Complete Local Reasoning About Parameterized Programs Over Topologies

Ruotong Cheng and Azadeh Farzan(✉)

University of Toronto, Toronto, Canada
azadeh@cs.toronto.edu

Abstract. This paper investigates the algorithmic safety verification problem of infinite-state parameterized concurrent programs over a rich set of communication topologies. The goal is to automatically produce a proof of correctness in the form of a *universally quantified* inductive invariant, where the quantification is over the nodes in the topology. We illustrate that under reasonable assumptions on the underlying topology, the problem can be reduced to and solved as a *compositional* scheme, that is, the verification of the parameterized family is reduced to a set of *local* proofs, in a *complete* manner. We propose a verification algorithm and demonstrate through a set of benchmarks over several different topologies that our approach is effective in proving parameterized programs safe.

1 Introduction

Verifying the safety of parameterized programs is a fundamental and well-studied problem in computer-aided verification. This paper focuses on an instance of this problem that is specified by two distinguishing features: (1) The parameterized programs are *infinite-state* and defined over *a rich class of underlying communication topologies*. (2) Safety is formulated as the reachability of an error state that relates a constant number of threads.

In verification of concurrent programs, compositional verification, often called *thread-modular reasoning*, aims to construct a proof of correctness for the entire program by composing correctness proofs for individual threads. This is, however, known to be *incomplete* for reasoning about parameterized programs. In [17], the concept of *thread modularity at many levels* is proposed as an alternative. It is proved to be *complete* w.r.t. a class of proofs in the form of universally quantified invariants, called *Ashcroft Invariants*. Rather than focusing on one thread at a time, the *local* proof can be the proof of the parallel product of k threads, for a fixed k, when an Ashcroft invariant with $k-1$ quantifiers proves the program correct. This scheme, however, is deeply tied to the assumption that the program comprises replicated threads communicating through a *bounded number* of shared global variables. What is the nature of compositional verification of parameterized programs if they are over an arbitrary communication topology?

We propose an answer to this question by using the inherent *symmetry* in such parameterized programs. Let us use an example to illustrate the core ideas.

E. Darulova et al. (Eds.): CAV 2026, LNCS 16682, pp. 67–89, 2026.
https://doi.org/10.1007/978-3-032-32519-8_4

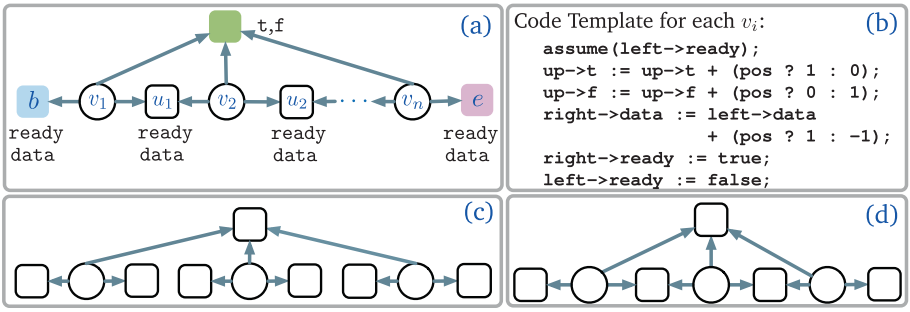


Fig. 1. Pipeline Example (a,b). Small pipeline subprograms (c,d).

Consider the parameterized program given in Fig. 1(a, b), which is a family of pipelines with n processes (depicted as circle nodes) arranged on a line in linear order, where the first and last nodes are *distinguished* from the rest. Each pair of neighbouring processes shares two variables, an integer-typed `data` and a Boolean-typed `ready`, which reside on the square nodes in between. `ready` being true indicates that `data` has been computed and is ready to be read; all instances of `ready` are initially `false`. Additionally, the green (top) square node hosts two *global* counters `t` and `f`. Each process has an uninitialized Boolean-typed variable `pos`, which serves as a nondeterministic Boolean choice. Observe that there are *unboundedly many shared variables and local variables*, which is an important point of departure from classic models such as that of [13,17].

Assume that each circle node atomically executes the given code template, where `left` and `right` refer to the corresponding square node neighbour of the process. The idea is that, depending on the nondeterministic value of `pos`, the process increments or decrements the value before passing it along. Separately, the counts of true and false values for local `pos` variables are recorded in global counters `t` and `f`. We want to prove the property below, which can be proved using a universally quantified inductive invariant:

$$e{\triangleright}\texttt{ready} \implies e{\triangleright}\texttt{data} = b{\triangleright}\texttt{data} + \texttt{t} - \texttt{f}.$$

How can we discover this proof compositionally? The first key observation is that the smaller proofs do correspond to small subprograms (of programs in the parameterized family), but they do not necessarily have to be a single (or several) threads. For instance, Fig. 1(c, d) illustrates two small subprograms with three processes each. In each case, the program includes processes *and* the *neighbourhoods* accessed by them. The distinction between the two programs is that in (d), the processes are strictly neighbours, but not in (c). The three-threaded programs in (c, d), however, *do not* belong in the pipeline parameterized family. It is clear why (c) is not a pipeline, and (d) does not have distinguished beginning and end nodes, and hence is not in the family.

The thesis of this paper is that the proof for the pipeline parameterized family can be composed from the proofs of such subprograms. Intuitively, to construct

the proof for any arbitrarily-sized pipeline, we break it into smaller pieces. If we have proofs for sufficiently many pieces that are distinct up-to-isomorphism, then we can use these proofs to give a proof for the entire pipeline program. This is fundamentally based on a *symmetry* argument formalized through an appropriate notion of local isomorphism.

In Sect. 4, we prove a core result that if the parameterized family includes *sufficiently many such small programs*, then a generalized Ashcroft invariant proves the entire family correct *if and only if* it proves the small programs correct. Since programs in the family are instantiated from topologies in the family, this effectively puts a condition on the family of topologies. This has two orthogonal implications: (1) A candidate Ashcroft invariant can be verified using a bounded number of checks (Theorem 2), and (2) rather than searching for an Ashcroft invariant for the entire family, one can look for one for this bounded set. The former is a *cut-off* style theorem for verification. The latter is a key observation behind our compositional verification algorithm. This is inspired by the proof of relative completeness from [9], where we linked a (different) proof system for parameterized programs to a generalization of Ashcroft invariants. We use these invariants, which accommodate arbitrary topologies, use a core observation from the proof, and extend it to prove our *cut-off* theorem.

The generalized Ashcroft invariants are not exactly *local* proofs, even for the small programs. The proof of the cut-off theorem relies on the insight that, without loss of generality, isomorphic neighbourhoods can share proofs, even if they are in different parts of the topology. In Sect. 5, we rely on this insight again to argue that each Ashcroft invariant can be syntactically rearranged (i.e., *normalized*) so that proofs of isomorphic neighbourhoods are *syntactically grouped*. We refer to this transformation as *normalization*, and it streamlines the ingredients (function and predicate symbols) used in the matrix of the formula so that (1) there are distinct parts about the topology and the data manipulated by the program, and (2) using symmetry, one can enforce a bound on the size of terms in the matrix (Lemma 1). This results in a *bounded search space* for the ingredients of Ashcroft invariants, which can now be discovered through purely *local* proof searches for small programs. In other words, *normalization* produces a *complete template* for proof search. This search is then performed through the collection of *local* verification conditions as a system of constrained Horn clauses (CHC). As a consequence of (1), these local proofs become entirely oblivious to the program topology, which eliminates the need for axiomatizing the topology.

One can produce a proof for the entire parameterized family by using the procedure outlined above to collect constraints from all sufficiently many small programs of our cut-off theorem. There are, however, two obstacles to making this a pragmatic solution. First, we already mentioned that our pipeline example does not include the required small subprograms. In general, a typical user-friendly topology family (e.g. rings, lines, and grids) does not include these small subprograms. In [9], we argue that one can *close* these topologies by adding the missing small programs while preserving the safety/unsafety status of the entire family. Second, even if one liberally closes the family, there is an enormous amount of

redundancy in constraints collected from all necessary small programs. In Sect. 6, we give a construction that takes a standard user-friendly topology family, like the pipelines of Fig. 1, and populates it with enough small programs to guarantee the required sufficiency conditions for compositional reasoning, and produces a CHC system with less redundancy from them. We take our treatment of redundancy further by introducing a set of *complete* optimizations in Sect. 7. These optimizations take advantage of the fact that our methodology of extracting a *complete template* for proof search from normalized Ashcroft invariants brings a degree of flexibility to this search. The normal form can change, and the set of templates for the search changes with it, but the guarantee of completeness remains.

We have implemented our methodology in a tool called Mosaic, which takes as input a family of parameterized programs specified as a family of topologies and an assignment of nodes to a bounded number of code templates. Our experimental results show that the method is very effective in proving interesting properties of programs over a broad set of topologies. Furthermore, in Sect. 8, as a theoretical case study, we derive the algorithm of [17] as a special case of our framework to demonstrate its robustness.

2 Background

Structures and Maps Between Them. We are concerned with structures in the context of first-order logic with equality. A *vocabulary* $\mathcal{V}$ is a set of predicate symbols and function symbols, each with a certain *arity* given by a global map ar. A $\mathcal{V}$*-structure* $\mathbb{A}$ is defined as an *underlying set* with an *interpretation* of the symbols in $\mathcal{V}$. We generally use $\mathbb{A}$ to also refer to the underlying set of $\mathbb{A}$.

Let $\mathbb{A}, \mathbb{B}$ be two $\mathcal{V}$-structures. An *embedding* $f : \mathbb{A} \hookrightarrow \mathbb{B}$ is an injection that preserves and reflects the interpretation of all predicate and function symbols. An *isomorphism* is a surjective embedding. An *automorphism* is an isomorphism from a structure to itself. If there is an embedding from $\mathbb{A}$ into $\mathbb{B}$, we say $\mathbb{A}$ is an *embedded substructure* of $\mathbb{B}$; if furthermore, the underlying set of $\mathbb{A}$ is a subset of that of $\mathbb{B}$ and the inclusion map is an embedding, we say $\mathbb{A}$ is a *substructure* of $\mathbb{B}$. The *substructure generated* by a subset A of $\mathbb{A}$ is the smallest substructure containing A that is closed under applying functions from $\mathcal{V}$.

Constrained Horn Clauses. For brevity, we assume the background theory (for program data) is given by a $\mathcal{V}_{\mathsf{Data}}$-structure $\mathbb{D}$. Let $\mathcal{R}$ be a set of predicate symbols representing unknowns, disjoint from $\mathcal{V}_{\mathsf{Data}}$. A *constrained Horn clause* (CHC) is a formula in the vocabulary $\mathcal{V}_{\mathsf{Data}} \cup \mathcal{R}$ that is of the form $B_1 \wedge \cdots \wedge B_n \wedge C \Rightarrow H$, where every B_i is an application $p(x_1, \ldots, x_{\mathsf{ar}(p)})$ of an unknown predicate symbol $p \in \mathcal{R}$ to free variables $x_1, \ldots, x_{\mathsf{ar}(p)}$, C is a $\mathcal{V}_{\mathsf{Data}}$-formula, and the *head* H is either $\bot$ or an application as in B_i.

A *CHC system* $\mathcal{C}$ is a finite set of CHCs. It is *satisfiable* if there exists an interpretation for each unknown predicate symbol that makes the universal closure of each clause in $\mathcal{C}$ valid modulo $\mathbb{D}$. A (syntactic) *solution* to $\mathcal{C}$ maps

every unknown p in $\mathcal{C}$ to a $\mathcal{V}_{\mathsf{Data}}$-formula with $\mathsf{ar}(p)$ free variables such that the subsets defined by these formulas constitute a satisfying interpretation.

3 Parameterized Programs and Their Proofs

In this section, we formalize concurrent programs over individual topologies and then parameterized programs over parameterized families of topologies. We formalize a class of safety proofs for such parameterized programs in the form of universally quantified inductive invariants.

3.1 Concurrent Program Over Topology

In our setup, a *topology* is a finite logical structure $\mathbb{T}$ over a vocabulary $\mathcal{V}_{\mathsf{Node}}$, whose underlying elements are called *nodes*. For a set of nodes V, we define the *neighbourhood* $N(V)$ as the substructure generated by the nodes in V. For example, Fig. 1(d) illustrates the neighbourhood of two consecutive nodes in a pipeline, and Fig. 1(c) that of two non-neighbouring nodes. We sometimes identify a tuple of nodes $\boldsymbol{v} \in \mathbb{T}^k$ with the set $\{v_1, \ldots, v_k\}$ and write $N(\boldsymbol{v})$.

We assume there is a variable on every node, which can be a structure consisting of multiple variables, standing in for variables local to a process or shared between neighbouring processes. Intuitively, a process that runs on a node only has access to the variables in its neighbourhood. For brevity, we fix a single data domain $\mathbb{D}$ over a vocabulary $\mathcal{V}_{\mathsf{Data}}$ and assume all variables have domain $\mathbb{D}$.

States and Transitions. For any node v, we say a valuation $N(v) \to \mathbb{D}$ is a *local state* of v. A valuation $\mathbb{T} \to \mathbb{D}$ is a *global state*. A *concurrent program* $\mathbf{P}$ *over a topology* $\mathbb{T}$ can be viewed as a transition system that captures the changes in the local states, which together comprise changes in the global state.

Formally, $\mathbf{P}$ is a pair $\langle \mathbb{T}, [\![-]\!] \rangle$, where $[\![-]\!]$ maps every node a to a binary relation on its local states, which induces a transition relation over the global states by declaring that variables outside the neighbourhood are unaffected: for any global states μ, μ', we have $\mu \xrightarrow{v} \mu'$ if $(\mu|_{N(v)}, \mu'|_{N(v)}) \in [\![v]\!]$ and $\mu|_{\overline{N(v)}} = \mu'|_{\overline{N(v)}}$. We say $\mu \to \mu'$ if $\mu \xrightarrow{v} \mu'$ for some $v \in \mathcal{V}_{\mathsf{Node}}$.

We consider the vocabulary $\mathcal{V}_{\mathsf{Node}}$ (resp. $\mathcal{V}_{\mathsf{Data}}$) to have a single sort Node (resp. Data). Let $\mathfrak{p}$ and $\mathfrak{p}'$ be two formal symbols of type $\mathsf{Node} \to \mathsf{Data}$. We define a set $\textsc{Transition}[\mathbb{T}]$ consisting of formulas of the form $\phi[\mathfrak{p}(\boldsymbol{u}), \mathfrak{p}'(\boldsymbol{w})]$ (the notation $\mathfrak{p}(\boldsymbol{u})$ abbreviates the list $\mathfrak{p}(u_1), \ldots, \mathfrak{p}(u_{|\boldsymbol{u}|})$, and similarly for $\mathfrak{p}'(\boldsymbol{w})$), where ϕ is a $\mathcal{V}_{\mathsf{Data}}$-formula and $\boldsymbol{u}, \boldsymbol{w}$ are tuples over $\mathbb{T}$. We assume every $[\![v]\!]$ is definable by such a formula where the nodes are from $N(v)$.

Safety Specification. We assume that the *safety* of a concurrent program is specified through a pair of functions $\langle \mathsf{init}, \mathsf{error} \rangle$, where init and error map each node $v \in \mathbb{T}$ to a set of local states of v.[1] For a global state μ, we say μ satisfies

[1] To simplify presentation, we assume error is a unary function in our formalism. It is straightforward to account for an m-ary error function, where m is no more than the width of the Ashcroft invariant to search for (see Sect. 3.3).

init if $\mu|_{N(v)} \in \mathsf{init}(v)$ for all $v \in \mathbb{T}$, and μ satisfies error if $\mu|_{N(v)} \in \mathsf{error}(v)$ for some $v \in \mathbb{T}$. A program $\mathbf{P}$ is safe w.r.t. $\langle \mathsf{init}, \mathsf{error} \rangle$ if for any global states μ, μ' such that $\mu \to \mu'$, if μ satisfies init, then μ' does not satisfy error.

The class of formulas Assertion$[\mathbb{T}]$ is defined similarly to Transition$[\mathbb{T}]$, except that the symbol $\mathfrak{u}'$ does not appear. We assume that every $\mathsf{init}(v)$ and $\mathsf{error}(v)$ is definable by such a formula where the nodes are from $N(v)$.

We abuse notations slightly and also use $\mathsf{init}(v)$, $\mathsf{error}(v)$, and $[\![v]\!]$ to refer to the formula that defines each.

3.2 Parameterized Program Over a Family of Topologies

Parameterized programs are generally defined over a notion of *symmetry*. That is, two programs are considered to be part of the same family when they are in some sense symmetric. In other words, we want symmetric nodes to have symmetric semantics. To formalize this, we use a *local* notion of symmetry defined based on the concept of a *local isomorphism*, inspired by [9].

Symmetry Equivalence. For structures $\mathbb{T}$ and $\mathbb{T}'$ over the same vocabulary, given tuples $\boldsymbol{u} \in \mathbb{T}^d$ and $\boldsymbol{v} \in \mathbb{T}^{d'}$, we say any isomorphism $\beta : N(\boldsymbol{u}) \to N(\boldsymbol{v})$ is a *local isomorphism* (from $\boldsymbol{u}$ to $\boldsymbol{v}$) if it maps $\boldsymbol{u}$ to $\boldsymbol{v}$ (component-wise). *Local isomorphisms* induce an equivalence relation $\simeq$, called *local symmetry*, on tuples over $\mathbb{T}$ and $\mathbb{T}'$: we have $\boldsymbol{u} \simeq \boldsymbol{v}$ if there is a local isomorphism from $\boldsymbol{u}$ to $\boldsymbol{v}$. For example, in the pipeline topology of Fig. 1(a), any two circle nodes are equivalent up to symmetry, as long as they are not the distinguished begin/end circle node; those two nodes are in two other symmetry equivalence classes.

The local isomorphism $\beta : N(\boldsymbol{u}) \to N(\boldsymbol{v})$ induces a map $\beta^* : (N(\boldsymbol{v}) \to \mathbb{D}) \to (N(\boldsymbol{u}) \to \mathbb{D})$ between local states by $\mu \mapsto \mu \circ \beta$. We lift β^* to (sets of) tuples of valuations in the natural way. In other words, symmetric nodes have symmetric semantics, which means that proof arguments from one generalize to the other up to symmetry.

Parameterized Programs. Given a vocabulary $\mathcal{V}_{\mathsf{Node}}$ and a data domain $\mathbb{D}$, a *parameterized (concurrent) program* is an infinite family $\mathcal{P}$ of programs over $\mathcal{V}_{\mathsf{Node}}$-topologies such that for every pair of programs $\langle \mathbb{T}, [\![-]\!] \rangle$ and $\langle \mathbb{T}', [\![-]\!]' \rangle$ in the family, for any $v \in \mathbb{T}$ and $v' \in \mathbb{T}'$ such that $v \simeq v'$ via β (namely $v' = \beta(v)$), we have $[\![v]\!] = \beta^*[\![v']\!]$. We write $\mathcal{P}$ as an indexed set $\{\mathbf{P}_i\}_{i \in \mathcal{I}}$, where $\mathbf{P}_i = \langle \mathbb{T}_i, [\![-]\!]_i \rangle$.

Safety Specification. The safety of a parameterized program is expressed by a *parameterized specification* $\{\langle \mathsf{init}_i, \mathsf{error}_i \rangle\}_{i \in \mathcal{I}}$, where the pair $\langle \mathsf{init}_i, \mathsf{error}_i \rangle$ is a safety specification for $\mathbf{P}_i$. Naturally, we expect the specification to also be symmetric: Formally, for any $i, j \in \mathcal{I}$, $u \in \mathbb{T}_i$, and $v \in \mathbb{T}_j$ such that $u \simeq v$ via β, we have $\mathsf{init}_i(u) = \beta^* \mathsf{init}_j(v)$ and $\mathsf{error}_i(u) = \beta^* \mathsf{error}_j(v)$. We say $\mathcal{P}$ is safe if $\mathbf{P}_i$ is safe w.r.t. $\langle \mathsf{init}_i, \mathsf{error}_i \rangle$ for every $i \in \mathcal{I}$.

3.3 Generalized Ashcroft Invariants

We adapt a class of universally quantified invariants for a generic set of topologies that appear in [9] with the simplifying assumption that program counter variables are available, and the control flow can be encoded through them; this is without loss of generality. The complete formal definition is recalled in [7, Appendix A]. In short, we define QF-Ashcroft as a fragment of first-order formulas over the vocabulary $\mathcal{V}_{\mathsf{Node}} \uplus \mathcal{V}_{\mathsf{Data}} \uplus \{\mathfrak{u}\}$ of two sorts, Node and Data, where the special symbol $\mathfrak{u}$ is typed $\mathsf{Node} \to \mathsf{Data}$, such that no quantification is over Node and every free variable is of sort Node. A generalized Ashcroft assertion of *width* k is a sentence of the form $\forall \nu_1, \ldots, \nu_k.\, \varphi[\nu_1, \ldots, \nu_k]$, where $\varphi \in$ QF-Ashcroft and $\nu_1, \ldots, \nu_k$ are Node-sorted variables.

The truth value of a (generalized) Ashcroft assertion is given by a $\mathcal{V}_{\mathsf{Node}}$-topology $\mathbb{T}$ and a global state $\mu : \mathbb{T} \to \mathbb{D}$, interpreting the symbol $\mathfrak{u}$ (recall that we fix the data domain). For a program $\mathbf{P}$ and a safety specification $\langle \mathsf{init}, \mathsf{error} \rangle$, a *generalized Ashcroft invariant* is an Ashcroft assertion that is *inductive* w.r.t. $\mathbf{P}$ in the standard sense and *sufficient* to establish safety. Φ is an Ashcroft invariant for a parameterized program w.r.t. a parameterized specification if it is an invariant for every member of the family.

4 A Cut-Off Theorem for Ashcroft Invariants

The key observation of this section is as follows. Under certain conditions about the underlying family of topologies, an Ashcroft assertion for *a bounded number of programs* in a parameterized family is also an Ashcroft invariant for the entire parameterized family. This is formally stated as Theorem 1 at the end of this section. This can be viewed as a *cut-off* style theorem for Ashcroft invariants over these topologies. It is intuitively clear that the conditions required for such a theorem to hold have to do with the amount of *symmetry* present in the topology family. In some sense, we want the finite set of programs to enumerate all possible different *scenarios* that one may observe in any instance of the family of any size. In [9], such a condition is defined, albeit used for a different purpose:

Definition 1 ([9]). *Let $k \in \mathbb{N}_+$ and $\mathcal{T}$ be a family of topologies over a vocabulary $\mathcal{V}$. A* basis of rank k *for $\mathcal{T}$ is a set $\mathcal{S}$ of $\mathcal{V}$-topologies where for any $\mathbb{T} \in \mathcal{T}$ and $\boldsymbol{v} \in \mathbb{T}^k$, there is some $\mathbb{S} \in \mathcal{S}$ and embedding $f : \mathbb{S} \hookrightarrow \mathbb{T}$ such that $\boldsymbol{v} \in f(\mathbb{S})^k$.*

Recall the Pipeline example of Fig. 1. Let us assume $k = 3$. If we pick three arbitrary *middle* circle nodes from the pipeline, there are three possibilities, depending on whether any pair is consecutive or not. Figure 1(c,d) illustrates two of them. A full basis of rank 3 with 15 members is listed in [7, Appendix A]. Since the *first* and *last* nodes of the pipeline are distinguished, there are other choices of three *arbitrary* nodes beyond choosing only *middle* points.

We can now state the following new theorem that relates Ashcroft invariants of a parameterized program to Ashcroft invariants of a *finite* parameterized program, specifically the one over the basis:

Theorem 1 *Let $\mathcal{P} = \{\mathbf{P}_i\}_{i\in\mathcal{I}}$ be a parameterized program over a topology family $\mathcal{T} = \{\mathbb{T}_i\}_{i\in\mathcal{I}}$. For any $k \in \mathbb{N}$, if $\mathcal{T}$ contains a basis $\{\mathbb{T}_j\}_{j\in\mathcal{J}}$ of rank $k+1$ as a subset, then for any Φ of width k and parameterized safety specification, Φ is an Ashcroft invariant for $\mathcal{P}$ if and only if it is an Ashcroft invariant for $\{\mathbf{P}_j\}_{j\in\mathcal{J}}$.*

We give a formal proof in [7, Appendix C]. To get a sense of why the theorem holds, and its connection to the *rank of the basis* for the topology family, we give an informal argument for the nontrivial direction here. Assume we have an Ashcroft invariant Φ for the basis $\{\mathbb{T}_j\}_{j\in\mathcal{J}}$, but Φ is not an Ashcroft invariant of the parameterized family. Then, it must not be an invariant of a specific member based on some topology $\mathbb{T}_i$. Assuming the problem is not with initialization or safety of the invariant, there must be a specific statement of a specific node v violating the inductivity of Φ, which must be witnessed by a concrete tuple of nodes $\boldsymbol{u}$ of length k. By the premise and Definition 1, we know that some substructure of $\mathbb{T}_i$ containing v and $\boldsymbol{u}$ can be identified with a topology in the basis, which is a contradiction, because this violation must be observed in that member of the basis against our assumptions.

Observe that Theorem 1 also indicates that checking whether a given Ashcroft assertion is indeed an invariant can be done through a bounded number of checks under the same conditions. In Sect. 5.2, we further argue (see Proposition 4) that these checks are limited to entailments in the data domain $\mathbb{D}$ and hence *independent* of the topology family. In a sense, the topology family through the high level view of Theorem 1 determines a set of such checks, but any individual check is oblivious to it.

Theorem 2 *Let $k \in \mathbb{N}$ and $\mathcal{T}$ be a topology family such that a finite basis of rank $k+1$ contained in $\mathcal{T}$ is given. For any Ashcroft assertion Φ of width k, parameterized program over $\mathcal{T}$, and parameterized safety specification, one can compute a $\mathcal{V}_{\mathsf{Data}}$-sentence that is valid modulo $\mathbb{D}$ iff Φ is an Ashcroft invariant.*

5 From Ashcroft Invariants to Local Proofs

Theorem 1 links the proofs of small programs over the *basis* of the topology to the proof of the entire parameterized family. These proofs, however, are in the form of Ashcroft invariants. In this section, we shift gears to local proofs, which are algorithmically searchable. We first introduce a normalization procedure for Ashcroft invariants, and then model the resulting *local* verification conditions as a set of Hoare triples. Finally, we state the proof search problem for a single program in the topology as a CHC system.

5.1 Normalizing Ashcroft Assertions

A free-form Ashcroft invariant of width k can have an arbitrarily sized and shaped matrix, which can mix and match terms on the topology family and the underlying data. We use the underlying symmetry of the topology family to argue that any Ashcroft invariant can be rearranged in a way that would

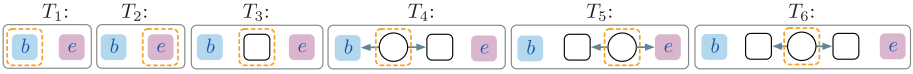
syntactically separate the terms about the topology family and the data, and further put a bound on the size of the universe of terms that can appear in it.

We start with two related key observations: (1) Every *quantifier-free formula* represents the union of a number of *local symmetry equivalence classes*, and (2) Under appropriate conditions, every *local symmetry equivalence class* can be represented by a single *quantifier-free formula*.

There is a well-established concept of *types of quantifier rank* r in model theory (e.g., see [22, Sec. 3.4]), where a special case would be when the rank $r = 0$, and hence the focus is on quantifier-free formulas. Consider a tuple $\boldsymbol{v} \in \mathbb{T}^k$ where $\mathbb{T}$ is a $\mathcal{V}$-structure. The *quantifier-free k-type of* $\boldsymbol{v}$ *over* $\mathbb{T}$, denoted $\mathsf{qftype}(\mathbb{T}, \boldsymbol{v})$, is the set of quantifier-free $\mathcal{V}$-formulas ϕ such that $\mathbb{T} \models \phi(\boldsymbol{v})$. We say $(\mathbb{T}, \boldsymbol{v})$ *realizes* $\mathsf{qftype}(\mathbb{T}, \boldsymbol{v})$. Quantifier-free k-types precisely capture the quotient $\mathbb{T}^k/\simeq$ (i.e. local symmetry equivalence classes) as follows:

Proposition 1 *Let* $\mathbb{T}$ *and* $\mathbb{T}'$ *be* $\mathcal{V}$*-structures. For any tuples* $\boldsymbol{v} \in \mathbb{T}^k$ *and* $\boldsymbol{u} \in \mathbb{T}'^k$*, we have* $\boldsymbol{v} \simeq \boldsymbol{u}$ *if and only if* $\mathsf{qftype}(\mathbb{T}, \boldsymbol{v}) = \mathsf{qftype}(\mathbb{T}', \boldsymbol{u})$.

Example 1 Consider a simplification of the example in Fig. 1, where the top node and associated statements are omitted, and the beginning and ending nodes are exposed as constant symbols instead. There are 6 quantifier-free 1-types that are realized in the topology family $\{\mathbb{T}_n\}_{n \geq 3}$:



where in each case, the tuple (of size 1) realizing the 1-type is marked. □

Under certain finiteness conditions, a *single* quantifier-free formula is sufficient to determine a quantifier-free k-type. We say a family $\mathcal{T}$ of topologies is *uniformly locally finite* (see [16, Sec. 6.4]) if for any $d \in \mathbb{N}_+$, there is a number $n \in \mathbb{N}$ such that the size of any neighbourhood generated by d nodes is bounded above by n, in any member of $\mathcal{T}$. Practically, such topologies can encode networks where every process is connected to a bounded number of shared resources through bounded set of edge labels. Under the assumption of a finite vocabulary and uniform local finiteness, we can establish two important facts:

Lemma 1 *Given a topology family that is uniformly locally finite and has a finite vocabulary, we have:*

(L1) There are finitely many quantifier-free types (or equivalently local symmetry equivalence classes), and each is definable by a single quantifier-free formula. Moreover, these formulas are computable.

(L2) Every quantifier-free formula is equivalent to the disjunction of finitely many quantifier-free formulas, where each defines a distinct local symmetry equivalence class (as per the above item).

Example 2 (Continuation of Example 1) Assume every `pos` is initialized to `true`. Suppose the vocabulary $\mathcal{V}_{\mathsf{Node}}$ consists of unary function symbols `l` and `r` (for left and right), constant symbols `b` and `e`, and the unary predicate symbol `isProc`

indicating a process/circle node. Then, the quantifier-free formula $\alpha_1[\nu]$ defining T_1 from Example 1 is

$$(\nu = \mathtt{l}(\nu) = \mathtt{r}(\nu) = \mathtt{b}) \wedge (\nu \neq \mathtt{e}) \wedge (\mathtt{e} = \mathtt{l}(\mathtt{e}) = \mathtt{r}(\mathtt{e})) \wedge \neg\mathtt{isProc}(\nu) \wedge \neg\mathtt{isProc}(\mathtt{e}).$$

The lemma is analogous to [22, Theorem 3.15], but the proof is slightly more involved due to the function symbols in $\mathcal{V}$. Intuitively, uniform local finiteness ensures that it suffices to consider quantifier-free formulas where the terms are of a bounded height; combined with the finiteness of the vocabulary, it follows that there are finitely many such formulas.

Given an Ashcroft assertion $\forall \boldsymbol{\nu}.\, \varphi[\boldsymbol{\nu}]$, we *normalize* as follows:

1. By pushing negations inward, $\varphi[\boldsymbol{\nu}]$ is rewritten as a positive Boolean combination of formulas of two distinct forms $\phi_{\mathsf{Node}}[\boldsymbol{\nu}]$ and $\phi_{\mathsf{Data}}[\boldsymbol{\nu}]$, where ϕ_{Node} is a quantifier-free $\mathcal{V}_{\mathsf{Node}}$-formula.
2. Using Lemma 1(L2), we replace each sub-formula of the form $\phi_{\mathsf{Node}}[\boldsymbol{\nu}]$ with a disjunction of α_r's, where $\alpha_1, \ldots, \alpha_m$ define the $\simeq$-equivalence classes in the topology family. This results in a formula $\varphi'[\boldsymbol{\nu}]$.
3. We convert $\varphi'[\boldsymbol{\nu}]$ into disjunctive normal form. Observe that, since α_r and $\alpha_{r'}$ $(r \neq r')$ define two disjoint equivalence classes, we have $\alpha_r \wedge \alpha_{r'} \equiv \bot$. Hence, each disjunct can have at most one appearance of some α_r. To ensure that each disjunct has precisely one appearance of some α_r, we do this process instead to the equivalent formula $\varphi'[\boldsymbol{\nu}] \wedge \bigvee_{r=1}^{m} \alpha_r$.
4. After the previous step, we end up with a formula like $\bigvee_{r=1}^{m} \alpha_r[\boldsymbol{\nu}] \wedge \varphi'_r[\boldsymbol{\nu}]$. But, the $\mathcal{V}_{\mathsf{Node}}$-terms that appear under φ'_r can be arbitrary. This means that the same node can be referenced in several different ways. As the last step, we put these $\mathcal{V}_{\mathsf{Node}}$-terms into normal forms so that we impose a unique way of referencing each node. Fortunately, one can argue (see Remark 1) that for every $\simeq$-equivalence class, there is a bounded set of representative terms such that every node can be referenced in a unique way. Let $t_r^{(1)}, \ldots, t_r^{(n_r)}$ be a list of representative terms. We change all references to nodes in $\psi_r[\boldsymbol{\nu}]$ to use the representative terms and get the following formula as the result of the normalization process: $\forall \boldsymbol{\nu}.\, \bigvee_{r=1}^{m} \alpha_r[\boldsymbol{\nu}] \wedge \phi_r[\mathfrak{p}(t_r^{(1)}[\boldsymbol{\nu}]), \ldots, \mathfrak{p}(t_r^{(n_r)}[\boldsymbol{\nu}])]$.

Remark 1 For any k-tuple $\boldsymbol{v}$ over $\mathbb{T} \in \mathcal{T}$, the neighbourhood $N(\boldsymbol{v})$ is of bounded size. Let the tuple $\boldsymbol{u}$ represent all nodes in $N(\boldsymbol{v})$ and for each $i = 1, \ldots |N(\boldsymbol{v})|$. A set of representative terms chooses a term for each u_i, which are by construction free of repetitions. Alternatively, equalities between different possible choices of representative terms appear as formulas in $\mathsf{qftype}(\mathbb{T}, \boldsymbol{v})$, and the choice of the representative term can be viewed as choosing one of the equivalent terms. We call $\boldsymbol{t}$ a list of *representative terms* for $\mathsf{qftype}(\mathbb{T}, \boldsymbol{v})$ where $t^{(i)}(\boldsymbol{v}) = u_i$.

Theorem 3 *Under the condition in Lemma 1, every Ashcroft assertion is equivalent over $\mathcal{T}$ to a normalized Ashcroft assertion of the same width.*

Example 3 (Continuation of Example 1) Let $\alpha_1[\nu], \ldots, \alpha_6[\nu]$ be quantifier-free formulas defining the types $T_1, \ldots, T_6$. Consider the given Ashcroft assertion

$$\forall \nu.(\mathtt{isProc}(\nu) \wedge \mathtt{l}(\nu) \neq \mathtt{b}) \rightarrow \mathtt{ready}(\mathfrak{p}(\mathtt{r}(\nu))) \rightarrow \mathtt{data}(\mathfrak{p}(\mathtt{b})) < \mathtt{data}(\mathfrak{p}(\mathtt{r}(\nu))).$$

After step 1, a ϕ_{Node} subformula in this can be $\neg(\mathtt{isProc}(\nu) \wedge \mathtt{l}(\nu) \neq \mathtt{b})$. It is satisfied in the topology family by a union of equivalence classes, which in this case correspond to $T_1, \ldots, T_4$, and hence this subformula is replaced in step 2 with $\bigvee_{r=1}^{4} \alpha_r[\nu]$. After step 3, the formula $\varphi'_5[\nu]$, corresponding to T_5 (from Example 1), is $\mathtt{ready}(\mathtt{p}(\mathtt{r}(\nu))) \rightarrow \mathtt{data}(\mathtt{p}(\mathtt{b})) < \mathtt{data}(\mathtt{p}(\mathtt{r}(\nu)))$. If we choose $(\nu, \mathtt{l}(\nu), \mathtt{b}, \mathtt{e})$ as the representative terms for T_5, then in step 4 this is rewritten to $\mathtt{ready}(\mathtt{p}(\mathtt{e})) \rightarrow \mathtt{data}(\mathtt{p}(\mathtt{b})) < \mathtt{data}(\mathtt{p}(\mathtt{e}))$. However, if we choose $(\nu, \mathtt{l}(\nu), \mathtt{b}, \mathtt{r}(\nu))$ as the representative terms, then $\varphi'_5[\nu]$ stays as it is. □

5.2 Hoare Triples From a Normalized Ashcroft Invariant

A normalized Ashcroft assertion is basically a template invariant of width k. To instantiate the template, we need to search for interpretations for its uninterpreted symbols. Given a specific program, the verification conditions for the validity of the invariant can be used as constraints in this search. Below, we define precisely what these verification conditions are in the form of Hoare triples obtained from the normalized Ashcroft invariant for a specific program.

Fix vocabularies $\mathcal{V}_{\mathsf{Node}}$ and $\mathcal{V}_{\mathsf{Data}}$. For any formula $\varphi[\nu_1, \ldots, \nu_k]$ in QF-Ashcroft, topology $\mathbb{T}$, and tuple $\boldsymbol{v}$ in $\mathbb{T}^k$, expanding the vocabulary $\mathcal{V}_{\mathsf{Node}}$ with parameters from $\mathbb{T}$, we can obtain a closed QF-Ashcroft formula $\varphi[\boldsymbol{v}]$ by substituting $\boldsymbol{v}$ for the free variables. Then, for a given topology $\mathbb{T}$, any Ashcroft assertion $\Phi := \forall \boldsymbol{\nu}.\, \varphi[\boldsymbol{\nu}]$ is equivalent to a conjunction $\bigwedge_{\boldsymbol{v} \in \mathbb{T}^k} \varphi[\boldsymbol{v}]$, and we can relate Φ to a set of Hoare triples in Fig. 2. In [9, Proposition 6.3], valid Hoare triples are obtained from an Ashcroft invariant in the same style. We observe that the *converse* also holds.

$$
\begin{array}{l}
\forall \boldsymbol{w} \in \mathbb{T}^k, \forall v_0, w \in \mathbb{T}\text{:} \\
\quad \{ \bigwedge_{v \in \mathbb{T}} \mathsf{init}(v) \} \ \epsilon \ \{\varphi[\boldsymbol{w}]\} \qquad \text{(Init)} \\
\quad \{ \bigwedge_{\boldsymbol{v} \in \mathbb{T}^k} \varphi[\boldsymbol{v}] \} \ v_0 \ \{\varphi[\boldsymbol{w}]\} \qquad \text{(Cont)} \\
\quad \{ \bigwedge_{\boldsymbol{v} \in \mathbb{T}^k} \varphi[\boldsymbol{v}] \} \ \epsilon \ \{\neg \mathsf{error}(w)\} \qquad \text{(Safe)}
\end{array}
$$

Fig. 2. Set $H^{\mathbb{T}}_{\Phi}$ of Hoare triples for Φ.

Proposition 2 *Let* $\mathbf{P}$ *be a program over a finite topology* $\mathbb{T}$*. For any* Φ*, the Hoare triples in* $H^{\mathbb{T}}_{\Phi}$ *are valid if* ***and only if*** Φ *is an Ashcroft invariant for* $\mathbf{P}$*.*

We say a Hoare triple is in normal form if its precondition and postcondition are Assertion$[\mathbb{T}]$ formulas for some $\mathcal{V}_{\mathsf{Node}}$-structure $\mathbb{T}$.

Proposition 3 *Any Hoare triple in* $H^{\mathbb{T}}_{\Phi}$ *is equivalent to one in normal form.*

In particular, for a normalized Ashcroft assertion $\forall \boldsymbol{\nu}.\, \varphi[\boldsymbol{\nu}]$, we have

$$\varphi[\boldsymbol{v}] \equiv \phi_r[\mathtt{p}(t_r^{(1)}(\boldsymbol{v})), \ldots, \mathtt{p}(t_r^{(n_r)}(\boldsymbol{v}))] \qquad (1 \leq r \leq m)$$

where r is the unique number such that $T_r = \mathsf{qftype}(\mathbb{T}, \boldsymbol{v})$, also denoted by $r(\boldsymbol{v})$.

For every node v, let x_v be a fresh Data-sorted variable; it is fine to overlap names between different topologies. We define a substitution operation on any

$$\forall \boldsymbol{w} \in \mathbb{T}^k, \forall v_0, w \in \mathbb{T}_i\text{:} \quad \bigwedge_{v \in \mathbb{T}_i} \textsc{Subst}(\mathsf{init}_i(v)) \implies \mathit{Inv}_{r(\boldsymbol{w})}(\boldsymbol{x}_{U(\boldsymbol{w})}) \tag{1}$$

$$\bigwedge_{\boldsymbol{v} \in \mathbb{T}_i^k} \mathit{Inv}_{r(\boldsymbol{v})}(\boldsymbol{x}_{U(\boldsymbol{v})}) \wedge \textsc{Subst}(\mathsf{Global}_{\mathbb{T}_i}[\![v_0]\!]_i) \implies \mathit{Inv}_{r(\boldsymbol{w})}(\boldsymbol{x}'_{U(\boldsymbol{w})}) \tag{2}$$

$$\bigwedge_{\boldsymbol{v} \in \mathbb{T}_i^k} \mathit{Inv}_{r(\boldsymbol{v})}(\boldsymbol{x}_{U(\boldsymbol{v})}) \implies \textsc{Subst}(\neg \mathsf{error}_i(w)) \tag{3}$$

Fig. 3. Constrained Horn Clauses for a single program over topology $\mathbb{T}_i$

Assertion[$\mathbb{T}$] or Transition[$\mathbb{T}$] formula φ: let Subst(φ) be the $\mathcal{V}_{\mathsf{Data}}$-formula obtained from φ by substituting $\mathfrak{u}(v)$ with x_v and $\mathfrak{u}'(v)$ with x'_v for all $v \in \mathbb{T}$. In addition, we use φ' to denote the formula obtained from φ by replacing every $\mathfrak{u}$ with $\mathfrak{u}'$. The following proposition relates the validity of a Hoare triple in normal form to an entailment in the background theory $\mathbb{D}$ for program data. We define $\mathsf{Global}_{\mathbb{T}}[\![v]\!]$ as the formula $[\![v]\!] \wedge \bigwedge_{u \in \mathbb{T} \setminus N(v)} (\mathfrak{u}(u) = \mathfrak{u}'(u))$.

Proposition 4 *Let φ and ψ be* Assertion[$\mathbb{T}$] *formulas.*

- *For any $a \in \mathbb{T}$, the Hoare triple $\{\varphi\}\ a\ \{\psi\}$ is valid if and only if $\mathbb{D} \models$ Subst$(\varphi \wedge \mathsf{Global}_{\mathbb{T}}[\![a]\!] \Rightarrow \psi')$.*
- *The Hoare triple $\{\varphi\}\ \epsilon\ \{\psi\}$ is valid if and only if $\mathbb{D} \models$ Subst$(\varphi \Rightarrow \psi)$.*

5.3 CHC Encoding for Normalized Ashcroft Invariants

We start by using the observations from Propositions 2 and 4 to set up a search for an Ashcroft invariant for a single program in the topology as a CHC encoding. From the normalization procedure (Theorem 3), we collect the quantifier-free k-types T_r and their corresponding representative terms $t_r^{(1)}, \ldots, t_r^{(n_r)}$, for each T_r $(1 \le r \le m)$. Let $\mathbf{P}_i$ be a program over a topology $\mathbb{T}_i \in \mathcal{T}$. The CHC system for searching for an Ashcroft invariant of width k for $\mathbf{P}_i$ is based on unknown predicates of the form $\mathit{Inv}_i : \mathbb{D}^{n_i} \to \mathsf{bool}$ $(1 \le i \le m)$.

The clauses are listed in Fig. 3. For any $\mathbb{T} \in \mathcal{T}$ and tuple $\boldsymbol{v} \in \mathbb{T}^k$, the tuple $\boldsymbol{x}_{\boldsymbol{v}}$ abbreviates $(x_{v_1}, \ldots, x_{v_k})$ and the tuple $U(\boldsymbol{v})$ lists the neighbours of $\boldsymbol{v}$ in the same order as $t_{r(\boldsymbol{v})}$, i.e., $U(\boldsymbol{v}) = (t_j^{(1)}(\boldsymbol{v}), \ldots, t_j^{(n_j)}(\boldsymbol{v}))$, where $j = r(\boldsymbol{v})$.

The clauses are based on Proposition 4 for the validity of the Hoare triples $H_{\Phi}^{\mathbb{T}_i}$ (see Fig. 2), which certify that the Ashcroft assertion Φ is an invariant for the program over $\mathbb{T}_i$.

Theorem 4 *The CHC system of Fig. 3 is (syntactically) satisfiable iff an Ashcroft invariant of width k for $\langle \mathbb{T}_i, [\![-]\!]_i \rangle$ w.r.t. $\langle \mathsf{init}_i, \mathsf{error}_i \rangle$ exists. Moreover, the invariant has the form $\forall \boldsymbol{\nu}.\ \bigvee_{r=1}^{m} \alpha_r[\boldsymbol{\nu}] \wedge \mathit{Inv}_r[\boldsymbol{\mu}(t_r^{(1)}[\boldsymbol{\nu}]), \ldots, \boldsymbol{\mu}(t_r^{(n_r)}[\boldsymbol{\nu}])]$.*

6 Compositional Proof Search for an Ashcroft Invariant

In Sect. 5, we established how to effectively search for ingredients of an Ashcroft invariant for a single program. Combining this with the observation of Theorem 1, one can immediately see how this can be extended to an entire parameterized program. Intuitively, we list all the constraints for all the programs in the basis as stated in Theorem 1.

Theorem 5 *Let $\mathcal{P} = \{\langle \mathbb{T}_i, [\![-]\!]_i \rangle\}_{i \in \mathcal{I}}$ be a parameterized program whose topologies satisfy the finiteness assumptions. Let $\mathcal{S} = \{\langle \mathsf{init}_i, \mathsf{error}_i \rangle\}_{i \in \mathcal{I}}$ be a parameterized specification. For any $k \in \mathbb{N}$, given a finite subset of $\mathcal{T}$ that constitutes a basis of rank $k+1$, one can effectively construct a CHC system that has a solution if and only if there is an Ashcroft invariant of width k for $\mathcal{P}$ w.r.t. $\mathcal{S}$.*

There are however two problems with this process as a practical algorithm: (1) Practically no user-friendly topology family has a finite basis to satisfy the conditions of Theorem 1, and (2) Even for those that do, this process may produce a CHC system with a very high amount of *redundancy*. In [9], the first problem is addressed through a procedure that augments topologies with all the missing small programs to form a finite basis. We use this idea and turn it into a reasonable algorithm that permits the user to specify the input parameterized program using a natural family of topologies (e.g. *rings*) that does not contain a finite basis, and generates the CHC encoding by efficiently deciding what small programs to add and which constraints from Fig. 3 to include.

6.1 Subprograms and Downward Closure

Given a program $\mathbf{P} = \langle \mathbb{T}, [\![-]\!] \rangle$, for any substructure $\mathbb{S}$ of $\mathbb{T}$, the *subprogram* of $\mathbf{P}$ over $\mathbb{S}$ is the program $\mathbf{P}|_{\mathbb{S}} = \langle \mathbb{S}, [\![-]\!]|_{\mathbb{S}} \rangle$. Note that a safety specification for $\mathbf{P}$ can be restricted to a safety specification for any subprogram. Adding subprograms to a parameterized program always preserves unsafety. We show that under certain conditions, it also preserves safety. Given a topology $\mathbb{T}$, we say an initial condition init is *extensible* if for any substructure $\mathbb{S}$ of $\mathbb{T}$, for any initial global state μ on $\mathbb{S}$, there is an initial global state μ' on $\mathbb{T}$ such that $\mu'|_{\mathbb{S}} = \mu$.

Theorem 6 *Let $\mathcal{P}$ and $\mathcal{P}'$ be parameterized programs such that $\mathcal{P} \subset \mathcal{P}'$ and every program of $\mathcal{P}'$ is a subprogram of some program in $\mathcal{P}$. Given any parameterized safety specification for $\mathcal{P}$, the safety of $\mathcal{P}'$ implies the safety of $\mathcal{P}$, and the converse holds if the initial condition on every topology in $\mathcal{P}$ is extensible.*

For any topology $\mathbb{T}$, define $\mathsf{sub}_k(\mathbb{T})$ as the set of substructures of $\mathbb{T}$ generated by k not necessarily distinct nodes. For any parameterized program $\mathcal{P} = \{\mathbf{P}_i\}_{i \in \mathcal{I}}$, where $\mathbf{P}_i$ is over topology $\mathbb{T}_i$, define $\mathsf{CL}(\mathcal{P}, k)$ as the parameterized program obtained from $\mathcal{P}$ by adding the subprogram $\mathbf{P}_i|_{\mathbb{S}}$ for any $i \in \mathcal{I}$ and $\mathbb{S} \in \mathsf{sub}_k(\mathbb{T}_i)$. The following proposition, which follows from [9, Proposition 6.7], presents a finite basis of rank k contained in the topologies of $\mathsf{CL}(\mathcal{P}, k)$ under the assumptions in Theorem 3.

Proposition 5 *Let $k \in \mathbb{N}_+$ and $\mathcal{T}$ be a locally uniformly finite class of topologies over a finite vocabulary. Let $\mathcal{S} := \bigcup_{\mathbb{T} \in \mathcal{T}} \mathsf{sub}_k(\mathbb{T})$. Then $\mathcal{S}$ is a basis for $\mathcal{T}$ of rank k that is finite up to isomorphism.*

6.2 CHCs for Downward Closure

If we blindly collect Hoare triples from all small programs in the basis, we end up with redundant Hoare triples with the same postcondition and command, but different preconditions, where one is subsumed by the other in a purely syntactic way. Avoiding these redundancies by constructing the full set and pruning it is clearly not a desirable solution. In Algorithm 1, we present an alternative. It takes as input a parameterized program $\mathcal{P}$, a parameterized specification $\mathcal{S}$, and a number $k \in \mathbb{N}_+$, and returns a CHC system that encodes the existence of an Ashcroft invariant for the downward closure of $\mathcal{P}$. The indexed sets $\mathcal{P}$ and $\mathcal{S}$ are given as computable functions. We assume there is an oracle $\text{QF-Types}(\mathcal{T}, k)$ that returns a list of all quantifier-free k-types in $\mathcal{T}$, where each type is represented by a k-tuple in a topology in $\mathcal{T}$. The algorithm may simplify the verification condition of one program by deferring to that of a smaller program, leveraging the fact that the *smallest* substructure containing the nodes in a command and postcondition is in the downward closure.

Algorithm 1. CHCs for a Downward Closure

Require: Program $\mathcal{P} = \{\langle \mathbb{T}_i, [\![-]\!]_i \rangle\}_{i \in \mathcal{I}}$ on topologies $\mathcal{T} = \{\mathbb{T}_i\}_{i \in \mathcal{I}}$ satisfying the finiteness assumptions, $\mathcal{S} = \{\mathbf{S}_i\}_{i \in \mathcal{I}}$ is a parameterized specification, and $k \in \mathbb{N}_+$

procedure CHCs-for-Downward-Closure($\mathcal{P}, \mathcal{S}, k$)
  compute $\alpha_1, \ldots, \alpha_m$ and representative terms $t_r^{(j)}$ as per Theorem 3
  $\mathcal{C} \leftarrow \varnothing$
  for all $(\mathbb{T}_i, \boldsymbol{w}) \in \text{QF-Types}(\mathcal{T}, k)$ **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \left\{ \bigwedge_{v \in N(\boldsymbol{w})} \text{Subst}(\mathsf{init}_i(v)) \Rightarrow \mathit{Inv}_{r(\boldsymbol{w})}(\boldsymbol{x}_{U(\boldsymbol{w})}) \right\}$
  for all $(\mathbb{T}_i, v_0\, \boldsymbol{w}) \in \text{QF-Types}(\mathcal{T}, k+1)$ **do**
    $\phi \leftarrow \text{Subst}(\mathsf{Global}_{N(v_0 \boldsymbol{w})}[\![v_0]\!]_i)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \left\{ \bigwedge_{\boldsymbol{v} \in N(v_0 \boldsymbol{w})^k} \mathit{Inv}_{r(\boldsymbol{v})}(\boldsymbol{x}_{U(\boldsymbol{v})}) \wedge \phi \Rightarrow \mathit{Inv}_{r(\boldsymbol{w})}(\boldsymbol{x}'_{U(\boldsymbol{w})}) \right\}$
  for all $(\mathbb{T}_i, w) \in \text{QF-Types}(\mathcal{T}, 1)$ **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \left\{ \bigwedge_{\boldsymbol{v} \in N(w)^k} \mathit{Inv}_{r(\boldsymbol{v})}(\boldsymbol{x}_{U(\boldsymbol{v})}) \Rightarrow \text{Subst}(\neg \mathsf{error}_i(w)) \right\}$
  return $\mathcal{C}$

Theorem 7 *The algorithm* CHCs-for-Downward-Closure$(\mathcal{P}, \mathcal{S}, k)$ *returns a CHC system that is (syntactically) satisfiable if and only if there is an Ashcroft invariant of width k for $\mathsf{CL}(\mathcal{P}, k+1)$ w.r.t. $\mathcal{S}$.*

There are $|\text{QF-Types}(\mathcal{T}, k)|$ unknown predicates produced by Algorithm 1, the same as Theorem 5, and $O(|\text{QF-Types}(\mathcal{T}, k+1)|)$ clauses are generated. Both are characteristics of the topology family, which reflect its complexity.

For a specific topology family $\mathcal{T}$, the procedure QF-Types$(\mathcal{T}, k)$ can often be implemented by enumerating node tuples of length k in small instances of $\mathcal{T}$. For instance, if $\mathcal{T}$ is the family of lines in the style of Fig. 1 (a), it suffices to enumerate node tuples in line instances with up to $2k + 1$ circle nodes.

7 Complete Encoding Optimizations

A key feature of our solution is that the normalization process grants us flexibility and control in devising the *complete* template for proof search. In this section, we take advantage of this control to introduce a couple of *complete* optimizations for the encoding. We first lay the formal foundation for these optimizations through a *parametric* variance of the normalization process introduced in Sect. 5.1. Concrete optimizations are defined as instantiations of the parameter. This means that we can give a single proof of completeness for the template (Theorem 8) and the completeness of all individual optimizations presented in this section immediately follows. The soundness is trivial, as any instantiation of a restricted template can be converted into an instantiation of a full template.

7.1 Parametric Normalization of Ashcroft Invariants

Let us continue with the normalization procedure in Sect. 5.1. We claim that with further syntactic manipulations to a normalized Ashcroft assertion, the formulas ϕ_r's can be guaranteed to be *symmetric*: Let the types in the topology family be $T_1, \ldots, T_m$ and $\boldsymbol{v}_r$ be a node tuple that realizes T_r for each r. We say the formula ϕ_r is *symmetric* (w.r.t. the type T_r) if for any automorphism f on $N(\boldsymbol{v}_r)$, we have $\phi_r[x_1, \ldots, x_{n_r}] \equiv \phi_r[x_{\pi_f(1)}, \ldots, x_{\pi_f(n_r)}]$, where π_f is the permutation on indices $\{1, \ldots, n_r\}$ induced by f and the representative terms of T_r. We exploit the symmetry of these formulas in Sect. 8.

Moreover, some ϕ_r's can be assumed to be $\top$ (true). We use an *indicator* vector $\boldsymbol{\chi} \in \{\top, \bot\}^m$ to *indicate* which. We say $\boldsymbol{\chi}$ *selects a basis of rank* k from $\mathcal{T}$ if the collective neighbourhoods of the node tuples at the indicated indices form a basis of rank k for the topology; formally, if $\mathcal{U} := \{N(\boldsymbol{v}_r) : \chi_r \equiv \top\}$ is a basis of rank k for $\mathcal{T}$.

Theorem 8 *Assume the setting of Theorem 3. Let $\boldsymbol{\chi} \in \{\top, \bot\}^m$ be any indicator vector that selects a basis of rank k from $\mathcal{T}$. Then over $\mathcal{T}$, any Ashcroft assertion is equivalent to a normalized Ashcroft assertion of the same width*

$$\forall \nu_1, \ldots, \nu_k. \bigvee_{r=1}^{m} \alpha_r[\boldsymbol{\nu}] \wedge \phi_r[\boldsymbol{\mu}(t_r^{(1)}[\boldsymbol{\nu}]), \ldots, \boldsymbol{\mu}(t_r^{(n_r)}[\boldsymbol{\nu}])]$$

such that for every r, ϕ_r is symmetric w.r.t. the type T_r, and $\phi_r \equiv \top$ if $\chi_r \equiv \bot$.

The proof idea is that, given a formula $p[x_1, x_2]$, the conjunction $p[x_1, x_2] \wedge p[x_2, x_1]$ is invariant under any permutation on the indices $\{1, 2\}$.

We use the symmetry of ϕ_r's in Sect. 8 and the indicator vector $\boldsymbol{\chi}$ to define optimizations in this section. Given $\boldsymbol{\chi}$, we may modify Algorithm 1 by skipping $\boldsymbol{w}$ such that $\chi_{r(\boldsymbol{w})} \equiv \bot$ in line 4 and 6 and, similarly, skipping $\boldsymbol{v}$ such that $\chi_{r(\boldsymbol{v})} \equiv \bot$ in line 8 and 10. For an input parameterized program $\mathcal{P}$ to the algorithm, this simplification is *complete* if $\boldsymbol{\chi}$ selects a basis of rank k from the topology family of $\mathsf{CL}(\mathcal{P}, k+1)$.

7.2 One Predicate per Neighbourhood

By default, our CHC encoding uses one unknown predicate per quantifier-free k-type $T_1, \ldots, T_m$ in $\mathcal{T}$. Let $\mathbb{U}_r$ be the neighbourhood of an arbitrary tuple with type T_r, which is unique up to isomorphism. We define an equivalence relation on the types $T_1, \ldots, T_m$ based on these neighbourhoods: we say $T_r \sim T_s$ if $\mathbb{U}_r$ is isomorphic to $\mathbb{U}_s$. Then, the $\sim$ relation provides a choice of $\boldsymbol{\chi}$ that always induces a complete optimization to Algorithm 1: for each equivalence class in $\{T_1, \ldots, T_m\}/\!\sim$, χ_r chooses precisely one representative from the class. The net effect is that we use one unknown predicate per neighbourhood generated by k nodes. In Sect. 9, we refer to this optimization as `OPN`.

7.3 Distinct Process Nodes as Generators

We start by assuming that the nodes in a topology $\mathbb{T}$ are partitioned into two kinds: process nodes and resource nodes; for example, circles and squares in Fig. 1. Resource nodes do not execute code. We add a unary predicate isProc to the vocabulary $\mathcal{V}_{\mathsf{Node}}$ to indicate whether a node is a process node. We require that the neighbourhood of any resource node contains no process node, and every resource node is in the neighbourhood of some process node.

Skipping Resource Nodes. Recall that Algorithm 1 implicitly adds subprograms over the topologies $\{N(\boldsymbol{v}) \mid \boldsymbol{v} \text{ is a } (k+1)\text{-tuple over some } \mathbb{T} \in \mathcal{T}\}$ to guarantee a basis of rank $k+1$. Under the modelling assumptions mentioned above, however, it suffices to consider $\boldsymbol{v}$ consisting of process nodes. We can exclude the rest, which are theoretically unnecessary by, for each r, setting χ_r to $\bot$ if $\boldsymbol{v}_r$ (an arbitrary tuple that realizes T_r, as defined in Sect. 7.1) contains a resource node. Meanwhile, in lines 6 and 9 of Algorithm 1, we also skip any v_0 and w that are resource nodes.

Distinct Generators. We can further exclude topologies generated by fewer than k process nodes from the downward closure, by setting χ_r to $\bot$ for any r such that $\boldsymbol{v}_r$ contains repetitive nodes. Since line 9–10 of Algorithm 1 implicitly closes $\mathcal{T}$ under substructures generated by one node, we change them to the following:

for all $(\mathbb{T}, w) \in \textsc{Substructures}(\mathcal{T}, k, 1)$ **do**

$$\mathcal{C} \leftarrow \mathcal{C} \cup \left\{ \bigwedge_{\boldsymbol{v} \in \mathbb{T}^k : \chi_{r(\boldsymbol{v})}} Inv_{r(\boldsymbol{v})}(\boldsymbol{x}_{U(\boldsymbol{v})}) \Rightarrow \textsc{Subst}(\neg \mathsf{error}_i(w)) \right\}$$

where $\textsc{Substructures}(\mathcal{T}, k, 1)$ returns, up to isomorphism, a list of all structures $\mathbb{T}$ together with a distinguished node $w \in \mathbb{T}$ such that $\mathbb{T}$ is generated by k distinct process nodes over any topology in $\mathcal{T}$. We refer to the combined optimization in this subsection as `DPG`.

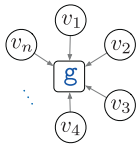
8 Theoretical Case Study: Deriving the Solution of [17]

To illustrate that the solution presented in this paper is robust, we look at the specific classic topology of *identical replicated threads sharing boundedly many global variables*, which is the premise of [17], and show that the solution of [17] is a special case of our solution.

$$
\begin{aligned}
\mathit{Init}(x_0, x_1) \wedge \mathit{Init}(x_0, x_2) &\implies \mathit{Inv}(x_0, x_1, x_2) \\
\mathit{Inv}(x_0, x_1, x_2) \wedge \mathit{Inv}(x_0, x_2, x_1) \wedge s(x_0, x_1, x_0', x_1') &\implies \mathit{Inv}(x_0', x_1', x_2) \\
\mathit{Inv}(x_0, x_1, x_2) \wedge \mathit{Inv}(x_0, x_2, x_1) \wedge s(x_0, x_2, x_0', x_2') &\implies \mathit{Inv}(x_0', x_1, x_2') \\
\mathit{Inv}(x_0, x_1, x_2) \wedge \mathit{Inv}(x_0, x_3, x_2) \wedge \mathit{Inv}(x_0, x_1, x_3) & \\
\mathit{Inv}(x_0, x_2, x_1) \wedge \mathit{Inv}(x_0, x_2, x_3) \wedge \mathit{Inv}(x_0, x_3, x_1) \wedge s(x_0, x_3, x_0', x_3') &\implies \mathit{Inv}(x_0', x_1, x_2) \\
\mathit{Inv}(x_0, x_1, x_2) \wedge \mathit{Inv}(x_0, x_2, x_1) &\implies \neg \mathit{Error}(x_0, x_1)
\end{aligned}
$$

Fig. 4. Clauses generated by Algorithm 1 for the star topology.

First, we model the topology as a *star*. The vocabulary $\mathcal{V}_{\mathsf{Node}}$ consists of a single constant symbol g. The $\mathcal{V}_{\mathsf{Node}}$-structure $\mathbb{T}_n$ has underlying set $\{0, 1, \ldots, n\}$, where 0 is the interpretation of g (the resource node hosting all shared variables), and the positive numbers stand for the n indistinguishable process nodes v_i's.



In [17], the safety specification is given as a pair of $\mathcal{V}_{\mathsf{Data}}$-formulas *Init* and *Error*, and the transitions are given by a $\mathcal{V}_{\mathsf{Data}}$-formula s. To simplify the presentation, we fix $k = 2$ and assume *Error* is stated for a single node. In our notation, for any $n \in \mathbb{N}_+$ and $v \neq 0$, we have $\mathsf{init}_n(v) = \mathit{Init}(\mathfrak{µ}(0), \mathfrak{µ}(v))$, $\mathsf{error}_n(v) = \mathit{Error}(\mathfrak{µ}(0), \mathfrak{µ}(v))$, and $[\![v]\!]_n = s(\mathfrak{µ}(0), \mathfrak{µ}(v), \mathfrak{µ}'(0), \mathfrak{µ}'(v))$; for $v = 0$, we have $\mathsf{init}_n(v) = \top$, $\mathsf{init}_n(v) = \bot$, and $[\![v]\!]_n = \bot$.

Let $\mathcal{P} := \{\langle \mathbb{T}_n, [\![-]\!]_n \rangle\}_{n \geq 2}$. The refined downward closure process in Sect. 7.3 does not add any new subprogram to $\mathcal{P}$, as the topology family already contains $\{\mathbb{T}_2, \mathbb{T}_3\}$, which is also a basis of rank 3.

Our normalization produces $\forall u_1, u_2.\, \alpha[u_1, u_2] \rightarrow \phi[\mathfrak{µ}(\mathsf{g}), \mathfrak{µ}(u_1), \mathfrak{µ}(u_2)]$, where $\alpha[u_1, u_2] := u_1 \neq u_2 \wedge \mathsf{isProc}(u_1) \wedge \mathsf{isProc}(u_2)$, as a *complete* template for an Ashcroft assertion of width 2 for $\mathcal{P}$, which incidentally is precisely how an Ashcroft assertion is defined in [17].

Running Algorithm 1 modified with the optimization described in Sect. 7.3, with minor syntactic simplification, we obtain the clauses $\mathcal{C}$ in Fig. 4.

By the discussion in Sect. 7.1, without loss of generality, we may assume that $Inv(x_0, x_1, x_2) \equiv Inv(x_0, x_2, x_1)$ for all x_0, x_1, x_2, which prunes the clauses illustrated in gray, and results in a set of clauses $\mathcal{C}'$ that is equisatisfiable to $\mathcal{C}$. The set of clauses $\mathcal{C}'$ is exactly the CHC encoding for 2-thread-modular proofs [17, Figure 6], except that we do not treat control locations explicitly.

9 Experimental Results

Implementation. Our tool, Mosaic, takes as input a number $k \in \mathbb{N}_+$, a parameterized program over topologies $\mathcal{T} = \{\mathbb{T}_i\}_{i \in \mathcal{I}}$ over a finite vocabulary $\mathcal{V}$ (assuming the indices $\mathcal{I}$ are consecutive natural numbers), a parameterized safety specification, and a pair of numbers $i, i' \in \mathcal{I}$ with the assumption that all quantifier-free $(k+1)$-types appear in $\mathcal{T}$ already appear in $\{\mathbb{T}_j \mid i \leq j \leq i'\}$. For the parameterized safety specification, we allow a more general form of error states that can relate m nodes in a topology, where $m \leq k$. Mosaic implements Algorithm 1 with optimization options `OPN` and `DPG` (Sect. 7). The output encoding is in SMT-LIB2 format and can be passed to a variety of existing CHC solvers [6,18,21]. Here, we report the results for Z3/Spacer [10,21] except for one benchmark, which can be solved by Golem [6] but not Spacer. Details about the experiment can be found in [7, Appendix D].

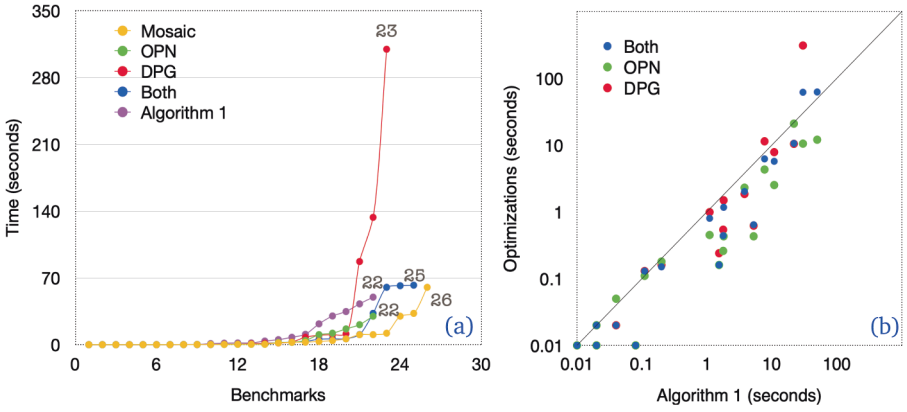


Fig. 5. (a) Quantile plot. (b) Scattered plot of optimizations vs baseline.

Evaluation Goals. The main goal of our evaluation is straightforward: Establishing whether Mosaic can prove interesting properties of parameterized programs over a rich family of topologies. As a secondary goal, we evaluate the impact of the optimizations proposed in Sects. 7.2 and 7.3.

Benchmarks. There is not a lot of diversity in the example pool in the literature when it comes to different topologies for parameterized programs. Hence,

beyond including a few examples from the literature, we designed many more benchmarks over many more topologies. In particular, we have 30 programs over (1) rings, (2) lines (like example of Fig. 1), (3) grids, (4) bounded-depth trees [9], and (5) binary trees (full list in [7, Appendix D.1]). The generality and flexibility offered by logical structures are very helpful in modelling some extra requirements for specialized versions of some of these topologies. For instance, a direction predicate for the line and an orientation predicate for the ring can be added effortlessly, and our benchmark set also includes these variations.

Results. Figure 5 presents our results, where Mosaic corresponds to the best performance of all options. The quantile plot includes the count of benchmarks solved by each option. Of the 26 benchmarks solved by Mosaic, the winning time of 16 is by `OPN`, 8 by `DPG`, 13 by the combination, and 4 by Algorithm 1 (counting ties). The optimizations clearly have a significant impact. Remarkably, they do not consistently lead to an improved time: There are a few outliers above the middle line in Fig. 5(b), which incidentally do not include the 3 cases where Algorithm 1 times out. This is because they target the size of the encoding, but a smaller encoding is not necessarily always easier to solve for the backend solver. [7, Appendix D.2] lists the full results, including encoding sizes.

The 4 benchmarks that Mosaic cannot solve are due to one of the following reasons: (1) non-linear reasoning required (1 tree benchmark), (2) encoding is too large (1 grid benchmark), and (3) CHC solver fails (1 line and 1 ring benchmark).

Limitations and Discussion. The key limitations are related to the unscalability and unpredictability of the backend CHC solvers. Large encodings (e.g. a grid example) cannot be handled. A minor change in Fig. 1 example can flip the status of its solvability, without much change in the encoding size.

Another bottleneck for scalability is that the basis of rank $k+1$ becomes too large as k increases, especially for a larger vocabulary $\mathcal{V}_{\mathsf{Node}}$. For instance, this happens for width-2 invariants for grid examples, since the distinctness of the four neighbours causes the basis to include 347 small programs even with optimizations, and one of two such examples times out.

Finally, we came across benchmarks (not included) that do not admit universally quantified inductive invariants as proofs, where a richer class of invariants or ghost variables are required to handle them.

10 Related Work

The problem of verification of parameterized programs is generally known to be undecidable [4]. Our focus is on the *algorithmic* verification of infinite-state parameterized programs. As such, we do not survey the rich literature on finite-state parameterized programs [1] nor deductive verification techniques [5,28].

Compositional Proofs of Parameterized Programs. Compositional proofs have been a central theme in concurrency verification. We made a thorough comparison with the work in [17] in Sect. 8, which is singled out because it is

sound and complete, albeit for a fixed topology. Similar results, but for finite-state programs, have appeared in [24]. However, general cut-off results like ours and the one in [17] for the star topology were not given.

The work in [5,25–27] is noteworthy for the common themes of the use of symmetry in compositional verification. In particular, in [26], the insight of identifying *balanced* (therefore *locally symmetric*) nodes in compositional verification is used to establish *compositional cutoffs*. The programs are finite-state, and the key result is the decidability of compositional model checking in polynomial time. Specifically, the problem is formally framed as deciding, for all programs in the parameterized family, the existence of interference-free local proofs for every process, which collectively become a global universal invariant *with one quantifier*. Our work generalizes this view by permitting universal quantification of any depth and generalizing compositional cutoffs to bases of any rank. However, the starting point of [26] is local proofs, whereas we start with global invariants and look for a sound and complete reduction to local proofs.

Handling Multiple Communication Topologies. Several topologies are studied for finite-state programs with multiple communication primitives in [2], where the decision procedure for each topology requires a bespoke representation of the configurations and a pre-order on them that induces subconfigurations for abstraction. Later in [3], parameterized finite-state programs over a rich set of topologies are studied, where several uniform (across topology classes) decidability and complexity results are given. The focus for decidability is on what they call *homogeneous* topologies, and none of the topologies in our benchmark set is homogeneous. In particular, there are known undecidability results [11] about rings. For infinite-state programs, a rich class of topologies are targeted in [5] in the context of deductive verification. However, the invariant templates are predetermined by the user, and their small model property relates to checking concrete verification conditions rather than safety through Ashcroft invariants.

Parameterized Model Checking of Infinite-State Programs. For algorithmic verification of infinite-state parameterized programs, there exist techniques that automatically generate universally quantified invariants [12,15,19,23], all for a single topology. One can view *thread-modular proofs at many levels* [13,17] as the conclusion of this line of work by proposing proof systems that form a strict hierarchy and are relatively complete w.r.t. Ashcroft invariants.

11 Conclusion and Future Work

On the conceptual side, this paper observes that under certain conditions about the families of topologies, a universally quantified invariant of an unbounded family of concurrent parameterized programs over these topologies can be verified through a bounded number of checks (Theorem 1), which are limited to entailments in the underlying data domain (Theorem 2). Hence, verification conditions do not require any axiomatization of the underlying topology family. On the algorithmic side, using the above observations, we propose a *complete* algorithm for discovering such an invariant of a given width k, if one exists.

Our implementation uses CHC encodings (and solvers) as a point of convenience. The solvers do not yet scale to the level of handling the encodings of larger programs. The paper offers two *complete* optimizations to improve the encoding, but a more practical algorithm will have to rely on a combination of abstraction and decomposition techniques to make the invariant search tractable. We hope that our conceptual contribution in separation of reasoning about data and topologies will bring new opportunities for discovery on this path.

Recall that Algorithm 1 relies on an oracle that computes the quantifier-free types for a topology family. In the implementation, these are computed through brute-force enumeration relying on upper-bounds that are manually calculated for the topology class. The technical problem of necessary/sufficient conditions that guarantee efficient computability of quantifier-free types remains open.

Universally quantified invariants are clearly not enough to capture every proof (or even every specification) of interest in these contexts. It will be interesting to investigate if similar cut-off results can be proven for richer classes of invariants with quantifier alternations (e.g., see [20]), and independently, if commutativity-based reductions can be used so that a universally quantified invariant is sufficient for proving the reduced parameterized program safe (see [14] for the star topology).

Data Availibility Statement. Proofs and experiment details can be found at [7]. Our tool Mosaic and the benchmarks used in the evaluation (Sect. 9) can be found at [8], where version 0.1.1 was submitted for artifact evaluation and includes the first 21 benchmarks listed in [7, Appendix D.1].

Disclosure of Interests. Nothing relevant to declare.

References

1. Abdulla, P.A., Delzanno, G.: Parameterized verification. Int. J. Softw. Tools Technol. Transfer **18**(5), 469–473 (2016). https://doi.org/10.1007/s10009-016-0424-3
2. Abdulla, P.A., Haziza, F., Holík, L.: All for the price of few. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Verification, Model Checking, and Abstract Interpretation, pp. 476–495. Springer, Cham (2013). https://doi.org/10.1007/978-3-642-35873-9_28
3. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. Distrib. Comput. **31**(3), 187–222 (2018). https://doi.org/10.1007/s00446-017-0302-6
4. Apt, K.R., Kozen, D.C.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. **22**(6), 307–309 (1986). https://doi.org/10.1016/0020-0190(86)90071-2
5. Ashmore, R., Gurfinkel, A., Trefler, R.: Local Reasoning for Parameterized First Order Protocols. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2019. LNCS, vol. 11460, pp. 36–53. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20652-9_3
6. Blicha, M., Britikov, K., Sharygina, N.: The Golem Horn Solver. In: Enea, C., Lal, A. (eds.) Computer Aided Verification, pp. 209–223. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_10

7. Cheng, R., Farzan, A.: Complete Local Reasoning About Parameterized Programs Over Topologies (2026). https://doi.org/10.48550/arXiv.2605.15143
8. Cheng, R., Farzan, A.: Mosaic: artifact for "complete local reasoning about parameterized programs over topologies". Zenodo (2026). https://doi.org/10.5281/ZENODO.19851981
9. Cheng, R., Farzan, A.: Symmetric Proofs of Parameterized Programs (2026). https://doi.org/10.48550/arXiv.2601.18745
10. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
11. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 85–94. POPL '95, Association for Computing Machinery, New York, NY, USA (1995). https://doi.org/10.1145/199448.199468
12. Emmi, M., Majumdar, R., Manevich, R.: Parameterized verification of transactional memories. In: Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 134–145. PLDI '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1806596.1806613
13. Farzan, A., Kincaid, Z., Podelski, A.: Proof spaces for unbounded parallelism. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 407–420. ACM, Mumbai India (2015). https://doi.org/10.1145/2676726.2677012
14. Farzan, A., Klumpp, D., Podelski, A.: Commutativity Simplifies Proofs of Parameterized Programs. Proc. ACM Program. Lang. **8**(POPL), 2485–2513 (2024). https://doi.org/10.1145/3632925
15. Gurfinkel, A., Shoham, S., Meshman, Y.: SMT-based verification of parameterized systems. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 338–348. FSE 2016, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2950290.2950330
16. Hodges, W.: A Shorter Model Theory. Cambridge University Press, Cambridge; New York (1997)
17. Hoenicke, J., Majumdar, R., Podelski, A.: Thread modularity at many levels: A pearl in compositional verification. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, pp. 473–485. ACM, Paris, France (2017). https://doi.org/10.1145/3009837.3009893
18. Hojjat, H., Rümmer, P.: The ELDARICA Horn Solver. Formal Methods in Computer Aided Design (FMCAD), pp. 1–7 (2018). https://doi.org/10.23919/FMCAD.2018.8603013
19. Hojjat, H., Rümmer, P., Subotic, P., Yi, W.: Horn clauses for communicating timed systems. EPTCS **169**, 39–52 (2014)
20. Koenig, J.R., Padon, O., Immerman, N., Aiken, A.: First-order quantified separators. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 703–717. ACM Conferences, Association for Computing Machinery (2020). https://doi.org/10.1145/3385412.3386018
21. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Formal Methods Syst. Des. **48**(3), 175–205 (2016). https://doi.org/10.1007/s10703-016-0249-4
22. Libkin, L.: Elements of Finite Model Theory. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-662-07003-1

23. Monniaux, D., Gonnord, L.: Cell Morphing: From Array Programs to Array-Free Horn Clauses. In: Rival, X. (ed.) SAS 2016. LNCS, vol. 9837, pp. 361–382. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53413-7_18
24. Namjoshi, K.S.: Symmetry and Completeness in the Analysis of Parameterized Systems. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 299–313. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69738-1_22
25. Namjoshi, K.S., Trefler, R.J.: Local Symmetry and Compositional Verification. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 348–362. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_23
26. Namjoshi, K.S., Trefler, R.J.: Parameterized Compositional Model Checking. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 589–606. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_39
27. Namjoshi, K.S., Trefler, R.J.: Symmetry Reduction for the Local Mu-Calculus. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 379–395. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_22
28. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. ACM SIGPLAN Notices **51**(6), 614–630 (2016). https://doi.org/10.1145/2980983.2908118