Noname manuscript No.
(will be inserted by the editor)

Formally Verified Algorithms for Upper-Bounding State Space Diameters

Mohammad Abdulaziz · Michael Norrish · Charles Gretton

Received: date / Accepted: date

Abstract A *completeness threshold* is required to guarantee the completeness of planning as satisfiability, and bounded model checking of safety properties. We investigate completeness thresholds related to the *diameter* of the underlying transition system. A valid threshold, the diameter is the maximum element in the set of lengths of all shortest paths between pairs of states. The diameter is not calculated exactly in our setting, where the transition system is succinctly described using a (propositionally) factored representation. Rather, an upper bound on the diameter is calculated compositionally, by bounding the diameters of small abstract subsystems, and then composing those.

We describe our formal verification in HOL4 of compositional algorithms for computing a relatively tight upper bound on the system diameter. Existing compositional algorithms are characterised in terms of the problem structures they exploit, including acyclicity in state-variable *dependencies*, and acyclicity in the state space. Such algorithms are further distinguished by: (i) whether the bound calculated for abstractions is the diameter, *sublist* diameter or *recurrence* diameter, and (ii) the "direction" of traversal of the compositional structure, either top-down or bottom-up. As a supplement, we publish our library—now over 14k lines—of HOL4 proof scripts about transition systems. That shall be of use to future related mechanisation efforts, and is carefully designed for compatibility with hybrid systems.

Keywords Formal Verification · Diameter · Transition Systems · Completeness Threshold · AI Planning · SAT-based Planning · Bounded Model Checking

Mohammad Abdulaziz
Technical University of Munich
E-mail: mohammad.abdulaziz@in.tum.de

Michael Norrish
Australian National University and Data61 Canberra Research Lab.
E-mail: michael.norrish@data61.csiro.au

Charles Gretton
Australian National University and Griffith University
E-mail: charles.gretton@anu.edu.au

1 Introduction

The state spaces of problems in fields such as artificial intelligence (AI) planning and model checking can be modelled as digraphs (a.k.a. directed acyclic graphs), where vertices and edges correspond to states and transitions, respectively. Such a digraph is represented as a *propositionally factored transition system* in a language such as STRIPS, by Fikes and Nilsson [29], or SMV, by McMillan et al. [40]. Factored representations have the advantage of being smaller, sometimes exponentially, than an explicit representation of the corresponding digraph. Compact factored representations are necessary because of the enormous state spaces of realistic systems.

The *diameter* of a digraph is a well studied topological property that is conceptually important in this setting. It is equal to the maximum element in the set of lengths of all shortest paths between pairs of vertices – i.e. the length of a longest shortest path. The diameter is a solution-length bound, an ingredient that can guarantee the completeness of powerful solution procedures. For example, the diameter corresponds to a *completeness threshold* in planning as satisfiability, see Kautz et al. [34], and bounded model checking of safety properties, see Biere et al. [11].

A number of algorithms have been proposed to compute the diameter. State-of-the-art algorithms have a computation time worse than quadratic in the size of the digraph [5, 57]. Such algorithms suppose: (i) the digraph is available in an explicit form (i.e. not in a factored form), or (ii) that the diagraph is constructed explicitly during the diameter calculation. Consequently they are rarely applicable in our setting, where factored representations of digraphs with 2^{1000} vertices are not uncommon. Called the state space explosion problem, this has important consequences for a variety of reasoning tasks, and is discussed more broadly by Clarke et al. [19]. Due to high runtime cost and an inability to scale to super-large digraphs, the setting of planning and model checking necessitates an indirect approach to calculating a solution-length bound.

Discovery and exploitation of compositional structure underlies a number of important solution procedures that reason about factored transition systems [9, 16, 21, 30, 32, 35, 55]. The only known feasible approaches for computing completeness thresholds are compositional in the sense that the diameter is approximated by composing bounds computed for subproblems [2, 8, 37, 47]. In this work we formally verify compositional algorithms that over-approximate the diameter. Such algorithms upper-bound the state space diameters of a set of subproblems that correspond to *abstractions* of the concrete system. Those bounds are composed to produce an upper bound on the diameter of that concrete system. The main advantage of this approach is that abstractions have substantially smaller state spaces.[1] In general it is not always the case that such an approach is usefully applicable to the given transition system. Nonetheless, there are structures whose presence in the transition system make it possible to compositionally upper-bound the diameter. We consider compositional algorithms that exploit acyclicity in "state-variable dependencies" [3], and acyclic-

[1] An abstraction's state space corresponds to a *minor* of the digraph modelling the state space of the concrete system. A minor is derived by a sequence of vertex contractions, as well as edge and vertex deletions. Vertex contraction of two vertices is an operation that replaces the two vertices with one new vertex that inherits their incoming and outgoing edges.

ity in the state space [4]. We find such structures to be abundant in AI planning and verification benchmarks.

This work describes our formal verification of compositional bounding algorithms using the interactive theorem prover HOL4 [52]. Using that proof assistant we discovered many mistakes in our work at different stages, as well as mistakes in the literature. Indeed, many of the algorithms we treat were invented by us, and followed from insights gained during proof mechanisation work that helped us substantially strengthen the compositional approach. We would describe this as a "computer-aided algorithm design" process that started with the objective of proving the soundness of theorems and algorithms from the literature.

In addition to helping us to develop our algorithms, and giving us extra assurance about our results, we believe that the formalisation of AI algorithms is of great utility. This is especially true if those algorithms are to be deployed in safety critical applications, or in autonomous exploration of extraterrestrial space. The justifications for requiring formal correctness guarantees are twofold. 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.

Along with this manuscript we publish a rich formal proof library about transition systems. Many theorems in our formalisation apply to infinite state transition systems. Accordingly, we anticipate that our library shall be used for applications beyond the algorithms that we verify in this work, like algorithms for hybrid systems, for instance.

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 such that, at a given time, each room opens with one element from a set of keys. An assignment to state variables will model, among other things, whether or not a specific room can be accessed using a specific key.

In the factored representation multiple transitions (i.e. edges) are factored into actions that identify which states are directly accessible from 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 5), as naturally.

Abstractions Compositional approaches involve solving given problems for state spaces of "abstractions" of the given system, which are minors of the digraph modelling the

state space. Abstractions that we consider are *projection* over a set of state variables (see Section 4.1), and *snapshotting* over an assignment of a set of state variables (see Section 6). In projection, state variables are removed from the factored system. This is equivalent to repetitively merging pairs of vertices in the state space with the resulting vertex inheriting all the edges of its constituents, i.e. vertex contraction. In snapshotting, we exclude form the factored system the actions whose preconditions or effects violate a partial assignment to state variables. This can be understood as consecutive edge and vertex deletions in the 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 variables, where a set vs_2 depends on vs_1 iff vs_2 has a variable that depends on some variable in vs_1. A practically important class of projections are ones done on partitions of the state variables whose members are closed under mutual dependency, i.e. dependencies between different equivalence classes of variables are acyclic. Those partitions capture the abundantly present acyclic dependencies between different modules of real-world systems, such as different circuit components.

1.1 Contributions

We have formally verified compositional algorithms that compute a completeness threshold. Here, a threshold is an upper bound on the diameter of the state space of a propositionally factored transition system. Our proofs of the validity of these algorithms are done using the interactive theorem prover HOL4. This work is the first publication of the details of the proof mechanisation work associated with our previous conference papers, [3] and [4].

The first algorithm is the *top-down algorithm*. It exploits acyclic variable dependencies to compositionally upper-bound the diameter by using *recurrence diameters* of projections. The *recurrence diameter* is the length of the longest simple (i.e. loop-free) path. That algorithm was introduced in [3], where we showed that it outperforms the best algorithm of the time—the *bottom-up algorithm*—in terms of tightness of computed bounds. In [3] we briefly presented a preliminary form of the formalisation of the top-down algorithm. In this work we provide a complete formalisation of that algorithm. We have also generalised much of the theory underlying that formalisation, so that many of the general theorems now apply to systems whose state variables are not necessarily Boolean and whose state spaces are not necessarily finite.

We introduced the two other algorithms in [4]. The second compositional algorithm exploits acyclicity in the state space. The third algorithm is a hybrid algorithm that combines the top-down algorithm and the compositional algorithm for acyclic state spaces in a mutually recursive way. In [4] we showed that the hybrid algorithm produces bounds that are exponentially tighter than the top-down algorithm. In this work we formally verify both algorithms in HOL4.

In order to formally verify the correctness of our compositional algorithms we built a library of formal proofs about factored transition systems in the Higher Order

Logic (HOL) interactive theorem prover HOL4, by Slind and Norrish [52]. 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. [28, 44]) focuses on classical textbook results on finite automata, and reachability of states within those automata in the setting of model checking LTL formulae. 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.

1.2 Paper Structure

This paper is organised as follows. In Section 2 we discuss previous related work on upper-bounding or computing the diameter. In Section 3 we formalise the basic concepts related to factored transition systems, action execution semantics and their formalisation. In Section 4 we discuss background on existing algorithms for exploiting acyclic dependency. In Section 5 we describe our formalisation of the top-down algorithm. In Section 6 we describe our formalisation of our upper-bounding algorithm that exploits acyclicity in the state space. Section 7 gives our hybrid algorithm and the formalisation of that. Lastly, in Section 8 we present concluding remarks on our experience in formalising our algorithms and possible directions for future research.

2 Related Work

2.1 Diameter and Bounding Algorithms

Computing the diameter exactly can be done by 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 et al. [31], Alon et al. [6], Chan et al. [17], and Rafael Yuster [57]). Furthermore, Roditty et al. [49] showed that there is strong evidence that the diameter cannot be computed in time better than quadratic. 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. [5] 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 et al. [49], Chechick et al. [18] and Aboud et al. [1]. To our knowledge all diameter approximation techniques studied in the algorithms community are lower bounding techniques.

Treating upper bounds on diameters, Paradalos et al. [43] showed that computing the recurrence diameter is an NP-hard problem, with the only known exact solutions taking 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. [13], 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 the ones by Alon et al. [7] and Björklund et al. [12], can only find paths logarithmic in the length of the longest path.

In the graph theory and the combinatorics communities, diameters of undirected graphs have been extensively studied since 1965, where work by Moon et al. [41], Erdős et al. [27], and Knyazev [36] computed upper bounds for different classes of undirected graphs. In the case of digraphs work on upper-bounding diameters started more recently. In 1992 Soares [53] provided a tight upper bound on the diameter of biregular digraphs. Then starting in 2000, Dankelmann et al. treated different structures of digraphs [23, 24, 25].

The completeness of bounded model-checking (Biere et al. [10, 11]) and other model checking methods [45], and satisfiability (SAT) based planning (Kautz et al. [34]) depends on having an upper bound on the diameter or the recurrence diameter (depending on the verification problem at hand). 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. Studies on this topic have been undertaken by Biere et al. [10, 11], Kroening et al. [38, 39], Sheeran et al. [51], Bundala et al. [15], and Clarke [20]. The dominant approach to compute the diameter or the recurrence diameter in verification applications is via encodings in SAT (see [11, 39]). Most notably, it is conjectured in [11] 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 in [39] to $O(N \log N)$.

Applying the compositional approach for diameter upper-bounding was pioneered by Baumgartner et al. [8], in the context of bounded model-checking. They showed that the diameter of the state space is upper-bounded by a polynomial in the recurrence diameters of projections. In that work they introduced an algorithm, the *bottom-up* algorithm, to combine the recurrence diameters of projections into an upper bound on the concrete system diameter. That algorithm used acyclicity in state-variable dependencies to elicit the necessary projections. Also, more recently, Rintanen and Gretton applied a similar algorithm to upper-bound the diameters for AI planning problems [47], but using the state space cardinalities as proxies for recurrence diameters.

In [3], we introduced the *top-down* algorithm to combine recurrence diameters into an upper bound on the concrete system diameter. We experimentally showed that this algorithm consistently computes tighter bounds than the bottom-up algorithm on standard AI planning benchmarks. We also showed that instead of using recurrence diameter, one can use the *sublist diameter*, a topological property that can be exponentially smaller than the recurrence diameter.

Lastly, in [4] we introduced two more compositional upper-bounding algorithms. The first algorithm exploits acyclicity in the state space to compute value-based abstractions we referred to as *snapshots*. It then computes the diameters of the snap-

shots and combines them into an upper bound on the concrete system diameter. We also introduced a hybrid algorithm that exploits both acyclicity in state-variable dependency and acyclicity in the state space. That algorithm computes much smaller abstractions than any of the previous compositional algorithms, and also computes exponentially tighter bounds. We showed that those bounds combined with a modern SAT-based planner [46], enabled the verification of open benchmarks (i.e. problems whose validity was never verified), most notably, large instances of the *hotel key protocol* introduced by Jackson [33, p. 185].

2.2 Formalisation Related-Work

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 logic by Constable et al. [22] and Doczkal et al. [26]. It was also formalised by Paulson [44] in Isabelle/HOL. He used hereditarily finite sets as the type of states, unlike our formalisation which uses finite maps. Furthermore, Wu et al. [56] formally prove that theorem using a formalisation of regular expressions. Results and algorithms related to reachability in automata were formalised by Sprenger [54], Schimpf et al. [50], and Esparza et al. [28], where the main goal of those authors was to obtain formally verified model checking algorithms and implementations.

3 Basic Concepts and Notations

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, which 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 denoted 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. $\{f(u_2) \mid u_2 \in \mathsf{children}_{\mathcal{G}_\alpha}(u_1)\}$.*

A DAG is represented by a list of vertices topologically sorted w.r.t. a binary relation R that encodes edges of a digraph, as follows. 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 (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.

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

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

We now define factored transitions systems, which are factored representations of digraphs 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. [2]

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 $\pi = (p, e)$, $\mathcal{D}(\pi) \equiv \mathcal{D}(p) \cup \mathcal{D}(e)$.*

We formalise factored transition systems by first defining states to be finite maps $\alpha \mapsto \beta$ from polymorphic type α to polymorphic type β. 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 $(\alpha \mapsto \beta) \times (\alpha \mapsto \beta) \to \mathit{bool}$. Note that in the formalisation 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. Consequently much of our scaffolding can be applied to theories about hybrid systems.

The next concept we define is execution, as follows.

Definition 3 (Execution)**.** *When an action π $(= (p, e))$ is executed at state x, it produces a successor state $\pi(x)$, formally defined as $\pi(x) =$ if $p \subseteq x$ then $e \uplus x$ else x. We lift execution to lists of actions $\overrightarrow{\pi}$, so $\overrightarrow{\pi}(x)$ 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.*

Action execution and action sequence execution are formalised as follows:

HOL4 Definition 3 (Execution)**.**

[2] This representation is equivalent to representations commonly used in the model checking and AI planning communities (e.g. STRIPS by Fikes and Nilsson [29] and SMV by McMillan et al. [40]). 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.

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

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

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 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.

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

Now that we have defined states, actions and execution, we are able to define a factored transition system as follows.

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 $\delta^* = \{\overrightarrow{\pi} \mid \textsf{set}(\overrightarrow{\pi}) \subseteq \delta\}$, where $\textsf{set}(l)$ 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 $V = \mathbb{U}(\delta)$, $E \equiv \{(x,\pi(x)) \mid x \in \mathbb{U}(\delta), \pi \in \delta\}$, and f is the identity function. We refer to $\mathcal{G}(\delta)$ as the state space of δ. Lastly, for states x and x', $x \rightsquigarrow x'$ denotes that there is a $\overrightarrow{\pi} \in \delta^*$ such that $\overrightarrow{\pi}(x) = x'$.

We formalise the set of valid states and valid action sequences as follows:

HOL4 Definition 4 (Factored System)**.**

$$\mathbb{U}(\delta) = \{\, x \mid \mathcal{D}(x) = \mathcal{D}(\delta)\,\}$$

$$\delta^* = \{\, \overrightarrow{\pi} \mid \textsf{set}\ \overrightarrow{\pi} \subseteq \delta\,\}$$

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, v_2 is false, and the domain of the state is the set of variables $\{v_1, v_2\}$. $(\{v_1, \overline{v_2}\}, \{v_3\})$ is an action that if executed in a state where v_1 and $\overline{v_2}$ hold, it sets v_3 to true. $\mathcal{D}((\{v_1, \overline{v_2}\}, \{v_3\})) = \{v_1, v_2, v_3\}$.

Example 1. *Consider the following factored representation, δ, and the digraph in Figure 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

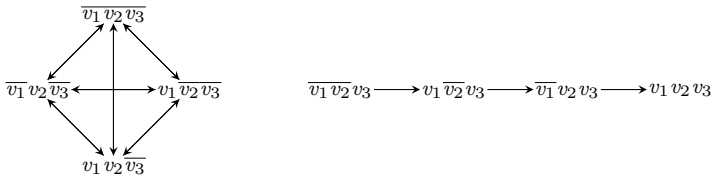


Fig. 1: The state space of δ in Example 1.

those modes represents one connected component of the digraph in Figure 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.

The topological property that we consider upper-bounding is the 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.

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

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

We note that instead of quantifying over pairs of states, we quantify over states and paths starting from those states. This avoids complexities arising from the situation when two states are not connected. 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. In HOL4, we define the diameter in the following way:

HOL4 Definition 5 (Diameter)**.**

$$d(\delta) \;=\; \mathsf{max}\;\{\mathsf{min}\;(\mathit{\Pi}^d\;\;(x,\overrightarrow{\pi},\delta))\;\mid\; x \;\in\; \mathbb{U}(\delta)\;\wedge\;\overrightarrow{\pi}\;\in\;\delta^*\}$$

where $\mathit{\Pi}^d$ is defined as

$$\mathit{\Pi}^d\;\;(x,\overrightarrow{\pi},\delta)\;=\;\{\,|\overrightarrow{\pi}'|\;\mid\;\mathsf{ex}(x,\overrightarrow{\pi}')\;=\;\mathsf{ex}(x,\overrightarrow{\pi})\;\wedge\;\overrightarrow{\pi}'\;\in\;\delta^*\}$$

In the above definition, we first define the function $\mathit{\Pi}^d$ 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 $\mathit{\Pi}^d\;\;(x,\overrightarrow{\pi},\delta)$ would be the length of the shortest action sequence equivalent to $\overrightarrow{\pi}$ executed at x. The diameter then would be the maximum length of the shortest equivalent action sequence for all pairs of states and valid action sequences.

Example 2. *Consider δ from Example 1. The diameter of δ is three since there is no action sequence that connects the states $\{\overline{v_1}, \overline{v_2}, v_3\}$ and $\{v_1, v_2, v_3\}$ having fewer than three actions.*

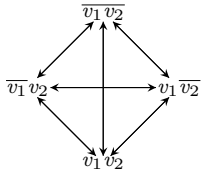


Fig. 2: The contraction of the state space equivalent to the projection of δ on $\{v_1, v_2\}$.

We note that a sufficient condition for the diameter and other topological properties that we define (i.e. the sublist and recurrence diameters) to exist, is that δ is finite and the codomain of the different states is *bool*. Those conditions appear in some of the theorems that follow. They guarantee 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.

4 Using Acyclic Dependency for Compositional Bounding

In this section we provide some background material on compositional algorithms that use abstractions induced by acyclicity in state-variable dependency. Those abstractions are obtained by *projecting* the given system on sets of state variables closed under mutual *dependency*. We first formalise the concepts of projection and state-variable dependency. We then describe the bottom-up [8, 47] and the top-down [3] compositional upper-bounding algorithms, which both exploit acyclic dependencies.

4.1 Projection and State-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 6 (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 $x\restriction_{vs}$, $\pi\restriction_{vs}$, $\overrightarrow{\pi}\restriction_{vs}$ and $\delta\restriction_{vs}$ 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.*

Example 3. *Letting $vs = \{v_1, v_2\}$, below is the projection $\delta\restriction_{vs}$, for δ from Example 1. Figure 2 shows $\mathcal{G}(\delta\restriction_{vs})$.*

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

To formalise projecting 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 6 (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

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

Please note that $f(\!|t|\!)$ is a pretty printing of HOL4's stock image function. 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 $\textsf{sat-pre}\;\; (x,\overrightarrow{\pi})$. 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. This condition is formally defined as follows.

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

An important way to capture structure in factored systems is state-variable dependency analysis, which can be used to obtain projections that are very useful in compositional reasoning, e.g. projections on sets of variables that are closed under mutual dependency. Also in many cases upper-bounding algorithms rely on the *dependency graph*, a concept defined as follows.

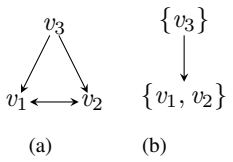


Fig. 3: (a) the dependency graph of δ from Example 1, and (b) a lifted dependency graph of δ. Actions induce edges in (a): for example, v_1 and v_2 co-occur in action effects, while v_3 only happens to be in the precondition.

Definition 7 (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:* [3] *(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 8 (Dependency Graph)**.** *This graph, sometimes called the* causal graph*, was described independently by Knoblock [35] and then Williams an Nayak [55]. $\mathcal{G}_{\mathcal{D}(\delta)}$ 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 9 (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 4. *Figure 3a and Figure 3b show the dependency graph and a lifted dependency graph of δ from Example 1.*

We formalised dependency between variables and dependency between sets of variables as follows.

HOL4 Definition 7 (Dependency)**.** *For two variables v_1 and v_2, we define dependency as*[4]

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

For sets of variables vs_1 and vs_1, we define

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

[3] Our definition is equivalent to those in [35, 55] in the context of AI planning.

[4] 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. In HOL, we formalise acyclic dependency graphs as topologically sorted lists of sets of state variables. Below is the formal definition of a lifted dependency DAG. It is an instantiation of top-sorted-abs, where the relation that induces the digraph is the dependency relation between variables.

HOL4 Definition 8 (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.

4.2 Upper-Bounding the Diameter using Recurrence Diameters

Previous attempts to employ the compositional approach to upper-bound the diameter computed an upper bound on the diameter of abstractions and then combined them into an upper bound on the diameter of the concrete system. The approaches in [8, 47] used the *recurrence diameters* of abstractions of the system at hand to upper-bound the concrete system diameter. 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 10 (Recurrence Diameter)**.** *Let* $\mathsf{distinct}(x, \vec{\pi})$ *denote that all states traversed by executing* $\vec{\pi}$ *at* x *are distinct states.*

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

It should be clear that the recurrence diameter is an upper bound on the diameter and that it can be exponentially larger than the diameter.

Example 5. *Consider* δ *from Example 1. The recurrence diameter of* δ *is three since the state space of* δ *has a simple path with four different states in it. In particular the path between states* $\{\overline{v_1}, \overline{v_2}, v_3\}$ *and* $\{v_1, v_2, v_3\}$ *has three actions.*

The recurrence diameter can be exponentially (in the number of state variables) larger than the diameter. For instance consider $\delta\restriction_{\{v_1, v_2\}}$ *from Example 3. The diameter of that projection is one, since its state space is the clique shown in Figure 2 and accordingly any state can be reached from any other state in one transition. One the other hand the recurrence diameter is three since there are multiple paths in that state space with four different states.*

Two common features of these approaches are:

(i) they introduced an algorithm that maps every dependency graph to a polynomial in the recurrence diameters of abstractions, such that the value of that polynomial is an upper bound on the diameter.

(ii) that polynomial is computed by an algorithm defined recursively "bottom-up" on acyclic dependency graphs (or equivalently, design netlists).

In this section we review the bottom-up algorithm from [8, 47] ($\mathsf{M}_{\mathsf{sum}}$), which is defined as a polynomial generator recursively defined on acyclic dependency graphs. We then review top-down algorithm from [3] ($\mathsf{N}_{\mathsf{sum}}$) that does the recursion in the opposite direction.

4.2.1 The Bottom-Up Approach

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

Definition 11 (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 12 (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 13 (Bottom-Up Acyclic Dependency Compositional Bound)**.**

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

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 [8], and the size of state space (e.g. worst case is $2^{|\mathcal{D}(\delta)|}$), used in [47]. 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 [8], 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 reformulation of Theorem 1 from [8] is as follows.

Theorem 1. *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 6. *Figure 4 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* $b(\delta\lfloor_{vs_i})$ *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) $\mathsf{M}_4 = b_4 + (1 + b_4)(\mathsf{M}_1 + \mathsf{M}_2 + \mathsf{M}_3)$
$= 2b_1 + b_2 + b_3 + b_4 + b_1b_3 + 2b_1b_4 + b_2b_4 + b_3b_4 + b_1b_3b_4$.

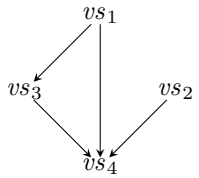


Fig. 4: A lifted dependency graph for a factored digraph that has four sets of variables closed under mutual dependency.

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 that $\mathsf{M}_{\mathsf{sum}}$ 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 bounds 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.

4.2.2 The Top-Down Approach

In this section we review the top-down algorithm. From a mechanisation perspective, an advantage of the top-down algorithm is that it is easy to formally verify as an upper-bounding algorithm for the diameter. It also avoids the redundant terms highlighted in the previous section, and was experimentally shown in [3] to be tighter than the bottom-up algorithm.

Definition 14 (Acyclic Dependency Compositional Bound)**.**

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

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

$\mathsf{N}_{\mathsf{sum}}$ 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. We note that $\mathsf{N}_{\mathsf{sum}}$ is also monotonic: for a bounding function b_1 that is an upper bound on another bounding function b_2, $\mathsf{N}_{\mathsf{sum}}\langle b_2\rangle(\delta, A_{\mathrm{VS}}) \leq \mathsf{N}_{\mathsf{sum}}\langle b_1\rangle(\delta, A_{\mathrm{VS}})$ holds, for an acyclic dependency graph A_{VS}.

Theorem 2. *For any factored representation δ with an acyclic lifted dependency graph A_{VS}, $d(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta, A_{\mathrm{VS}})$.*

We discuss the formal proof of this theorem in the next section. However, we now give an example that contrasts the bounds computed by $\mathsf{N}_{\mathsf{sum}}\langle b\rangle$ versus the ones computed using $\mathsf{M}_{\mathsf{sum}}\langle b\rangle$.

Example 7. *Again we refer to A_{VS} from Figure 4. 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.

5 Formally Verifying the Top-Down Algorithm

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. We first discuss the way we formally define the top-down algorithm. We then discuss the proof, where we prove that $\mathsf{N}_{\mathsf{sum}}\langle rd\rangle$ upper-bounds the diameter indirectly. In the proof, we first define the *sublist diameter* (denoted by ℓ), an upper bound on the diameter and a lower bound on the recurrence diameter. We then show that the sublist diameter can be compositionally bounded by $\mathsf{N}_{\mathsf{sum}}\langle\ell\rangle$ in case of an acyclic dependency graph. Theorem 2 then follows since $\mathsf{N}_{\mathsf{sum}}$ is monotonic and since the sublist diameter upper-bounds the diameter and lower bounds the recurrence diameter.

5.1 Formally Defining the Top-Down Algorithm

The first step in defining the top-down algorithm is by defining the following generic algorithm, defined on DAGs encoded as topologically sorted lists.

HOL4 Definition (Weightiest Path)**.**

```
wp R w g f u1 (u2::A) =
  if R u1 u2 then
    g (f (w u1) (wp R w g f u2 A)) (wp R w g f u1 A)
  else wp R w g f u1 A
wp R w g f u1 [] = w u1
```

This algorithm is a generalisation of an algorithm that computes the longest path in a DAG starting at a vertex u_1. It takes as an argument the relation that induces the DAG R, a weighing function w that assigns a weight to every vertex in the DAG, and two functions that combine the weights: f which combines the weights of vertices on one path to compute the path weight; and g which combines weights of different paths, as well as the vertex of interest u_1 and the DAG A.

Commonly occurring weight combination functions have the following monotonicity properties.

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

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

The previous definitions describe different notions of functions whose value is at least as large as their arguments and functions that are non-decreasing. If the weight combination functions f and g have those properties, the following monotonicity properties hold for the abstract weighted longest path function.

HOL4 Proposition 1.

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

HOL4 Proposition 2.

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

HOL4 Proposition 3.

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

One way to interpret $\mathsf{N}\langle b\rangle(vs, \delta, A_{\text{VS}})$ is as a function that computes the weights of every path that starts with vs in the acyclic dependency graph A_{VS}, and then adds all of the path weights. Every path is given a weight that depends on the weights of the vertices and the number of edges that it traverses. The weight of a vertex $vs \in A_{\text{VS}}$ is $b(\delta\lfloor_{vs})$ and the weight of a path is the product of the weights of its vertices. Thus N can be defined as the following instance of wp. The relation that induces the digraph is the dependency relation between variables. The functions for combining node weights g and f are instantiated with addition and multiplication, respectively.[5]

HOL4 Definition 14 (Acyclic Dependency Compositional Bound)**.**

$$\begin{aligned}&\mathsf{N}\langle b\rangle(vs)\ = \\ &\quad \mathsf{wp}\ (\lambda vs_1\ vs_2.\ vs_1 \rightarrow vs_2)\ (\lambda vs.\ b(\delta\lfloor_{vs}))\ (+)\ (\times)\ vs\ A_{\text{VS}}\end{aligned}$$

We note that the reason we defined wp in such a generic way is to be able to factor what is common between N and the compositional bounding function that exploits acyclicity in state space (see Section 6) and yet accommodate the difference between them. The two functions operate on DAGs induced by different relations, weigh vertices with different functions, and combine those weights in different ways.

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

5.2 The Sublist Diameter

Definition 15 (Sublist Diameter)**.** *Recall that a list l' is a sublist of l, written $l' \preceq l$, 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,*

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

The sublist diameter is formalised as follows:

HOL4 Definition 15 (Sublist Diameter)**.** *We first define the sublist relation between lists as follows:*

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

Based on that, the sublist diameter is defined as:

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

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

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

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 $\overrightarrow{\pi}$.

As a sanity check to verify that our definition of the sublist diameter has the desired semantics, we prove the following theorem. This 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.

$$\begin{array}{l} \vdash \mathsf{FINITE}\ \delta \wedge x \in \mathbb{U}(\delta) \wedge \overrightarrow{\pi} \in \delta^* \Rightarrow \\ \quad \exists \overrightarrow{\pi}'.\ \mathsf{ex}(x, \overrightarrow{\pi}) = \mathsf{ex}(x, \overrightarrow{\pi}') \wedge \overrightarrow{\pi}' \preceq_\cdot \overrightarrow{\pi} \wedge |\overrightarrow{\pi}'| \leq \ell(\delta) \end{array}$$

Also, we prove the following theorem to aid us in deriving the compositional upper bound on the sublist diameter. It states a sufficient condition for upper-bounding 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.

$$\begin{array}{l} \vdash \mathsf{FINITE}\ \delta \wedge \\ \quad (\forall \overrightarrow{\pi}\ x. \\ \qquad x \in \mathbb{U}(\delta) \wedge \overrightarrow{\pi} \in \delta^* \Rightarrow \\ \qquad\quad \exists \overrightarrow{\pi}'.\ \mathsf{ex}(x, \overrightarrow{\pi}) = \mathsf{ex}(x, \overrightarrow{\pi}') \wedge \overrightarrow{\pi}' \preceq_\cdot \overrightarrow{\pi} \wedge |\overrightarrow{\pi}'| \leq k) \Rightarrow \\ \quad \ell(\delta) \leq k \end{array}$$

An important feature of the sublist diameter is that it is a lower bound on the recurrence diameter and an upper bound on the diameter. To show that in HOL we first formalise the recurrence diameter. The recurrence diameter is formalised in a significantly different way from the diameter, since it is best specified in terms of paths of states, and because we do not need to nest max and min. The recurrence diameter is formalised as follows:

HOL4 Definition 10 (Recurrence Diameter)**.**

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

where

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

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.

Based on this formalisation of the recurrence diameter, the bounding relations between the diameter, sublist diameter and the recurrence diameter are formalised in the following theorem.

HOL4 Theorem.

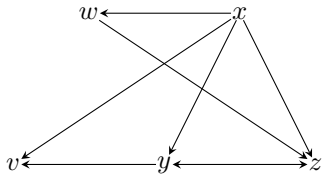
$$\vdash\ \mathsf{FINITE}\ \delta \;\Rightarrow\; d(\delta) \;\leq\; \ell(\delta) \;\wedge\; \ell(\delta) \;\leq\; rd(\delta)$$

In addition to being a lower bound on the recurrence diameter, the sublist diameter can be exponentially smaller than the recurrence diameter. This is because of a rather interesting fact: the value of sublist diameter depends on the factored representation. Two factored representations of the same state space can have different sublist diameters. This is in stark contrast to the diameter and recurrence diameter, neither of which is sensitive to the form of the factored representation. The following example highlights that difference between the subslist diameter and the other diameter functions.

Example 8. *Consider* $\delta\!\downarrow_{\{v_1,v_2\}}$ *from Example 3 and* $\delta_1 = \{k_1\!\downarrow_{\{v_1,v_2\}}, k_2\!\downarrow_{\{v_1,v_2\}}, k_3\!\downarrow_{\{v_1,v_2\}}, k_4\!\downarrow_{\{v_1,v_2\}}\}$*. Any action in* δ_1 *has an equivalent action in* $\delta\!\downarrow_{\{v_1,v_2\}}$ *in terms of execution outcome, and vice versa. Accordingly,* $\mathcal{G}(\delta_1)$ *and* $\mathcal{G}(\delta\!\downarrow_{\{v_1,v_2\}})$ *are identical, i.e.* δ_1 *and* $\delta\!\downarrow_{\{v_1,v_2\}}$ *have identical state spaces, and they have equal diameters and recurrence diameters. In particular* $d(\delta_1) = d(\delta\!\downarrow_{\{v_1,v_2\}}) = 1$*, and* $rd(\delta_1) = rd(\delta\!\downarrow_{\{v_1,v_2\}}) = 3$*.*

On the other hand, $\ell(\delta_1) = 1$*, 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* $[\pi] \preceq^{\cdot} \vec{\pi}$*. In contrast,* $\ell(\delta\!\downarrow_{\{v_1,v_2\}}) = 3$*, since no shorter sublist of the action sequence* $[p_1\!\downarrow_{\{v_1,v_2\}}; p_2\!\downarrow_{\{v_1,v_2\}}; p_3\!\downarrow_{\{v_1,v_2\}}]$ *starts at* $\{\overline{v_1}, \overline{v_2}\}$ *and results in* $\{v_1, v_2\}$*.*

Although we initially defined the sublist diameter as a tool for proving Theorem 2, the previous example shows that, instead of the recurrence diameter, the sublist diameter can be used as a base case function with the top-down approach to upper-bound the diameter. This can potentially yield exponentially tighter bounds.



(a)

Fig. 5: The dependency graph of the system in Example 9.

5.3 Validity of Top-Down Compositional Bounding with the Sublist Diameter

It is already established that the sublist diameter is an upper bound on diameter, $d(\delta) \leq \ell(\delta)$, and that the sublist diameter is a lower bound on the recurrence diameter $\ell(\delta) \leq rd(\delta)$. We noted previously that $\mathsf{N}_{\mathsf{sum}}$ is monotonic, in the sense that if b_1 upper bounds b_2 then $\mathsf{N}_{\mathsf{sum}}\langle b_2\rangle(\delta, A_{\text{VS}}) \leq \mathsf{N}_{\mathsf{sum}}\langle b_1\rangle(\delta, A_{\text{VS}})$. With those results in hand, Theorem 2 follows straightforwardly from the following lemma.

Lemma 1. *For any factored system δ with an acyclic lifted dependency DAG A_{VS}, $\ell(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle\ell\rangle(\delta, A_{\text{VS}})$.*

Our proof of Lemma 1 depends on the following argument. Consider a set of variables vs with only incoming dependencies from all the state variables in a system δ. For some $\overrightarrow{\pi} \in \delta^*$ suppose we remove actions from the projected action sequence $\overrightarrow{\pi}\downarrow_{vs}$, while preserving its execution outcome in $\delta\downarrow_{vs}$. In other words, we remove some $\delta\downarrow_{vs}$-redundant actions from $\overrightarrow{\pi}\downarrow_{vs}$. Due to the dependency structure relating to vs, that shortened projected action sequence can be "stitched" back into an execution that achives the outcome of $\overrightarrow{\pi}$ in δ. Moreover, that stiching operation produces an execution that is shorter than $\overrightarrow{\pi}$, because the $\delta\downarrow_{vs}$-redundant actions remain removed. This stitching process is formalised using a *stitching function*, $\underset{vs}{\between}$, the details of which are given below.

HOL4 Definition (Stitching Function)**.**

$$
\begin{aligned}
&(\pi' :: \overrightarrow{\pi}') \underset{vs}{\between} (\pi :: \overrightarrow{\pi}) \;=\; \\
&\quad \texttt{if}\ \textsf{varset-action}\ (\pi, vs)\ \texttt{then} \\
&\qquad \texttt{if}\ \pi' = \pi\downarrow_{vs}\ \texttt{then}\ \pi :: \overrightarrow{\pi}' \underset{vs}{\between} \overrightarrow{\pi}\ \texttt{else}\ (\pi' :: \overrightarrow{\pi}') \underset{vs}{\between} \overrightarrow{\pi} \\
&\quad \texttt{else}\ \pi :: (\pi' :: \overrightarrow{\pi}') \underset{vs}{\between} \overrightarrow{\pi} \\
&[\,] \underset{vs}{\between} \overrightarrow{\pi} \;=\; \textsf{FILTER}\ (\lambda\,\pi.\ \neg\textsf{varset-action}\ (\pi, vs))\ \overrightarrow{\pi} \\
&(\pi' :: \overrightarrow{\pi}') \underset{vs}{\between} [\,] \;=\; [\,]
\end{aligned}
$$

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 left-hand list, $\pi' :: \overrightarrow{\pi}'$, as a guide. The right-hand list of "unprimmed" actions, $\pi :: \overrightarrow{\pi}$, is filtered, with each action in the left-hand list meant to have a corresponding action in the right-hand list. The usage of the stitching function is illustrated in the following example.

Example 9. *Consider the following factored system, whose dependency graph is shown in Figure 5a.*

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

$\mathcal{D}(\delta) = c \cup p$, 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 $[\pi_2\restriction_c; \pi_3\restriction_c; \pi_5\restriction_c; \pi_6\restriction_c]$, which is in $({}^*\delta\restriction_c)$. A shorter action sequence, $\overrightarrow{\pi}_c$, achieving the same result as $\overrightarrow{\pi}\restriction_c$ is $[\pi_2\restriction_c; \pi_6\restriction_c]$. Since $\overrightarrow{\pi}_c$ is a sublist of $\overrightarrow{\pi}\restriction_c$, 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 $\overrightarrow{\pi}$.

We say that a system that exhibits the situation above, that is, having a set of variables in its domain with only incoming dependencies, has a *Parent-Child Structure* and this is formalised as follows.

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

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

The definition above says that δ exhibits a "parent-child" dependency structure, where vs is the "child", and the set $\overline{vs}$ is the "parent".[6]

The formalised lemma stating the aforementioned functionality of the stitching 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 $\mathsf{N}_{\mathsf{sum}}\langle\ell\rangle$ as a bound.

HOL4 Proposition 4.

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

[6] Note that we use HOL4's overloading capacity to hide the δ relative to whose domain the complement of vs is taken, i.e. $\mathcal{D}(\delta) \setminus vs$ is written as $\overline{vs}$.

In the previous statement, the predicate no-effectless-act $\overrightarrow{\pi}$ asserts that the action sequence has no actions with empty effects. In the proof of Lemma 1, for a lifted dependency DAG A_{VS}, we take every set of variables $p \in A_{\text{VS}}$, and remove all redundant actions whose effects are confined to p (i.e. p-actions). We then use the stitching function as our main proof tool to perform the action removal, as we describe in detail below.

To formalise the removal of p-actions, we define the following relation, which generalises the parent-child structure.

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

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

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 2. *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:

HOL4 Lemma 2.

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

where

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

Proof. The proof of Lemma 2 is constructive. Take $\overrightarrow{\pi}_{\overline{c}}$ to be a contiguous fragment of $\overrightarrow{\pi}$ that has no c-actions in it. We shall write D as shorthand for $\mathcal{D}(\delta)$ and $\overrightarrow{\pi}'_{\overline{c}}$ as shorthand for the execution $\overrightarrow{\pi}_p \underset{p}{\Join} \overrightarrow{\pi}_{\overline{c}}\lfloor_{D\setminus c}$. From HOL4 Proposition 4 it follows that $\overrightarrow{\pi}'_{\overline{c}}$ achieves the same $D \setminus c$ assignment as $\overrightarrow{\pi}_{\overline{c}}$. Precisely, $\overrightarrow{\pi}'_{\overline{c}}(x)\lfloor_{D\setminus c} = \overrightarrow{\pi}_{\overline{c}}(x)\lfloor_{D\setminus c}$. By definition $\overrightarrow{\pi}'_{\overline{c}}$ is a sublist of $\overrightarrow{\pi}_{\overline{c}}$ satisfying the length condition $\mathsf{n}(p,\overrightarrow{\pi}'_{\overline{c}}) \leq \ell(\delta\lfloor_p)$. Moreover, there is an action sequence $\overrightarrow{\pi}_p$ such that $\overrightarrow{\pi}_p(x) = \overrightarrow{\pi}_{\overline{c}}\lfloor_p(x)$ satisfying the length bound $|\overrightarrow{\pi}_p| \leq \ell(\delta\lfloor_p)$ and the sublist property $\overrightarrow{\pi}_p \preceq\cdot\ \overrightarrow{\pi}_{\overline{c}}\lfloor_p$. Recall, $\overrightarrow{\pi}_{\overline{c}}$

has no c-actions, and therefore there can be no changes to the truth assignments of c variables induced in δ when executing $\overrightarrow{\pi}_{\overline{c}}$. The preconditions on c variables in $\overrightarrow{\pi}'_{\overline{c}}$ actions are therefore all in agreement. Consequently, a concrete execution $\overrightarrow{\pi}'_{\overline{c}} \underset{D\setminus c}{\wr} \overrightarrow{\pi}_{\overline{c}}$ achieves the same result as $\overrightarrow{\pi}_{\overline{c}}$, but with at most $\ell(\delta\lfloor_p)$ p-actions. We can therefore replace $\overrightarrow{\pi}_{\overline{c}}$ by that reconstructed execution.

The above reconstruction can be used to replace each contiguous $\overrightarrow{\pi}_{\overline{c}}$ fragment in the concrete execution $\overrightarrow{\pi}$. That process terminates with an action sequence $\overrightarrow{\pi}'$ that has at most $\ell(\delta\lfloor_p)(\mathsf{n}(c, \overrightarrow{\pi})+1)$ p-actions. That statement is justified by the following HOL4 lemma, where list-frag (l_1,l_2) means that list l_2 is a contiguous sublist of l_1.

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

Because $\overrightarrow{\pi}'$ is the result of consecutive applications of the stitching function, it is a sublist of $\overrightarrow{\pi}$. The construction of $\overrightarrow{\pi}'$ can only remove p-actions, leaving the number of c-actions equal to their number in $\overrightarrow{\pi}$. □

We now describe how we use that result to prove Lemma 1, 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 $p \in A_{\text{VS}}$, the redundant actions of p are removed using Lemma 2 after removing the redundant actions of its children c. We then use the stitching function to reconcile both shortened action sequences. The formal statement of Lemma 1 follows, along with a description of how it is proved.

HOL4 Lemma 1.

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

Before we discuss the formal proof of the lemma above, we first introduce the following notation. Let $F(p, c, \overrightarrow{\pi})$ be the witness action sequence of Lemma 2. We know that:

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

Proof of Lemma 1. Firstly, for notational 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 $\mathsf{N}\langle\ell\rangle(vs, \delta, A_{\text{VS}})$.

Our proof of this lemma follows a constructive approach where we assume we have an action sequence $\overrightarrow{\pi} \in \delta^*$ and a state $x \in \mathbb{U}(\delta)$. 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 $\overrightarrow{\pi}(x) = \overrightarrow{\pi}'(x)$. We proceed by induction 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 $\overrightarrow{\pi}_{\overline{vs}} \in \delta\!\downharpoonright_{\overline{vs}}{}^*$ such that $\overrightarrow{\pi}_{\overline{vs}}(x) = \overrightarrow{\pi}\!\downharpoonright_{\overline{vs}}(x)$, $\overrightarrow{\pi}_{\overline{vs}} \preceq \overrightarrow{\pi}\!\downharpoonright_{\overline{vs}}$, and $\forall\ vs' \in l_{\mathrm{VS}}.\ \mathsf{n}(vs', \overrightarrow{\pi}') \leq \mathsf{N}\langle\ell\rangle(vs', \delta\!\downharpoonright_{\overline{vs}}, A_{\mathrm{VS}}) \leq \mathsf{N}\langle\ell\rangle(vs')$. Since $vs :: l_{\mathrm{VS}}$ is topologically sorted, $\overline{vs} \not\rightarrow vs$ holds. Letting $\overrightarrow{\pi}'_{\overline{vs}} = \overrightarrow{\pi}_{\overline{vs}} \!\downharpoonright\!\!\downharpoonright_{\overline{vs}} \overrightarrow{\pi}$, from HOL4 Proposition 4 we have $\overrightarrow{\pi}'_{\overline{vs}}(x) = \overrightarrow{\pi}(x)$. Furthermore, $\forall vs' \in l_{\mathrm{VS}}.\ \mathsf{n}(vs', \overrightarrow{\pi}'_{\overline{vs}}) \leq \mathsf{N}\langle\ell\rangle(vs')$ and $\overrightarrow{\pi}'_{\overline{vs}} \preceq^{\cdot} \overrightarrow{\pi}$. Let $C \equiv \bigcup \mathsf{children}_{A_{\mathrm{VS}}}(vs)$. The last step in this proof is to show that $F(vs, C, \overrightarrow{\pi}'_{\overline{vs}})$ is the required witness, which is justified because the generalised parent-child relation holds for δ, vs and C. Since the relations $=$, $\leq$ and $\preceq^{\cdot}$ are transitive, we have

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

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

6 Exploiting State Space Acyclicity

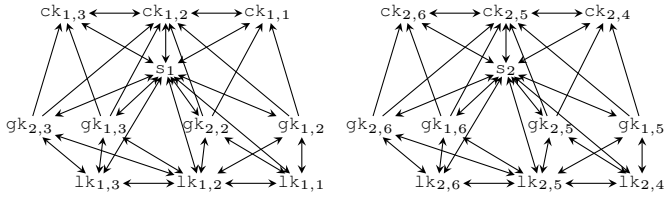
The practical utility of dependency graph based decompositions (like $\mathsf{N}_{\mathsf{sum}}$) provides a good motivation to pursue other structures, like state space acyclicity. State space acyclicity is independent of acyclicity in state-variable dependency. Thus, methods previously developed cannot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

6.1 Hotel Key Protocol

We now consider the hotel key protocol from [33]. 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

; 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)



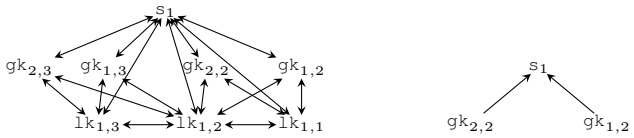
(b)

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

(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)

Fig. 6: (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.

with 1 room, 2 guests and 4 keys using a counterexample generator in [14]. The problem becomes more challenging for the safe version of the protocol, where the only feasible approach is using interactive theorem provers, as in [42].

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 6 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 10. *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 6c shows the projected system* $\delta\downarrow_{\text{ROOM1}}$.

Example 11. *Figure 6b shows a dependency graph associated with the system from Figure 6a. Let* ROOM2 $\equiv lk_2 \cup ck_2 \cup \{gk_{1,5}, gk_{1,6}, gk_{2,5}, gk_{2,6}\}$. *Figure 6b 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* $\delta\downarrow_{\text{ROOM1}}$ *and* $\delta\downarrow_{\text{ROOM2}}$ *can be used to upper-bound the diameter of the concrete system.*

6.2 State Space Acyclicity Compositional Bounding Constructs

State space acyclicity can be defined as follows.

Definition 16 (Acyclic Transition System)**.** δ *is acyclic iff* $\forall x, x' \in \mathbb{U}(\delta).\ x \neq x'$ *then* $x \not\rightsquigarrow x'$ *or* $x' \not\rightsquigarrow x$.

To formalise state space acyclicity in HOL4, we again use top-sorted-abs to represent the DAG modelling the state space. In this case, the relation that induces the digraph is the successor relation on states.

HOL4 Definition 16 (Acyclic Transition System)**.**

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

where

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

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

Example 12. *$\delta\downarrow_{\subset k_1}$ is acyclic. For example, no state satisfying $\subset K_{1,2}$ can be reached from a state satisfying $\subset K_{1,3}$. Now consider $\delta\downarrow_{\text{ROOM1}}$ from Example 10. The dependency graph of $\delta\downarrow_{\text{ROOM1}}$ is comprised of one strongly connected component (SCC). Thus, acyclicity in the assignments of $\subset k_1$ cannot be exploited in $\delta\downarrow_{\text{ROOM1}}$ 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 17 (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*

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

Example 13. *$\delta\downarrow_{\text{ROOM1}}\!\downarrow_{CK_{1,2}}$ is shown in Figure 6d.*

In HOL4 we model snapshotting as follows.

HOL4 Definition 17 (Snapshot)**.** *We first define the following relation between states*

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

Based on that, a snapshot is defined as follows:

$$\delta\!\downarrow_x = \{(p,e) \mid (p,e) \in \delta \wedge \mathsf{agree}\ p\ x \wedge \mathsf{agree}\ e\ x\}$$

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 now investigate how such acyclicity can be used for bounding. Let b be an arbitrary bounding function that satisfies $d(\delta) \leq b(\delta)$ for any δ. 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 (DAG). In that case, we have that $d(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs,\delta)$, where $\mathsf{S}_{\mathsf{max}}$ is a compositional bounding function defined as follows.

Definition 18 (Acyclic System Compositional Bound)**.** *Letting* $\mathsf{succ}(x,\delta) \equiv \{x' \mid \exists \pi \in \delta.\pi(x) = x'\}$, S *is*

$$\mathsf{S}\langle b\rangle(x, vs, \delta) = b(\delta\restriction_x) + \max_{x' \in \mathsf{succ}(x,\delta\lfloor_{vs})} (\mathsf{S}\langle b\rangle(x', vs, \delta) + 1)$$

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

S is only well-defined if $\delta\lfloor_{vs}$ is acyclic. We only seek to consider and interpret $\mathsf{S}_{\mathsf{max}}$ in systems $\delta\lfloor_{vs}$ where no execution can visit a state more than once. In that situation $\mathsf{S}_{\mathsf{max}}$ calculates the maximal cost of a traversal through the DAG formed by the state space of $\delta\lfloor_{vs}$. Completing that intuition, take $b(\delta\restriction_x)$ to be the cost of visiting a state x, and let the cost of traversing an edge between states be 1. These ideas are made concrete below, in Example 14.

Example 14. *Since* $\delta\lfloor_{ck_1}$ *is acyclic, and* $ck_{1,i} \in \mathbb{U}(\delta\lfloor_{ck_1})$*, then* $\mathsf{S}\langle d\rangle(ck_{1,i}, ck_1, \delta)$ *is well-defined, for* $i \in \{1,2,3\}$*. Denoting* $d(\delta\restriction_{ck_{1,i}})$ *with* $d_{1,i}$ *and* $\mathsf{S}\langle d\rangle(ck_{1,i}, ck_1, \delta)$ *with* $\mathsf{S}_{1,i}$*, we have* $\mathsf{S}_{1,3} = d_{1,3}$ *because* $\mathsf{succ}(ck_{1,3}, \delta\lfloor_{ck_1}) = \emptyset$*. We also have* $\mathsf{S}_{1,2} = d_{1,2}{+}1{+}S_{1,3} = d_{1,2}{+}1{+}d_{1,3}$ *and* $\mathsf{S}_{1,1} = d_{1,1}{+}1{+}S_{1,2} = d_{1,1}{+}1{+}d_{1,2}{+}1{+}d_{1,3} = d_{1,1} + d_{1,2} + d_{1,3} + 2$.

To formalise the function S we again use wp. 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 monotonicity property of wp, the relation that induces the digraph needs to be irreflexive, which is why the set of successors of a state succ 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.[7]

HOL4 Definition 18 (Acyclic System Compositional Bound)**.**

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

The following theorem states the validity of using $\mathsf{S}_{\mathsf{max}}$ to upper-bound the sublist diameter in case of the presence of acyclicity in the state space.

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

Theorem 3. *If $\delta\lfloor_{vs}$ is acyclic and b bounds ℓ, then $\ell(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.*

A formal statement of that theorem in HOL4 is as follows.

HOL4 Theorem 3.

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

The main structure of the formal proof of Theorem 3 is similar to that of Lemma 1. We first prove the following lemma.

Lemma 3. *For any δ and vs where $\delta\lfloor_{vs}$ is acyclic, $x \in \mathbb{U}(\delta)$, and $\overrightarrow{\pi} \in \delta^*$, there is $\overrightarrow{\pi}'$ such that $\overrightarrow{\pi}(x) = \overrightarrow{\pi}'(x)$, $|\overrightarrow{\pi}'| \leq \mathsf{S}\langle\ell\rangle(x\lfloor_{vs}, vs, \delta)$, and $\overrightarrow{\pi}' \preceq \overrightarrow{\pi}$.*[8]

A formal statement of that lemma is as follows.

HOL4 Lemma 3.

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

To prove that lemma we first define the function ∂ which 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 (Subsystem Trace)**.**

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

The following propositions hold for ∂.

HOL4 Proposition 5.

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

HOL4 Proposition 6.

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

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

HOL4 Proposition 7.

$$\vdash\ \partial\ (\overrightarrow{\pi}_1\ +\!\!+\ \overrightarrow{\pi}_2)\ vs\ x\ =\ \partial\ \overrightarrow{\pi}_1\ vs\ x\ +\!\!+\ \partial\ \overrightarrow{\pi}_2\ vs\ (\mathsf{ex}(x,\overrightarrow{\pi}_1))$$

Proof of Lemma 3. The proof is by induction on $\partial(x,\overrightarrow{\pi})$. The base case, $\partial(x,\overrightarrow{\pi}) = []$, is trivial. In the step case we have that $\partial(x,\overrightarrow{\pi}) = x' :: \overrightarrow{x}$ and the induction hypothesis: for any $x^* \in \mathbb{U}(\delta)$, and $\overrightarrow{\pi}^* \in \delta^*$ if $\partial(x^*,\overrightarrow{\pi}^*) = \overrightarrow{x}$ then there is $\overrightarrow{\pi}^{*\prime}$ where $\overrightarrow{\pi}^*(x^*) = \overrightarrow{\pi}^{*\prime}(x^*)$ and $|\overrightarrow{\pi}^{*\prime}| \leq \mathsf{S}\langle\ell\rangle(x^*\downarrow_{vs})$.

Since $\partial(x,\overrightarrow{\pi}) = x' :: \overrightarrow{x}$, we have $\overrightarrow{\pi}_1, \pi$ and $\overrightarrow{\pi}_2$ satisfying the conclusions of HOL4 Proposition 6. Based on HOL4 Proposition 6 and HOL4 Proposition 7 we have $\partial(x',\overrightarrow{\pi}_2) = \overrightarrow{x}$. Accordingly, letting x^*, and $\overrightarrow{\pi}^*$ from the inductive hypothesis be x', and $\overrightarrow{\pi}_2$, respectively, there is $\overrightarrow{\pi}'_2$ such that $\overrightarrow{\pi}_2(x') = \overrightarrow{\pi}'_2(x)$ and $|\overrightarrow{\pi}'_2| \leq \mathsf{S}\langle\ell\rangle(x'\downarrow_{vs})$.†

From HOL4 Proposition 6 and HOL4 Proposition 5 there is $\overrightarrow{\pi}'_1$ where $\overrightarrow{\pi}_1(x) = \overrightarrow{\pi}'_1(x)$ and $|\overrightarrow{\pi}'_1| \leq \ell(\delta\downarrow_{x\downarrow_{vs}})$. Letting $\overrightarrow{\pi}' \equiv \overrightarrow{\pi}'_1 \frown \pi :: \overrightarrow{\pi}'_2$, from HOL4 Proposition 6 and † we have $\overrightarrow{\pi}(x) = \overrightarrow{\pi}'(x)$ and $|\overrightarrow{\pi}'| \leq \ell(\delta\downarrow_{x\downarrow_{vs}}) + 1 + \mathsf{S}\langle d\rangle(x'\downarrow_{vs})$.‡

Lastly, from HOL4 Proposition 5 and HOL4 Proposition 6 we have $x\downarrow_{vs} = \overrightarrow{\pi}_1(x)\downarrow_{vs} = \overrightarrow{\pi}'_1(x)\downarrow_{vs}$ and accordingly $\pi\downarrow_{vs}(x\downarrow_{vs})$ is equal to $x'\downarrow_{vs}$. Based on that we have $x'\downarrow_{vs} \in \mathsf{succ}(x\downarrow_{vs}, \delta\downarrow_{vs})$. Then from HOL4 Proposition 2 and ‡ we have $|\overrightarrow{\pi}'| \leq \mathsf{S}\langle d\rangle(x\downarrow_{vs})$. □

Theorem 3 follows from Lemma 3 and Definitions 5 and 18.

7 Combining Acyclicity in Dependency and State Space: A Hybrid Algorithm

As we showed earlier, acyclicity in state-variable dependencies and acyclicity in the state space are independent. Accordingly an algorithm that combines the exploitation of both structures is needed. We now discuss our formal verification of the upper-bounding algorithm HYB, which combines exploitation of acyclic variable dependency with exploitation of acyclicity in state spaces. In [4] that algorithm was experimentally shown to compute much tighter bounds than $\mathsf{N}_{\mathsf{sum}}$, which already was better than the state of the art. Combined with the SAT-based AI planner Madagascar [46], it enabled the automatic verification of the safety of problems that were open (e.g. much larger instances of the hotel key protocol), and the generation of plans for problems that are otherwise out of reach for Madagascar.

Theorem 4. *Given that (i) 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, and (ii) ch is an arbitrary choice function, we have $\ell(\delta) \leq \textsc{Hyb}(\delta)$.*

Note that the sublist diameter ℓ is only computed for "base-case" problems – i.e. problems that are not further decomposed. Also, 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 ℓ is applied only to base-cases. This is because the runtime of $\mathsf{S}_{\mathsf{max}}$ can be linear in the state space of the concrete system, if its state space

Algorithm 1: $\textsc{Hyb}(\delta)$

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

is acyclic, which is unacceptable in our factored setting. This is shown in the next example.

Example 15. *For the hotel key system, $\mathcal{D}(\delta)$ has the partition $\{ck_1, ck_2, lk_1, lk_2, \{gk_{1,2}\}, \{gk_{1,3}\}, \{gk_{1,5}\}, \{gk_{1,6}\}, \{gk_{2,2}\}, \{gk_{2,3}\}, \{gk_{2,5}\}\{gk_{2,6}\}, \{s_1\}, \{s_2\}\}$. 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 $\textsc{Pur}(\delta)$ evaluates after $\prod_{vs \in \Omega(\delta)} |vs|$ calls to $\mathsf{S}_{\mathsf{max}}$.*

This computational burden is alleviated by applying $\mathsf{S}_{\mathsf{max}}$ to abstract subsystems obtained using projections that motivated Definition 14. Such abstractions can be significantly smaller than the concrete systems, thus motivating a hybrid approach that can exponentially reduce bound computation times.

Example 16. *Consider applying the approach outlined in Example 11 to compute* Pur *only on the abstractions $\delta\downarrow_{\textsc{Room1}}$ and $\delta\downarrow_{\textsc{Room2}}$. $\textsc{Pur}(\delta\downarrow_{\textsc{Room1}})$ can be evaluated in $\prod_{vs \in \Omega(\delta\downarrow_{\textsc{Room1}})} |vs|$ calls to $\mathsf{S}_{\mathsf{max}}$, where $\Omega(\delta\downarrow_{\textsc{Room1}})$=$\{ck_1, lk_1, \{gk_{1,2}\}, \{gk_{1,3}\}, \{gk_{2,2}\}, \{gk_{2,3}\}\}$. The same observation can be made for the evaluation time of $\textsc{Pur}(\delta\downarrow_{\textsc{Room2}})$. Thus the product expression in Example 15 is split into a sum if* Pur *is called on projections.*

Also note that the dependency graph $\mathcal{G}_{\mathcal{D}(\delta)}$ 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 17.

Example 17. *As shown in Figure 6b, the dependency graph of $\delta\downarrow_{\textsc{Room1}}$ has a single SCC, and thus not susceptible to dependency analysis. Taking a snapshot of $\delta\downarrow_{\textsc{Room1}}$ at the assignment $CK_{1,2}$ yields a system with one SCC in its dependency graph as well, as shown in Figure 6e. However, taking the snapshot of $\delta\downarrow_{\textsc{Room1}}\!\!\blacklozenge_{CK_{1,2}}$ 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 6f.*

7.1 Formal Verification of the Hybrid Algorithm

The hybrid algorithm Hyb is characterised in HOL as follows:

HOL4 Lemma (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))
    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⟨HYB⟩(x′) | x′ ∈ 𝕌(δ⌊vs)})
    else ℓ(δ)
```

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.

To guide HOL4 to extract the right termination conditions we need to add redundant if-then-else statements to the definition of HYB (not the characterisation), which is a cumbersome process. Those if-then-else statements specify explicitly what the termination conditions are, i.e. they specify constraints on the behaviour of the functional parameters passed to N and S, as shown in the following definition.

HOL4 Definition (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
               (λ δ'.
                  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
```

Lastly, proving the validity of the hybrid algorithm as an upper bound on the sublist diameter, and accordingly the diameter, follows directly from HOL4 Lemma 1 and HOL4 Theorem 3.

HOL4 Theorem 4.

```
⊢ FINITE δ ∧
  (∀ δ'.
     dep-DAG δ' (f₁ δ') ∧
     ∀ vs x⃗. (vs, x⃗) ∈ f₂ δ' ⇒ sspace-DAG δ'⌊vs x⃗) ⇒
   ℓ(δ) ≤ HYB f₁ f₂ δ
```

An important gain we had from the formalisation process is that it guided us to the explicit termination conditions for HYB.

8 Concluding Remarks

With this work we publish the details of the formal verification work behind a fruitful collaboration between the disciplines of AI planning and mechanised mathematics. In concluding, it is worth revisiting key results and observations made during our formalisation efforts, and the motivation for mechanisation. From the point of view of the interactive theorem-proving community, it is gratifying to be able to find and fix errors in the modern research literature. The insights which led us to develop the sublist diameter, and the top-down algorithm, followed preliminary attempts to formalise results by Rintanen and Gretton in [47]. Those efforts, reported in [3], uncovered an error in their theoretical claims, where they incorrectly state that the diameter can be compositionally bounded using abstractions induced by acyclicity in the dependency graph. Importantly, that error never shows up during experimentation, where compositional bounding using the diameter yields admissible results on all of

thousands of diverse problems from planning benchmarks. Nonetheless such an error cannot be tolerated in safety critical applications. This kind of elusive error makes a strong case for the utility of mechanical verification, which identifies and helps eliminate mistakes before they migrate into production systems.

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 [47] to do its plan-search, that system could incorrectly conclude that no plan exists.

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. The general organisation of the library is shown in Figure 7 and the sizes and descriptions of important theories is in Table 1. 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 which we found cumbersome and to which more automation could be helpful include: (i) searching for theorems in the library, (ii) the generation of termination conditions, and (iii) deriving the function form of relations for which uniqueness properties exist. Another more general issue that we faced is the absence of a mechanism akin to type classes in Isabelle/HOL, which could have allowed us to reduce the repetition of theorem hypotheses.

Future work can leverage our work on propositionally factored systems in more general settings, such as for systems in which the codomains of states are not necessarily Boolean, finite or even countable. This raises the possibility of applying and extending our algorithms and related proof scripts to hybrid systems. In particular, we developed a large library describing factored transition systems. Much of the theory in our library applies to factored systems that are not *propositionally* factored, and therefore that theory can be used for verifying algorithms on hybrid systems. Here, 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.

In terms of formally verifying AI planning algorithms, we have only scratched the surface. 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 [46]. 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 [46, 48]. 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.

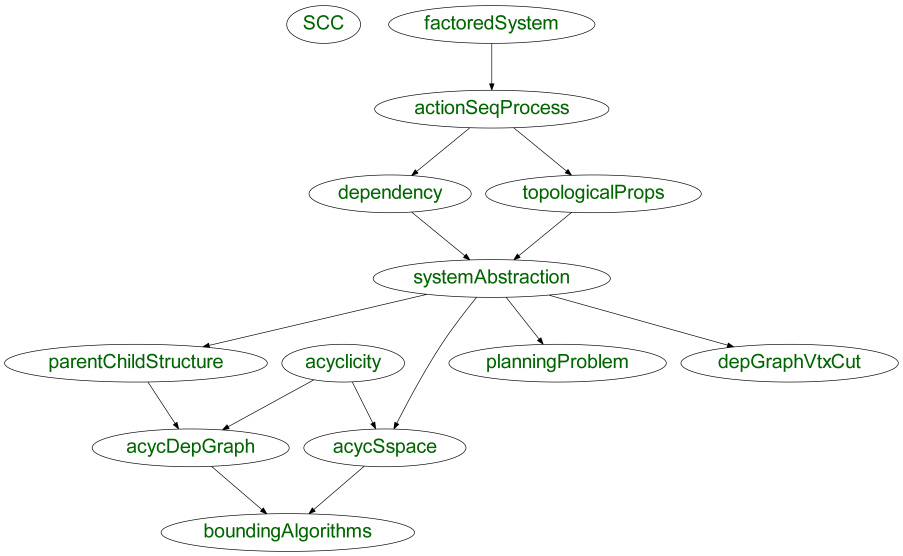


Fig. 7: The organisation of the different theories concerning factored transition systems. An edge from one theory to another indicates the dependence of the latter on the former.

HOL4 Notation and Availability All statements appearing with a turnstile ($\vdash$) are HOL4 theorems, automatically pretty-printed to LaTeX. All our HOL scripts, experimental code and data are available from: `https://bitbucket.org/MohammadAbdulaziz/planning.git`.

Acknowledgements We thank Daniel Jackson for suggesting applying diameter upper-bounding on the hotel key protocol verification. We also thank Dr. Alban Grastien and Dr. Patrik Haslum for their the very helpful discussions and insightful feedback which they gave us through the entire project. Lastly, we thank the anonymous reviewers for their detailed and helpful reviews.

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	1336	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.
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 1: A table showing the sizes of different theories and their content.

References

1. Abboud, A., Williams, V.V., Wang, J.: 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, pp. 377–391. SIAM (2016)
2. Abdulaziz, M., Gretton, C., Norrish, M.: Mechanising Theoretical Upper Bounds in Planning. In: Workshop on Knowledge Engineering for Planning and Scheduling (2014)
3. Abdulaziz, M., Gretton, C., Norrish, M.: Verified Over-Approximation of the Diameter of Propositionally Factored Transition Systems. In: Interactive Theorem Proving, pp. 1–16. Springer (2015)
4. Abdulaziz, M., Gretton, C., Norrish, M.: A State Space Acyclicity Property for Exponentially Tighter Plan Length Bounds. In: International Conference on Automated Planning and Scheduling (ICAPS). AAAI (2017)
5. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM Journal on Computing **28**(4), 1167–1181 (1999)
6. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. Journal of Computer and System Sciences **54**(2), 255–262 (1997)
7. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM (JACM) **42**(4), 844–856 (1995)
8. Baumgartner, J., Kuehlmann, A., Abraham, J.: Property checking via structural analysis. In: Computer Aided Verification, pp. 151–165. Springer (2002)
9. Berezin, S., Campos, S., Clarke, E.M.: Compositional reasoning in model checking. In: Compositionality: The Significant Difference, pp. 81–102. Springer (1998)
10. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)
11. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS, pp. 193–207 (1999)
12. Björklund, A., Husfeldt, T.: Finding a path of superlogarithmic length. SIAM Journal on Computing **32**(6), 1395–1402 (2003)
13. Björklund, A., Husfeldt, T., Khanna, S.: Approximating longest directed paths and cycles. In: International Colloquium on Automata, Languages, and Programming, pp. 222–233. Springer (2004)
14. Blanchette, J.C., Nipkow, T.: Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In: Interactive Theorem Proving, First International Conference, ITP 2010, pp. 131–146 (2010). DOI 10.1007/978-3-642-14052-5_11
15. Bundala, D., Ouaknine, J., Worrell, J.: 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, pp. 155–164. IEEE Computer Society (2012)
16. Case, M.L., Mony, H., Baumgartner, J., Kanzelman, R.: Enhanced verification by temporal decomposition. In: FMCAD 2009, 15-18 November 2009, Austin,

Texas, USA, pp. 17–24 (2009)
17. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. SIAM Journal on Computing **39**(5), 2075–2089 (2010)
18. Chechik, S., Larkin, D.H., Roditty, L., Schoenebeck, G., Tarjan, R.E., Williams, V.V.: Better approximation algorithms for the graph diameter. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1041–1052. Society for Industrial and Applied Mathematics (2014)
19. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Progress on the state explosion problem in model checking. In: Informatics, pp. 176–194. Springer (2001)
20. Clarke, E.M., Emerson, E.A., Sifakis, J.: Turing lecture: model checking–algorithmic verification and debugging. Communications of the ACM **52**(11), 74–84 (2009)
21. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM transactions on Programming Languages and Systems (TOPLAS) **16**(5), 1512–1542 (1994)
22. Constable, R.L., Jackson, P.B., Naumov, P., Uribe, J.C.: Constructively formalizing automata theory. In: Proof, language, and interaction, pp. 213–238 (2000)
23. Dankelmann, P.: The diameter of directed graphs. Journal of Combinatorial Theory, Series B **94**(1), 183–186 (2005)
24. Dankelmann, P., Dorfling, M.: Diameter and maximum degree in eulerian digraphs. Discrete Mathematics **339**(4), 1355–1361 (2016)
25. Dankelmann, P., Volkmann, L.: The diameter of almost eulerian digraphs. the electronic journal of combinatorics **17**(1), R157 (2010)
26. Doczkal, C., Kaiser, J.O., Smolka, G.: A constructive theory of regular languages in coq. In: International Conference on Certified Programs and Proofs, pp. 82–97. Springer (2013)
27. Erdős, P., Pach, J., Pollack, R., Tuza, Z.: Radius, diameter, and minimum degree. Journal of Combinatorial Theory, Series B **47**(1), 73–79 (1989)
28. Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.G.: A fully verified executable LTL model checker. In: International Conference on Computer Aided Verification, pp. 463–478. Springer (2013)
29. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence **2**(3-4), 189–208 (1971)
30. Filiot, E., Jin, N., Raskin, J.F.: Antichains and compositional algorithms for LTL synthesis. Formal Methods in System Design **39**(3), 261–296 (2011)
31. Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM Journal on Computing **5**(1), 83–89 (1976)
32. Helmert, M.: The Fast Downward planning system. Journal of Artificial Intelligence Research **26**, 191–246 (2006)
33. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
34. Kautz, H.A., Selman, B.: Planning as satisfiability. In: ECAI, pp. 359–363 (1992)
35. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial Intelligence **68**(2), 243–302 (1994)
36. Knyazev, A.: Diameters of pseudosymmetric graphs. Mathematical Notes **41**(6), 473–482 (1987)

37. Kroening, D.: Computing over-approximations with bounded model checking. Electronic Notes in Theoretical Computer Science **144**(1), 79–92 (2006)
38. Kroening, D., Ouaknine, J., Strichman, O., Wahl, T., Worrell, J.: Linear completeness thresholds for bounded model checking. In: Computer Aided Verification, pp. 557–572. Springer (2011)
39. Kroening, D., Strichman, O.: Efficient computation of recurrence diameters. In: VMCAI, pp. 298–309 (2003)
40. McMillan, K.L.: Symbolic model checking. In: Symbolic Model Checking, pp. 25–60. Springer (1993)
41. Moon, J.W., et al.: On the diameter of a graph. The Michigan Mathematical Journal **12**(3), 349–351 (1965)
42. Nipkow, T.: Verifying a hotel key card system. In: K. Barkaoui, A. Cavalcanti, A. Cerone (eds.) Theoretical Aspects of Computing (ICTAC 2006), *Lecture Notes in Computer Science*, vol. 4281. Springer (2006). Invited paper.
43. Pardalos, P.M., Migdalas, A.: A note on the complexity of longest path problems related to graph coloring. Applied mathematics letters **17**(1), 13–15 (2004)
44. Paulson, L.C.: A formalisation of finite automata using hereditarily finite sets. In: International Conference on Automated Deduction, pp. 231–245. Springer (2015)
45. Pnueli, A., Rodeh, Y., Strichman, O., Siegel, M.: The small model property: How small can it be? Information and computation **178**(1), 279–293 (2002)
46. Rintanen, J.: Planning as satisfiability: Heuristics. Artificial Intelligence **193**, 45–86 (2012)
47. Rintanen, J., Gretton, C.O.: Computing upper bounds on lengths of transition sequences. In: International Joint Conference on Artificial Intelligence (2013)
48. Robinson, N., Gretton, C., Pham, D.N., Sattar, A.: SAT-based parallel planning using a split representation of actions. In: ICAPS (2009)
49. Roditty, L., Vassilevska Williams, V.: Fast approximation algorithms for the diameter and radius of sparse graphs. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pp. 515–524. ACM (2013)
50. Schimpf, A., Merz, S., Smaus, J.G.: Construction of Büchi automata for LTL model checking verified in Isabelle/HOL. In: International Conference on Theorem Proving in Higher Order Logics, pp. 424–439. Springer (2009)
51. Sheeran, M., Singh, S., Stålmarck, G.: 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, pp. 108–125 (2000). DOI 10.1007/3-540-40922-X_8
52. Slind, K., Norrish, M.: A brief overview of HOL4. In: Theorem Proving in Higher Order Logics, *LNCS*, vol. 5170, pp. 28–32. Springer (2008)
53. Soares, J.: Maximum diameter of regular digraphs. Journal of Graph Theory **16**(5), 437–450 (1992)
54. Sprenger, C.: 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, pp. 167–183. Springer (1998)
55. Williams, B.C., Nayak, P.P.: A reactive planner for a model-based executive. In: International Joint Conference on Artificial Intelligence, pp. 1178–1185. Morgan

Kaufmann Publishers (1997)
56. Wu, C., Zhang, X., Urban, C.: A formalisation of the Myhill-Nerode theorem based on regular expressions (proof pearl). In: International Conference on Interactive Theorem Proving, pp. 341–356. Springer (2011)
57. Yuster, R.: Computing the diameter polynomially faster than APSP. arXiv preprint arXiv:1011.6181 (2010)