv) proving the uniqueness of relations, and (v) deriving the function form of a relation. Another more general issue that we faced is the absence of a mechanism akin to classes in Isabelle, which could have allowed us to reduce the repetition of theorem hypotheses.

In terms of formally verifying AI planning algorithms, we have only scratched the surface here. For instance, when a tight bound for a planning problem is known, one effective technique for finding a plan is to reduce that problem to SAT (Rintanen, 2012). Proposed reductions are constructive, in the sense that a plan can be constructed in linear time from a satisfying assignment to a formula. A key recent advance in the setting of planning-via-SAT has been the development of compact SAT-representations of planning problems. Such representations facilitate more efficient search (Rintanen, 2012; Robinson et al., 2009). In future work, we would like to verify the correctness of both the reductions to SAT, and the algorithms that subsequently construct plans from a satisfying assignment.

Bibliography

Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 377–391. SIAM, 2016.

Mohammad Abdulaziz and Lawrence C Paulson. An Isabelle/HOL formalisation of Green's theorem. In *International Conference on Interactive Theorem Proving*, pages 3–19. Springer, 2016.

Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4): 1167–1181, 1999.

Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4): 844–856, 1995.

Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

Rajeev Alur, Robert K Brayton, Thomas A Henzinger, Shaz Qadeer, and Sriram K Rajamani. Partial-order reduction in symbolic state space exploration. In *International Conference on Computer Aided Verification*, pages 340–351. Springer, 1997.

Eyal Amir and Barbara Engelhardt. Factored planning. In *IJCAI*, volume 3, pages 929–935. Citeseer, 2003.

Andreas Bauer, Peter Baumgartner, Martin Diller, and Michael Norrish. Tableaux for verification of data-centric processes. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 28–43. Springer, 2013.

Jason Baumgartner, Andreas Kuehlmann, and Jacob Abraham. Property checking via structural analysis. In *Computer Aided Verification*, pages 151–165. Springer, 2002.

Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, pages 87–90, 1958.

Sergey Berezin, Sérgio Campos, and Edmund M Clarke. Compositional reasoning in model checking. In *Compositionality: The Significant Difference*, pages 81–102. Springer, 1998.

Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. springer, 2004.

Marc Bezem and Dimitri Hendriks. On the mechanization of the proof of Hessenberg's theorem in coherent logic. *Journal of Automated Reasoning*, 40(1):61–85, 2008.

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207, 1999.

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

Andreas Björklund and Thore Husfeldt. Finding a path of superlogarithmic length. *SIAM Journal on Computing*, 32(6):1395–1402, 2003.

Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In *International Colloquium on Automata, Languages, and Programming*, pages 222–233. Springer, 2004.

Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *Interactive Theorem Proving, First International Conference, ITP 2010*, pages 131–146, 2010. doi: 10.1007/978-3-642-14052-5_11.

Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. In *International Joint Conference on Automated Reasoning*, pages 25–44. Springer, 2016.

Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300, 1997.

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33, 2001.

R. I. Brafman and C. Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003.

Ronen I Brafman and Carmel Domshlak. Factored planning: How, when, and when not. In *AAAI*, volume 6, pages 809–814, 2006.

Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom Jr. Backtrack searching in the presence of symmetry. *Nord. J. Comput.*, 3(3):203–219, 1996.

Daniel Bundala, Joël Ouaknine, and James Worrell. On the magnitude of completeness thresholds in bounded model checking. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 155–164. IEEE Computer Society, 2012.

T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

Michael L. Case, Hari Mony, Jason Baumgartner, and Robert Kanzelman. Enhanced verification by temporal decomposition. In *FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*, pages 17–24, 2009.

Timothy M Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

Shiri Chechik, Daniel H Larkin, Liam Roditty, Grant Schoenebeck, Robert E Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1041–1052. Society for Industrial and Applied Mathematics, 2014.

Yixin Chen and Guohui Yao. Completeness and optimality preserving reduction for planning. In *IJCAI*, pages 1659–1664, 2009.

Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Progress on the state explosion problem in model checking. In *Informatics*, pages 176–194. Springer, 2001.

Edmund Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 85–96. Springer, 2004.

Edmund M Clarke, Orna Grumberg, and David E Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

Edmund M Clarke, Reinhard Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1-2):77–104, 1996.

Edmund M Clarke, E Allen Emerson, Somesh Jha, and A Prasad Sistla. Symmetry reductions in model checking. In *Computer Aided Verification*, pages 147–158. Springer, 1998.

Edmund M Clarke, E Allen Emerson, and Joseph Sifakis. Turing lecture: model checking–algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.

Robert L Constable, Paul B Jackson, Pavel Naumov, and Juan C Uribe. Constructively formalizing automata theory. In *Proof, language, and interaction*, pages 213–238, 2000.

James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. *KR*, 96:148–159, 1996.

Peter Dankelmann. The diameter of directed graphs. *Journal of Combinatorial Theory, Series B*, 94(1):183–186, 2005.

Peter Dankelmann and Michael Dorfling. Diameter and maximum degree in Eulerian digraphs. *Discrete Mathematics*, 339(4):1355–1361, 2016.

Peter Dankelmann and Lutz Volkmann. The diameter of almost Eulerian digraphs. *the electronic journal of combinatorics*, 17(1):R157, 2010.

Leonardo De Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proc. TACAS'08/ETAPS'08*, pages 337–340, 2008.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Christian Doczkal, Jan-Oliver Kaiser, and Gert Smolka. A constructive theory of regular languages in Coq. In *International Conference on Certified Programs and Proofs*, pages 82–97. Springer, 2013.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In *ICAPS*, 2012.

Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Symmetry breaking: Satisficing planning and landmark heuristics. In *ICAPS*, 2013.

Jean-François Dufourd. An intuitionistic proof of a discrete form of the Jordan Curve Theorem formalized in Coq with combinatorial hypermaps. *Journal of Automated Reasoning*, 43(1): 19–51, 2009.

E Allen Emerson and A Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1):105–131, 1996.

Paul Erdős, Janos Pach, Richard Pollack, and Zsolt Tuza. Radius, diameter, and minimum degree. *Journal of Combinatorial Theory, Series B*, 47(1):73–79, 1989.

Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. In *International Conference on Computer Aided Verification*, pages 463–478. Springer, 2013.

M Eugene. Permutation groups and polynomial-time computation. In *Groups and Computation: Workshop on Groups and Computation, October 7-10, 1991*, volume 11, page 139. American Mathematical Soc., 1993.

Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 956–961, 1999.

Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 83–91, 2002.

Michael L Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5(1):83–89, 1976.

Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

Ruben A Gamboa. A formalization of powerlist algebra in ACL2. *Journal of Automated Reasoning*, 43(2):139–172, 2009.

Rob Gerth, Ruurd Kuiper, Doron Peled, and Wojciech Penczek. A partial order approach to branching time logic model checking. In *Theory of Computing and Systems, 1995. Proceedings., Third Israel Symposium on the*, pages 130–139. IEEE, 1995.

Patrice Godefroid. Using partial orders to improve automatic verification methods. In *International Conference on Computer Aided Verification*, pages 176–185. Springer, 1990.

Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

Michael JC Gordon and Tom F Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

Emmanuel Guere and Rachid Alami. One action is enough to plan. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 439–444, 2001.

Thomas C Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. In *The Kepler Conjecture*, pages 341–376. Springer, 2011.

John Harrison. Formalizing basic complex analysis. *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar and Rhetoric*, 10(23):151–165, 2007.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006a.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61 (3):16, 2014.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006. ISBN 978-0-262-10114-1.

Mathieu Jaume. A full formalization of SLD-resolution in the calculus of inductive constructions. *Journal of Automated Reasoning*, 23(3):347–371, 1999.

David Joslin and Amitabha Roy. Exploiting symmetry in lifted CSPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 197–202, 1997.

Fairouz Kamareddine and Haiyan Qiao. Formalizing strong normalization proofs of explicit substitution calculi in ALF. *Journal of Automated Reasoning*, 30(1):59–98, 2003.

H. A. Kautz and B. Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux. Factored planning using decomposition trees. In *IJCAI*, pages 1942–1947, 2007.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68 (2):243–302, 1994.

AV Knyazev. Diameters of pseudosymmetric graphs. *Mathematical Notes*, 41(6):473–482, 1987.

Daniel Kroening. Computing over-approximations with bounded model checking. *Electronic Notes in Theoretical Computer Science*, 144(1):79–92, 2006.

Daniel Kroening and Ofer Strichman. Efficient computation of recurrence diameters. In *VMCAI*, pages 298–309, 2003.

Daniel Kroening, Joël Ouaknine, Ofer Strichman, Thomas Wahl, and James Worrell. Linear completeness thresholds for bounded model checking. In *Computer Aided Verification*, pages 557–572. Springer, 2011.

Panagiotis Manolios and Daron Vroon. Ordinal arithmetic: Algorithms and mechanization. *Journal of Automated Reasoning*, 34(4):387–423, 2005.

Filip Marić. Formalization and implementation of modern SAT solvers. *Journal of Automated Reasoning*, 43(1), 2009.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL: The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Drew V McDermott. A heuristic estimator for means-ends analysis in planning. In *AIPS*, volume 96, pages 142–149, 1996.

Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014. ISSN 0747-7171. doi: http://dx.doi.org/10.1016/j.jsc.2013.09.003. URL `http://www.sciencedirect.com/science/article/pii/S0747717113001193`.

James McKinna and Robert Pollack. Some lambda calculus and type theory formalized. *Journal of Automated Reasoning*, 23(3):373–409, 1999.

Kenneth L McMillan. Symbolic model checking. In *Symbolic Model Checking*, pages 25–60. Springer, 1993.

Ian Miguel. Symmetry-breaking in planning: Schematic constraints. In *Proceedings of the CP'01 Workshop on Symmetry in Constraints*, pages 17–24, 2001.

John W Moon et al. On the diameter of a graph. *The Michigan Mathematical Journal*, 12(3): 349–351, 1965.

Tobias Nipkow. Verifying a hotel key card system. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *Theoretical Aspects of Computing (ICTAC 2006)*, volume 4281 of *Lecture Notes in Computer Science*. Springer, 2006. Invited paper.

Tobias Nipkow and Lawrence C Paulson. Proof pearl: Defining functions over finite sets. In *International Conference on Theorem Proving in Higher Order Logics*, pages 385–396. Springer, 2005.

Panos M Pardalos and Athanasios Migdalas. A note on the complexity of longest path problems related to graph coloring. *Applied mathematics letters*, 17(1):13–15, 2004.

Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer, 1994.

Lawrence C Paulson. A formalisation of finite automata using hereditarily finite sets. In *International Conference on Automated Deduction*, pages 231–245. Springer, 2015.

Nir Pochter, Aviv Zohar, and Jeffrey S Rosenschein. Exploiting problem symmetries in state-based planners. In *AAAI*, 2011.

Aldo Porco, Alejandro Machado, and Blai Bonet. Automatic reductions from PH into STRIPS or how to generate short problems with very long solutions. In *ICAPS*, 2013.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, 2010.

Jussi Rintanen. Symmetry reduction for SAT representations of transition systems. In *ICAPS*, pages 32–41, 2003.

Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

Jussi Rintanen and Charles Orgill Gretton. Computing upper bounds on lengths of transition sequences. In *International Joint Conference on Artificial Intelligence*, 2013.

Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. SAT-based parallel planning using a split representation of actions. In *ICAPS*, 2009.

Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524. ACM, 2013.

Alexander Schimpf, Stephan Merz, and Jan-Georg Smaus. Construction of Büchi automata for LTL model checking verified in Isabelle/HOL. In *International Conference on Theorem Proving in Higher Order Logics*, pages 424–439. Springer, 2009.

Jendrik Seipp, Florian Pommerening, Silvan Sievers, Martin Wehrle, Chris Fawcett, and Yusra Alkhazraji. Fast Downward Aidos. *International Planning Competition*, 2014.

Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, pages 108–125, 2000. doi: 10.1007/3-540-40922-X_8.

Alexander Shleyfman, Michael Katz, Malte Helmert, Silvan Sievers, and Martin Wehrle. Heuristics and symmetries in classical planning. In *AAAI*, 2015.

Silvan Sievers, Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz. Factored symmetries for merge-and-shrink abstractions. In *Proc. 29th National Conf. on Artificial Intelligence*. AAAI Press, 2015.

A Prasad Sistla and Edmund M Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)*, 32(3):733–749, 1985.

Konrad Slind and Michael Norrish. A brief overview of HOL4. In *Theorem Proving in Higher Order Logics*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.

José Soares. Maximum diameter of regular digraphs. *Journal of Graph Theory*, 16(5):437–450, 1992.

Christoph Sprenger. A verified model checker for the modal μ-calculus in Coq. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 167–183. Springer, 1998.

Paul Taylor. The fixed point property in synthetic domain theory. In *Logic in Computer Science, 1991. LICS'91., Proceedings of Sixth Annual IEEE Symposium on*, pages 152–160. IEEE, 1991.

Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 149–158. ACM, 2003.

Thomas Wahl and Alastair F. Donaldson. Replication and abstraction: Symmetry in automated formal verification. *Symmetry*, 2(2):799–847, 2010. doi: 10.3390/sym2020799. URL `http://dx.doi.org/10.3390/sym2020799`.

Brian C. Williams and P. Pandurang Nayak. A reactive planner for a model-based executive. In *International Joint Conference on Artificial Intelligence*, pages 1178–1185. Morgan Kaufmann Publishers, 1997.

Chunhan Wu, Xingyuan Zhang, and Christian Urban. A formalisation of the Myhill-Nerode theorem based on regular expressions (proof pearl). In *International Conference on Interactive Theorem Proving*, pages 341–356. Springer, 2011.

You Xu, Yixin Chen, Qiang Lu, and Ruoyun Huang. Theory and algorithms for partial order based reduction in planning. *arXiv preprint arXiv:1106.5427*, 2011.

Raphael Yuster. Computing the diameter polynomially faster than APSP. *arXiv preprint arXiv:1011.6181*, 2010.