Reachability in Parallel Programs Is Polynomial in the Number of Threads (Version with Proofs)

Alexander Malkis

Abstract

Reachability in parallel finite-state programs equipped with interleaving semantics is an important yet inherently difficult problem. Its complexity as a function of the number of threads n, while keeping the thread-local–memory size and the shared-memory size bounded by constants, has been explored only poorly. We significantly narrow this gap by measuring: (i) the *diameter*, i.e., the longest finite distance realizable in the transition graph of the program, (ii) the *local diameter*, i.e., the maximum finite distance from any program state to any thread-local state, and (iii) the computational complexity of finding bugs. We prove that all these are majorized by polynomials in n and, in certain cases, by linear, logarithmic, or even constant functions in n; we make the bounds explicit whenever possible. Our results shed new light on the widely expressed claim that one of the major obstacles to analyzing parallel programs is the exponential state explosion in the number of threads.

Keywords: multithreading, concurrency, transition graph, diameter, asynchronous execution, shared-memory communication, interleaving, complexity, exponential blow-up, state-space explosion, formal methods, threads, parallelism, operational semantics, formal languages, combinatorics, counting problems, graph theory, path problems.
2020 MSC (most applicable first): Primary: 68Q85, 68Q10, 05C12. Secondary: 68Q17, 05C30, 94A05, 68Q25, 90B40, 68Q04, 68Q45, 03D15, 68Q87, 68R10, 68R05, 05C20, 94C15, 68M10.

0. Introduction

Since 2004, the CPU-clock limit has been stagnating, which has immensely increased the demand for multithreading [85]. Conceptually, a multithreaded program consists of a batch of threads running in parallel; each thread can access only its private memory and the memory shared among the threads. The semantics of accessing the shared memory can be cumbersome [70, 79]; to simplify writing and analyzing multithreaded code, the code is typically written in such a way that every execution on a parallel machine can be viewed as an execution on a sequential machine that interleaves the steps of different threads. Even given this simplified framework, programming errors are widespread [45, 88], and the effects of failures can be devastating [5, 56, 73, 74]. In practice, almost every such failure can be viewed as a violation of a so-called safety property, which is, informally speaking, a property of the form "nothing bad happens in all executions."

We focus on the most basic safety properties of the form "a program state is not reachable from another program state"; any safety property can be reduced to such a state-to-state unreachability property [68]. Deciding such properties of programs in practice often incurs the infamous state-explosion problem, which is the phenomenon whereby the number of program states grows exponentially with the number of threads (hereinafter n) of the program analyzed [7, 19, 35, 40, 41, 76]. It is a practical "problem" because the program analyzer often runs out of resources while reasoning about these states and, much to a user's disappointment, fails to deliver a conclusive answer. In fact, reachability in finite-state multithreaded programs equipped with interleaving semantics is PSpace-complete (cf. § II.3). Since PSpace is a very robust class [81, § 8.2 and Exercise 8.4] containing a wealth of complete decision problems [46, 91], the PSpace-completeness characterization is rather unenlightening. In particular, it does not reveal too many details about the exponential blow-up with respect to n from a theoretical viewpoint (and the strictness of the inclusion PTime $\subset$ PSpace is only a conjecture so far anyway).

To study the state-explosion phenomenon, we consider a parametrized setting with a variable number of threads n and two constant parameters: the size of the local memory per thread and the size of shared memory. In this setting, we ask how the following quantities asymptotically grow with n: the diameter and the local diameter (both of which we define later) and the complexity of two natural reachability problems. At the core of our work is the informal question of whether the growth is fast (as suggested by the aforementioned PSpace complexity and by the state explosion occurring in the tools) or slow (sometimes occurring in the parametrized-complexity field).

As we will show, the second case holds. For reachability tasks that can be formulated in the parametrized setting described, the aforementioned blow-up and high complexity can be asymptotically avoided; we will quantify this statement.

The variable number of threads and the bounded sizes of local memory per thread and of shared memory might be observed in several areas; here, we name three. The first of these areas is high-performance computing; we consider applications in which the threads themselves are fixed in size, whereas what changes is the number of threads executed in parallel when a program is moved from one supercomputer to another (or, to a lesser extent, from one GPU to another) or when a program goes from a test setup to a fully parallel setup. (At the time these lines were written, Web search [36] returned numerous occurrences of "char thread_id;" and "char threadId;" in real-world C code, which allocated 8 bits for the thread identifier. Such pieces of code are likely to be of limited use or even erroneous on a system with more than 256 threads.) The second area is the modeling of memory limitations in dynamic systems. (A typical bug example is a thread-identifier overflow [89]: a server starts a new thread upon a new query from a client while using fixed space for thread identifiers. After the server runs for a sufficiently long time, the thread-identifier variable overflows, wreaking havoc. Modeling dynamic thread creation in a static-threaded finite-state program would have exposed the issue.) The third area is the modeling of Edge Computing, in which parallel computations (e.g., in the cloud) are given to a variable number of small-size computational nodes (for instance, embedded and mobile devices) which access a single server. For the purpose of modeling, the small nodes can be viewed as threads, the server can be viewed as shared memory, and various types of message passing can be replaced with the interleaving shared-memory semantics.

To proceed, let us introduce some terminology. Intuitively, a *program state* is a valuation of all the variables of the program, including the control-flow counters of all the threads. The *distance* from a program state s to a program state s' is the minimal number of program steps needed to reach s' from s along an execution (or ∞ if s' is unreachable from s). The *diameter* of a program is the maximal finite distance present in the program. If a bug finder outputs an error trace of the program analyzed, this trace is, in the worst case, at least as long as the program diameter. So the diameter of a program is a lower bound on the worst-case running-time of a bug finder on this program. At the same time, the diameter is equal to the number of steps that an ideal search (i.e., a search equipped with an oracle for the exact heuristic) would take to travel from a source program state to a target program state if these states are furthest apart but still connected. So the diameter is an upper bound on the running time for a successful, ideal search, in which the bug finder would always choose the right walk. These lower and upper bounds also apply to the searches in program models (and not only in original programs), e.g., models produced by predicate abstraction or in the inner loops of counterexample-guided abstraction refinement (CEGAR) schemes.

We are interested in the worst-case diameter among all the programs with the same number of threads (recall that the sizes of shared and local memory are fixed; to simplify our analysis, we assume that all threads have equally many local states). Thus, we concentrate on the function that, given a natural number n, returns the largest diameter over all programs with n threads. We call this function diamax. To the best of our knowledge, nothing is known about this function yet (except [60, 62, 64]). Certainly, diamax is majorized by the size of the state space, which is singly exponential in n. In a more general context, the conventional wisdom so far has been that the dependence between the worst-case distances in the transition graph of a (not necessarily multithreaded) program and the number of the components of the program (whether these are variables or threads) is exponential. In place of accepting this informal, conventional piece of wisdom, we prove a (much better) linear lower bound and a polynomial upper bound for multithreaded programs. Further, we demonstrate a stronger, linear upper bound for a certain subclass of programs. Moreover, we prove that, for a rather general class of probability distributions, the diameter of a random program is asymptotically almost surely at most linear. We also show that the program-state–to–program-state (non-) reachability problem belongs to the complexity class $\mathsf{NSpace}(\log n)$.

As the above notion of the (maximal) diameter is based on the program-state–to–program-state distance, this notion targets "nonlocal" properties concerning more than one thread, such as deadlock freedom or mutual exclusion. Still, many interesting nonreachability properties (of, for instance, real operating-system code) are "local," meaning that they are, simply put, of the following form: "a particular state of a given thread does not occur in any execution" (hereinafter, a state of a thread, shortly, a *thread state*, stands for a valuation of all the variables that the thread can access directly, including its control-flow counter). Such a property could, e.g., be specified by an `assert` statement of the programming language C (such a statement can refer to the shared and the thread-own variables). Moreover, program transformations and modeling may turn local properties into nonlocal ones or vice versa, sometimes producing intricate objects, e.g., internal models generated by automatic CEGAR loops.

Therefore, we also consider a related notion, the so-called *local diameter*. Roughly speaking, the local diameter of a program is the length of the shortest counterexample to any of the worst (i.e., hardest to refute, but still refutable) local safety properties. (A formal definition appears in § II.2.) If a bug finder outputs an error trace to a thread state, this output is, in the worst case, at least as long as the local diameter of the program. So the local diameter of a program is a lower bound on the worst-case time for finding local bugs in that program. At the same time, the local diameter is the number of steps that an ideal search (i.e., a search equipped with an oracle for the exact heuristic) would take to arrive from a source program state at a target thread state of a target thread in the worst case, i.e., maximizing over all triples (source program state, target thread, target thread state). So the local diameter is an upper bound on the running time for the successful, ideal thread-state search in which the bug finder always chooses the fastest walk. Again, these lower and upper bounds also apply to searches for local bugs in program models (and not only in original programs), e.g., the models produced by predicate abstraction or in the inner loops of the CEGAR schemes.

We show that the maximum local diameter for n-threaded programs is bounded above by a value independent of n, and that the least upper bound can be explicitly constructed (meaning, we can actually write, to any level of detail, an algorithm computing the least upper bound). Though it is possible to derive the boundedness through a careful interpretation and extension of results in the literature (cf. § I), explicit computability is not immediate. In contrast to this, we propose a mostly self-contained proof in § IV, parts of which are general enough to be potentially reusable in other contexts. Moreover, we show that the program-state–to–thread-state (non-)reachability problem belongs to the complexity class NC^1 [81, Definition 10.38] (again, considering the shared and thread-local memories bounded).

The fact that the lower bound on the diameter asymptotically exceeds the upper bound on the local diameter is, to the best of our knowledge, among the strongest non-algorithmic, asymptotic, formal arguments supporting the conventional wisdom that local safety properties are easier to deal with than nonlocal ones.

Summarizing, our *major contributions* are as follows:

- The definitions of the diameter and the local diameter of a multithreaded program (extending [64]).
- Constructively bounding the maximum local diameter from above for the parametrized case by an explicitly computable value independent of n (Definition and Corollary IV.3.8).
- Bounding diamax between a linear and a polynomial function in general (Theorems V.1.3 and V.2.1.22). The polynomial is of lower degree than previously known ones (Note V.2.1.23).
- A class of programs for which we show a linear upper bound in n on the diameter (Theorem V.2.2.3), and which matches the lower bound in certain cases (Note V.2.2.5).
- For rather general probability distributions on thread transitions, the diameter of a program is asymptotically almost surely at most linear in n (Theorem V.2.3.2).
- Deciding whether a program state is reachable from another program state is possible in $\mathsf{NSpace}(\log n)$ (Theorem VI.1).
- Deciding whether a state of a specified thread is reachable from a program state is possible (assuming that n is the only variable parameter) in NC^1 (Theorem VI.2).

The picture would be imperfect without reviewing a contrasting result in our context: (non-)reachability for finite-state interleaving-semantics multithreaded programs is PSpace-complete in the whole input length, i.e., for the unparametrized case, with respect to logspace reductions (Corollary II.3.3).

We conclude by discussing the growth of the diamax function (§ VII), including the theoretical benefits of its low growth to verification and bug finding.

Limitations

First, it is not the goal of this paper to empirically measure or directly improve contemporary techniques for verification or finding bugs. Rather, the paper contributes to the *classification* of the asymptotic complexity of search, with and without an oracle for the exact heuristic, in the parametrized setting in which the contribution of the variable number of threads n is singled out. For this purpose, we measure the distances in the transition graphs and use traditional complexity classes. (The classification in the parametrized-complexity class hierarchy is a separate topic by itself and is thus relegated to a later report.) Since we are interested in the dependency on n, the expressions that do not depend on n ("constants," especially those hidden in the asymptotic notation) are of minor importance here. Still, we sometimes get these constants for free and, as a service to the interested reader, we track or even optimize them if the corresponding proof methods easily permit this. Whether these constants are "optimal enough" (which would depend on the purpose and is often subjective) is not essential for our classification efforts: tooling and practicable techniques are orthogonal to the goals of this paper.

Second, it is not our intention to consider program families created from thread templates for which the sizes of shared and thread-local state spaces depend on parameters such as the number of threads n (e.g., [87]). For thread-template–based families, there is no standard dependency of the sizes of shared and thread-local state spaces on n (depending on the example, the dependency may not exist [65, §§ 7, 8] or be anywhere between linear [66, Example 13] and, for instance, $n^{\mathsf{O}(n)}$ [92]). Moreover, some parametrized programs (e.g., Readers-Writers) come with two or more independent variable parameters which together determine n and the sizes of shared and local state spaces. Investigations of such families would necessarily be more family-specific; results obtained for one family may not transfer to others. In contrast, this paper aims to deal with only one variable parameter and generically with the whole class of multithreaded programs rather than with particular families of multithreaded programs.

Third, the main purpose of §§ V.2.2 and V.2.3 is not large applicability areas (though we mention certain application scenarios in the two sections). The main purpose is, rather, a demonstration that the possibility of the existence of an elementary, explicit, and low-degree polynomial, even a linear one, as a general upper bound on diamax cannot be excluded.

I. Related work

In the narrowest sense, the complexity of deciding reachability in finite-state shared-memory multithreaded programs equipped with interleaving semantics has to the best of our knowledge not yet been determined with a formal proof. The (presumably) closest-related result published with a formal proof is the PSpace-completeness of the emptiness of the intersection of regular languages given by finite automata [51, Lemma 3.2.3]. The author claimed the relevance of the result on regular-language intersection as early as 2006 (cf. [66, 67]) and stated the PSpace-completeness of deciding reachability in multithreaded programs in 2010 (cf. [59]) without proof. Some other researchers mentioned vaguely the same result without proof (we are aware only of later claims, e.g., [11, p. 4]) or stated related results with proofs (e.g., on LTL for concurrent programs [82]).

As for singling out the contribution of the number of threads n, [58, p. 126] already observed that "symmetrization reduces the growth of [the description of the transition graph] from exponential with respect to [n] to polynomial with respect to [n]." To our knowledge, the polynomial degree has never been determined precisely in general; that is where we step in. In the special case that all the threads of a program execute the same code, it has been observed [9, 10, 28] that the symmetrized program has roughly n^L program states, where L is the number of the local states.

In a wider sense, a concise review of the reachability in parametrized concurrent programs based on only one template is given in [29, § 3.2]. Relaxing the communication model leads us to [20, 83, 84] (for rendezvous communication) or [13] (for token ring) among replicated processes.

Our results on local reachability are grounded in the finite-basis and Petri-net theories [39, 48]. The set of programs can be seen as a single well-structured transition system [1], yielding the decidability of thread-state reachability via a constant bound on the distances to a target state. However, [1] fixes the target state (not the initial state) and is formulated abstractly rather than in terms of shared-memory multithreaded programs equipped with interleaving semantics; therefore, additional work has to be invested in order to prove that the bound does not depend on the state we commence with (whether initial or target; both depend on n themselves), to transfer their results into the local-diameter graph-theoretic terminology for programs, and to prove the explicit computability of the local diameter.

Related practical reachability deciders employ symmetry reductions, which work well if $n>L$, and counter abstractions of various kinds. Symmetrization is possible, e.g., in the context of plain search [26] or in combination with a CEGAR loop and context inference [38]. Introducing counters is the main way symmetrization is implemented; sometimes the counter is abstracted to reduce the size of the state space [72]. For parametrized programs, cutoff techniques are applied. For example, [27] considers a computation model in which multiple copies of multiple process templates are concurrently executed, the shared state is absent, but each process executes specifically guarded transitions of restricted forms depending on the local state of other processes. Dynamically detecting cutoffs for identical copies of a single thread template starting in a fixed set of initial states is possible [47]; searching in the parallel composition of infinitely many copies can be reduced to searching in the parallel composition of a finite number of copies. The authors of [47] do not generalize their technique to copies of more than one template in their paper but mention that such generalizations are possible. A (non–counterexample-based) abstraction refinement for parametrized multithreaded programs consisting of copies of one thread is described in [2]. Assuming a bounded number of shared and local states as well as thread-local error states, we conjecture that the number of refinements from [2, Theorem 1] and the cutoff from [47] can roughly be bounded from above by the maximum local diameter. Searching in multithreaded programs can also be reduced to Petri nets [93, p. 14].

In general, all practical reachability deciders face the state-space–explosion problem (in the sense that the state space blows up exponentially with the number of threads); they exhaust space or time limits in a tool-dependent way or return an inconclusive answer. (However, on a set of carefully chosen templates, the tools performing symmetrization sometimes achieve exponential reductions.) We see claims such as, "The main obstacle to finite-state verification of concurrent systems is the state explosion problem: the number of states a concurrent system can reach is, in general, exponential in the number of concurrent processes in the system" [80] and "For fundamental reasons, we cannot avoid the exponential explosion in the number of threads" [32] spread throughout research-level texts. Such claims, taken literally, lead to asking whether the variable number of threads is in the exponent of a mathematical expression related to the computational difficulty of a reachability problem, rather than to the running time of a particular tool or to the size of a particular representation of the state space. This question is resolved in this paper in the negative, thereby positively resolving Open Problem 1.12.3 in [59]. We also show that several other straightforward mathematical formulations of the above quotations are all embarrassingly wrong: in these formulations, n is *never* in the exponent of the expressions obtained.

We mention five works which are closest to this paper. The first of these is [64], which handles most topics of the present paper for the special case of binary programs (a *binary* program is a program with two shared states and two local states). The present paper eliminates the restriction to binary programs, considering *all* finite-state multithreaded programs equipped with interleaving semantics. Further, all results in this paper, when restricted to the binary case, are at least as strong as, or stronger than, those of [64]. The second work is [63], which is [64] with all proofs. Being a generalization, our paper intentionally follows [63, 64] concerning the structure, terminology, and certain examples and claims. The reader might find similar formulations which are, unless otherwise stated, to be interpreted here in the parametrized context rather than in the binary one.

The third work, [60], is the conference version of this very paper and its direct precursor. The present paper enhances and sharpens [60]:

- This paper contains all low-level details and background material, whereas [60] contains a high-level, abridged presentation.
- We prove the PSpace-completeness of reachability in the unparametrized case w.r.t. the logspace reductions in Corollary II.3.3, whereas [60] only refers to [51, Lemma 3.2.3].
- We tighten the upper bound on diamax in Theorem V.2.1.22 by improving the degree by the factor of L compared to [60, Thm. IV.2.1.10] (Theorem IV.2.1.13 in the technical report accompanying [60]).
- We strengthen the upper bound on the diameter of programs from the special class in § V.2.2 compared to [60, § IV.2.2].
- We slightly extend the probabilistic analysis in § V.2.3 compared to [60, § IV.2.3].

The fourth work is an ongoing experiment [62] which motivates the research presented.

The fifth work, [61], is a short, higher-level version of this very paper containing proof ideas rather than proof details.

Broadly speaking, our solutions have been partially inspired and influenced by the insights from symmetry reduction [28], process replication [84], finite-base arguments [39], vector addition systems and Petri nets [55], counter abstraction [72], low-complexity arguments in case the shared memory cannot read-and-write as a single atomic operation [30], pattern-based verification [31], linear interfaces [52], and other research directions [6, 15, 16, 24, 25, 71].

Outside formal methods, the diameter is sometimes defined as the maximal distance, being infinity for not strongly-connected graphs [90], whereas we consider the maximal *finite* distance, since the transition graphs need not be strongly connected. Defined more widely, the notion of graph diameter is important in, e.g., [3] (standard algorithms on graphs), [17] (an old survey on diameter-related problems mostly for undirected graphs), [18] (spectral graph theory, restricted to strongly connected graphs), [22] (strongly connected Eulerian directed graphs without 2-cycles), [54] (networks), [0, 43] (centrality computation), and [23] (the best-case performance of the simplex method in linear programming).

II. Preliminaries

We now introduce the formal notation used throughout the paper.

As an aid for the reader, we sometimes put an exclamation mark above the relation sign of a claim that yet has to be proven. For instance, $X \overset{!}{\subseteq} Y$ means: we claim $X \subseteq Y$ and proceed to prove it.

We write $\lightning$ to indicate a contradiction.

In a Boolean formula, the underscore _ denotes an innermost existentially quantified anonymous variable; different underscores correspond to different variables, e.g., "$\varphi(\_, 1) \wedge \psi(\_, 0)$" means "$(\exists x\colon \varphi(x, 1)) \wedge (\exists x\colon \psi(x, 0))$." Similarly, in textual claims that are not formally typeset as formulas, an underscore denotes a value that does not have to be specified in the context. For example, "the triple is of the form $(0, \_, \_)$" means "there are l_1 and l_2 such that the triple is $(0, l_1, l_2)$."

We write $\mathbb{N}_+$ for the set of positive integers, $\mathbb{N}_{\geqslant 0}$ for the set of nonnegative integers, $\mathbb{Q}$ for the set of rationals, and $\mathbb{Q}_{\geqslant 0}$ for the set of nonnegative rationals. Unless otherwise stated, the implicit universe of variables is $\mathbb{N}_{\geqslant 0}$. To simplify the notation, we view natural numbers as ordinals, so $\forall\, i, j\colon i{<}j \Leftrightarrow i{\in}j$. (We thus escape additional notation for the set of thread identifiers such as Tid often seen in the literature [33]: Tid gets unnecessary, since the number of threads n can be viewed as the set of thread identifiers $\{0, \ldots, n{-}1\}$. Moreover, formulas such as "$\forall\, j \in n \setminus \{i\}\colon \ldots$" are simpler than "$\forall\, j \in \mathbb{N}_{\geqslant 0}\colon (j < n \wedge j \neq i) \Rightarrow \ldots$.")

Our map-constructor is right-associative, meaning that $X{\to}Y{\to}Z$ (where X, Y, and Z are variables representing arbitrary sets) is read as $X{\to}(Y{\to}Z)$, which is the set of functions mapping each element of X to some function from $Y{\to}Z$. Maps are sometimes written in λ-notation [8]; e.g., $\lambda\, x{\in}X.\, \lambda\, y{\in}Y.\, z$ is a particular element of $X{\to}Y{\to}Z$ assuming $z{\in}Z$. We write $X{\dashrightarrow}Y$ for the set of partial maps from X to Y, $X{\hookrightarrow}Y$ for the set of injective (in other terminology, one-to-one) maps from X to Y, $X{\twoheadrightarrow}Y$ for the set of surjective (in other terminology, onto) maps from X to Y, and $X{\hookrightarrow\!\!\!\!\rightarrow}Y$ for the set of bijections (in other terminology, one-to-one correspondences) from X to Y. The inverse of a bijection f is written as f^{-1}.

Given a partial map $f\colon X \dashrightarrow Y$, we write

$$\operatorname{dom} f \overset{\text{def}}{=\!=} \{x \in X \mid \exists\, y{\in}Y\colon (x, y) \in f\} \quad\text{and}\quad \operatorname{img} f \overset{\text{def}}{=\!=} \{y \in Y \mid \exists\, x{\in}X\colon (x, y) \in f\}$$

for the *domain* and *image* of f, respectively.

Given a map f and some elements a, b, we write $f[a{\mapsto}b]$ for the map which returns the value b for the argument a and behaves like f for all other arguments. Formally:

$$f[a{\mapsto}b] \overset{\text{def}}{=\!=} \lambda\, x \in (\operatorname{dom} f) \cup \{a\}. \begin{cases} f(x)\,, & \text{if } x \neq a\,, \\ b\,, & \text{if } x = a\,. \end{cases}$$

For finite functions mapping to some set of numbers (naturals, rationals, ...), $\|\cdot\|_1$ denotes the 1-norm and $\|\cdot\|_\infty$ the maximum norm: $\|f\|_1 \overset{\text{def}}{=} \sum_{i \in \operatorname{dom} f} |f(i)|$ and $\|f\|_\infty \overset{\text{def}}{=} \max\{|f(i)| \mid i \in \operatorname{dom} f\}$, where $\max \emptyset \overset{\text{def}}{=} 0$.

For a sequence σ with an index set I and $i{\in}I$, we mostly use the right subscript or the right superscript in square brackets to write the i^{th} element as σ_i or $\sigma^{[i]}$; the whole sequence is written as $(\sigma_i)_{i \in I}$ or $(\sigma^{[i]})_{i \in I}$, respectively. If the index set is some initial segment of natural numbers, we may write $i{<}n$ or $i{\leqslant}n$ instead of $i{\in}n$ or $i \in n{+}1$ in the right subscript position of the closing paren; the version with the weak inequality "$\leqslant$" additionally implies the nonemptiness of σ. If the index set of a sequence σ is a product $I{\times}J$ for some sets I and J, we sometimes opt for double indexing, writing the $(i, j)^{\text{th}}$

element $\sigma_{(i,j)}$ as $\sigma_i^{[j]}$ and $\left(\sigma_i^{[j]}\right)_{\substack{i\in I\\ j\in J}}$ for σ. The context determines which notation is most convenient.

The *cardinality* of a sequence is the number of elements in it:

$$\left|(s_i)_{i<k}\right| \quad \stackrel{\text{def}}{=} \quad k\,.$$

For an ordinal n and a set X, the set of n-tuples over X is denoted by X^n, which is simply another expression for $n{\to}X$. Thus we maintain the convention that the indexes of the components of a tuple start with 0.

By a slight abuse of notation, a plain number in the right superscript position of a symbol denotes the power of that symbol (where the multiplication operation is understood from the context), e.g., $3^2 = 9$ or $X^3 = (X{\times}X{\times}X)$.

For a set X, we write $\mathsf{id}_X \stackrel{\text{def}}{=} \{(x,x) \mid x{\in}X\}$ for the identity relation on X.

Given binary relations $f \subseteq X{\times}Y$ and $g \subseteq Y{\times}Z$, we write $g \circ f$ to denote the *right composition* "g after f," which is the relation

$$g \circ f \;\stackrel{\text{def}}{=}\; \{(x,z) \mid \exists\, y\colon\, (x,y) \in f \,\wedge\, (y,z) \in g\}\,.$$

"Right" refers to the fact that the right symbol is applied first. In a context where f and g are even maps, $g \circ f$ is called the *functional composition* of g and f.

If $\to$ is a binary relation, we write $\to^*$ for its reflexive-transitive closure (on a set taken from the context).

A *preorder* on a set X is a binary relation $\lesssim$ on X such that $\lesssim$ is reflexive on X and transitive. A *preordered set* is a pair $(X, \lesssim)$ of a set X and a preorder $\lesssim$ on X. A function $f\colon X{\to}X$ on a preordered set $(X, \lesssim)$ is *antitone* iff $\forall\, x,y \in X\colon x{\lesssim}y \;\Rightarrow\; f(y) \lesssim f(x)$.

Given an equivalence relation $\sim$ on a set X, we write

$$[x]_\sim \;\stackrel{\text{def}}{=}\; \{y \in X \mid x \sim y\}$$

for the equivalence class of x with respect to $\sim$ ($x \in X$) and ${}^{X}\!/_{\sim}$ for the set of equivalence classes. The *index of an equivalence relation* is the number of the equivalence classes.

The term $\log x$ denotes the logarithm of x in base 2.

II.2. Program notation

For $n \in \mathbb{N}_+$, an (n-threaded) *program* is a tuple

$$p \;=\; (\mathsf{Glob}, \mathsf{Loc}, \to_0, \ldots, \to_{n-1}) \tag{1}$$

such that Glob and Loc are finite nonempty sets and

$$\forall\, i{<}n\colon \quad \to_i \;\subseteq\; (\mathsf{Glob} \times \mathsf{Loc})^2\,.$$

(Adapted from [34].) Such a program is called *binary* iff $|\mathsf{Glob}| = |\mathsf{Loc}| = 2$. Elements of Glob are called *shared states* (also called *global states*), elements of Loc *local states*, elements of $\mathsf{Glob}{\times}\mathsf{Loc}$ *thread states*, elements of $\bigcup_{i<n}{\to_i}$ *thread transitions*. A *program state* is an element of

$$\mathsf{State} \quad \stackrel{\text{def}}{=} \quad \mathsf{Glob} \times \mathsf{Loc}^n\,.$$

For $(g, l) \in \mathsf{State}$, we call g its *shared part* and l_i its *i^{th} local part* ($i < n$). The *transition graph* induced by p is a directed graph $(\mathsf{State}, \longrightarrow)$ where

$$(g, l) \longrightarrow (g', l') \overset{\text{def}}{\iff} \exists\, i{<}n\colon \big((g, l_i) \to_i (g', l'_i) \wedge \forall\, j \in n \setminus \{i\}\colon l_j{=}l'_j\big)$$

for all $(g, l), (g', l') \in \mathsf{State}$. A *walk* in the graph is a sequence $(s_i)_{i<m}$ of states connected by program transitions, i.e., such that $s_i \longrightarrow s_{i+1}$ for all i with $i{+}1 < m$; a *path* is an injective walk, i.e., a walk $(s_i)_{i<m}$ satisfying $\forall\, i, j \in m\colon s_i{=}s_j \Rightarrow i{=}j$ (adapted from [54, 90]). The *length* of a nonempty walk is the number of times the walk takes an edge of the transition graph:

$$\mathsf{length}\big((s_i)_{i\leqslant m}\big) \overset{\text{def}}{=} m\ ,$$

where "$\leqslant$" in the subscript imposes an implicit requirement that the walk is nonempty, containing s_0. The *distance* from a program state s to a program state s' in the transition graph is the length of a shortest walk from s to s' (or infinity, if s' is unreachable from s):

$$\mathsf{d}(s, s') \overset{\text{def}}{=} \min\big\{m \;\big|\; \exists\,\text{walk } (s_0, \ldots, s_m) \text{ in } (\mathsf{State}, \longrightarrow) \text{ such that } s_0{=}s \wedge s_m{=}s'\big\},$$

where $\min \emptyset \overset{\text{def}}{=} \infty$. The *local distance* from a program state s to a thread state (g, a) of a thread i in the transition graph is the length of the shortest walk from s to a program state with local part a of thread i and shared part g (or infinity, if no such walk exists):

$$\mathsf{d}^{\mathsf{loc}}(s, i, (g, a)) \overset{\text{def}}{=} \min\left\{m \;\middle|\; \begin{aligned} \exists\, l \in \mathsf{Loc}^n,\ \text{walk } (s_0, \ldots, s_m) \text{ in } (\mathsf{State}, \longrightarrow)\colon \\ l_i = a \;\wedge\; s_0 = s \;\wedge\; s_m = (g, l) \end{aligned}\right\},$$

where again $\min \emptyset \overset{\text{def}}{=} \infty$.

From now on, if the program referred to in the above definitions is unclear from the context, we add a right subscript to specify the program; e.g., $\mathsf{d}_p(\ldots)$ is the distance in program p, and $\mathsf{d}^{\mathsf{loc}}_p(\ldots)$ is the local distance in program p.

II.3. PSpace-completeness of (unparametrized) reachability w.r.t. logspace reductions

Now we are going to show that reachability of program states and thread states in a multithreaded program (in the full, unparametrized case) is PSpace-complete with respect to logspace reductions.

(For the sake of clarity: no proofs of the PSpace-completeness of the above problems have been published so far to the best of our knowledge, but similar results have become folklore (cf. § I) at least as far as the schoolbook definition of PSpace-completeness via polynomial-time reductions (cf. [42, § 11] and [81, § 8.3]) is concerned. Therefore, § II.3 is merely provided for purpose of the logical consistency of the paper as a service to the reader, rather than as our outstanding contribution, and may be skipped without ill effects on first reading.)

We will provide direct and full-fledged proofs. Though one way to prove this would be to reuse [51, Lemma 3.2.3], we do not do this, since Kozen leaves checking that the reduction is logspace to the reader, and further, reusing Kozen's result would provide no significant savings.

For the purpose of defining the aforementioned reachability problems as formal languages and for the sake of simplicity, we make, without loss of generality, the following pretty ordinary assumptions within this section:

- In a program, the set of shared states is a subset of $\mathbb{N}_{\geqslant 0}$.

- In a program, the set of local states is a subset of $\mathbb{N}_{\geqslant 0}$.
- A natural number is encoded in binary.
- A finite set of items (e.g., a thread transition relation) is stored as a list of items (in some order) between the braces { and }.
- The separator inside a list of items is the usual comma ,.
- A tuple is stored as a list between parens ().

In particular, we let the alphabet Σ contain 0, 1, a comma, a left paren, a right paren, a left brace, and a right brace. We expect that minor variations (e.g., making the shared states and local states tuples of numbers instead of numbers) would not change the complexity result.

Let

$$\begin{aligned}
\mathsf{Reach} &\stackrel{\text{def}}{=} \left\{(p, s, s') \mid p \text{ is a program } \wedge\ s, s' \in \mathsf{State}_p \ \wedge\ \mathsf{d}_p(s, s') < \infty\right\}, \\
\mathsf{NonReach} &\stackrel{\text{def}}{=} \left\{(p, s, s') \mid p \text{ is a program } \wedge\ s, s' \in \mathsf{State}_p \ \wedge\ \mathsf{d}_p(s, s') = \infty\right\}, \\
\mathsf{Reach}^{\mathsf{loc}} &\stackrel{\text{def}}{=} \left\{(p, s, i, \tau) \,\middle|\, \begin{array}{l} p \text{ is a program } \wedge\ s \in \mathsf{State}_p \ \wedge\ i{<}n \\ \wedge\ \tau \in \mathsf{Glob}{\times}\mathsf{Loc} \ \wedge\ \mathsf{d}^{\mathsf{loc}}_p(s, i, \tau) < \infty \end{array}\right\}, \quad \text{and} \\
\mathsf{NonReach}^{\mathsf{loc}} &\stackrel{\text{def}}{=} \left\{(p, s, i, \tau) \,\middle|\, \begin{array}{l} p \text{ is a program } \wedge\ s \in \mathsf{State}_p \ \wedge\ i{<}n \\ \wedge\ \tau \in \mathsf{Glob}{\times}\mathsf{Loc} \ \wedge\ \mathsf{d}^{\mathsf{loc}}_p(s, i, \tau) = \infty \end{array}\right\}.
\end{aligned}$$

First, we present an upper bound:

Lemma II.3.1. $\mathsf{Reach}, \mathsf{NonReach}, \mathsf{Reach}^{\mathsf{loc}}, \mathsf{NonReach}^{\mathsf{loc}} \in \mathsf{PSpace}$.

Proof. A decider of any of the four problems initially performs the corresponding syntax check and, if the input does not have the required form, stops. So, from now on, let us assume that the syntax is valid.

For Reach and $\mathsf{Reach}^{\mathsf{loc}}$, the decider performs a nondeterministic search in the program transition graph, starting with the given source program state and nondeterministically choosing a successor at each step. Within this search, the decider maintains storage for only the current state and the auxiliary counters used for navigating within the input; the search history should not be remembered. For Reach, the decider accepts when the target program state is encountered. For $\mathsf{Reach}^{\mathsf{loc}}$, the decider accepts as soon as the target thread state of the target thread is seen.

For NonReach (or $\mathsf{NonReach}^{\mathsf{loc}}$, respectively), the decider applies Savitch's procedure [77] to search for the target program state (or the target thread state of the target thread, respectively). After the search is over, the decider accepts if no target state has been found. ■

Now we present a lower bound:

Lemma II.3.2. Reach*,* NonReach*,* $\mathsf{Reach}^{\mathsf{loc}}$*, and* $\mathsf{NonReach}^{\mathsf{loc}}$ *are PSpace-hard with respect to logspace reductions.*

Proof. We start with Reach.

Let L (without loss of generality, over the same alphabet Σ) be a language decided by a Turing machine T in space bounded above by a polynomial function. Choose a polynomial upper bound p that is logspace-computable and satisfies $p(x) \geqslant 2$ for all $x \in \mathbb{N}_{\geqslant 0}$, e.g., $p = \lambda x \in \mathbb{N}_{\geqslant 0}.\, (\min\{2^{c_1} \mid 2^{c_1} \geqslant x\})^{2^{c_2}}$ for constants $c_1, c_2 \in \mathbb{N}_+$. We provide a logspace reduction from L to Reach.

We assume that the machine T consists of the following components:

- Q: the set of states.
- $q_0 \in Q$: the starting state.
- $q_{\text{accept}} \in Q$: the accepting state.
- $\sqcup \notin \Sigma$: the blank symbol.
- $\Gamma \supseteq \Sigma \mathbin{\dot\cup} \{\sqcup\}$: the tape alphabet.
- $\delta\colon (Q \setminus \{q_{\text{accept}}\}) \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$: the transition function.

Without loss of generality, we assume further that T works on a semi-infinite tape open to the right [81, p. 168]; in particular, when T tries to move the head to the left of the leftmost cell, the symbol and the control state change as usual, but the head stays at the leftmost cell (though δ indicates L). We also assume, without loss of generality, that T erases the tape right before acceptance from right to left and leaves the head above the leftmost cell (so that the accepting configuration is unique).

The reduction maps a string w to a triple (P, s, s') of a program P and two of its states s and s' such that the transition graph of P has a walk from s to s' iff T accepts w. The program P will simulate T within bounded space, s will correspond to the initial configuration, and s' will correspond to the final configuration.

Given w, the reduction would output the following program, in which the i^{th} thread simulates changes to the i^{th} cell on the tape ($i{<}n$), enumerated from 0:

- $n = p(|w|)$,
- $\mathsf{Glob} = Q \times n$,
- $\mathsf{Loc} = \Gamma$,
- $\to_0 = \Big\{\big(((q,0),\gamma),((q',0),\gamma')\big) \;\Big|\; \delta(q,\gamma) = (q',\gamma',\mathrm{L})\Big\} \cup \Big\{\big(((q,0),\gamma),((q',1),\gamma')\big) \;\Big|\; \delta(q,\gamma) = (q',\gamma',\mathrm{R})\Big\}$,
- $\to_i = \Big\{\big(((q,i),\gamma),((q',i{-}1),\gamma')\big) \;\Big|\; \delta(q,\gamma) = (q',\gamma',\mathrm{L})\Big\} \cup \Big\{\big(((q,i),\gamma),((q',i{+}1),\gamma')\big) \;\Big|\; \delta(q,\gamma) = (q',\gamma',\mathrm{R})\Big\}$ for each positive $i < n{-}1$,
- $\to_{n-1} = \Big\{\big(((q,n{-}1),\gamma),((q',n{-}2),\gamma')\big) \;\Big|\; \delta(q,\gamma) = (q',\gamma',\mathrm{L})\Big\}$,
- $P = (\mathsf{Glob}, \mathsf{Loc}, (\to_i)_{i<n})$.

Let the source program state be $s = ((q_0, 0), w\underbrace{\sqcup\cdots\sqcup}_{n-|w|\text{ times}})$ and the target program state be $s' = ((q_{\text{accept}}, 0), \underbrace{\sqcup\cdots\sqcup}_{n\text{ times}})$. (Here, the string characters are indexed from the left, starting with 0.)

A run of T from the initial configuration $q_0 w \sqcup\sqcup \cdots$ to the final configuration $q_{\text{accept}} \sqcup\sqcup \cdots$ is simulated by a walk in the transition graph of P from s to s', and vice versa. The corresponding bijection maps a configuration of T of the form vqv' to the program state $((q, |v|), vv'\underbrace{\sqcup\cdots\sqcup}_{n-|vv'|\text{ times}})$ ($q \in Q$, $v, v' \in \Gamma^*$ such that $|vv'| < n$).

Computing the reduction starts with computing $|w|$ on the working tape, computing n, $n{-}1$, and $n{-}2$ on the working tape, and printing $\Big((Q{\times}n, \Gamma, (\rightarrow_0, \dots, \rightarrow_{n-1})), ((q_0, 0), w\underbrace{\sqcup\cdots\sqcup}_{n-|w|\text{ times}}), ((q_{\text{accept}}, 0), \underbrace{\sqcup\cdots\sqcup}_{n\text{ times}})\Big)$ to the output tape as follows:

- Encoding a shared state $(q, i) \in Q{\times}n$ as a natural number can be done by first viewing q as a natural number (which is trivial, since Q is fixed) and then computing $qn + i$, which takes $\mathsf{O}(\log n)$ space (again, since Q is fixed). Printing $Q{\times}n$ requires, in addition to the aforementioned auxiliary space for converting a single shared state, one counter to iterate from 0 till $n{-}1$.

- We can view Γ as a set of natural numbers, so each local state is a binary constant. Printing Γ amounts to printing a constant string anyway.

- Printing the transition relations requires an additional counter j on the working tape to iterate through the members of the ordinal n and space to compute $j \pm 1$ whenever the result lies in the interval $[0, n{-}1]$.

- Printing the initial program state requires encoding the shared state as above, copying the input, and a counter to iterate from $|w|$ to $n - 1$.

- Printing the target program state requires encoding the shared state as above and a counter to iterate from 0 to n.

Altogether, such computations need $\mathsf{O}(\log n)$ space on the working tape.

For $\mathsf{Reach}^{\mathsf{loc}}$, we adapt the above procedure by outputting (P, s, i, τ) for the target thread $i = 0$ and the thread state $\tau = ((q_{\text{accept}}, 0), \sqcup)$ instead of outputting (P, s, s').

For $\mathsf{NonReach}$ and $\mathsf{NonReach}^{\mathsf{loc}}$, the above is applied to a polynomial-space–bounded Turing machine accepting the complement of L instead of T (such a machine exists due to PSpace=coPSpace). ■

We combine Lemmas II.3.1 and II.3.2:

Corollary II.3.3. Reach, $\mathsf{NonReach}$, $\mathsf{Reach}^{\mathsf{loc}}$, *and* $\mathsf{NonReach}^{\mathsf{loc}}$ *are PSpace-complete with respect to logspace reductions.* □

Since every logarithmic-space reduction is a polynomial-time reduction, this result implies PSpace-completeness with respect to the (slightly more schoolbook) polynomial-time reductions.

II.4. Diameter and local diameter

The *diameter* of a program is the largest finite distance realizable in the transition graph:

$$\mathsf{diam}(p) \stackrel{\text{def}}{=} \max\big((\operatorname{img} \mathsf{d}_p) \setminus \{\infty\}\big)\,.$$

The *local diameter* of a program is the largest finite local distance realizable in the program's transition graph:

$$\mathsf{diam}^{\mathsf{loc}}(p) \stackrel{\text{def}}{=} \max\big((\operatorname{img} \mathsf{d}_p^{\mathsf{loc}}) \setminus \{\infty\}\big)\,.$$

Since we are interested in the dependency of the measured quantities on n, we fix Glob, Loc, $G \overset{\text{def}}{=} |\mathsf{Glob}|$, and $L \overset{\text{def}}{=} |\mathsf{Loc}|$ for the rest of the paper, unless explicitly stated otherwise. We thus write programs such as p in (1) more shortly as

$$p \;=\; (\rightarrow_0, \ldots, \rightarrow_{n-1}) \quad \text{or, even more compactly,} \quad p \;=\; (\rightarrow_i)_{i<n}\,.$$

The maximal possible diameter for an n-threaded program is denoted by

$$\mathsf{diamax}(n) \;\overset{\text{def}}{=}\; \max\{\mathsf{diam}(p) \mid p \text{ is an } n\text{-threaded program}\}\,.$$

The maximal possible local diameter for an n-threaded program is denoted by

$$\mathsf{diamax}^{\mathsf{loc}}(n) \;\overset{\text{def}}{=}\; \max\{\mathsf{diam}^{\mathsf{loc}}(p) \mid p \text{ is an } n\text{-threaded program}\}.$$

We say that a program $h = (\rightsquigarrow_i)_{i<m}$ is a *subprogram* of a program $p = (\rightarrow_i)_{i<n}$ if there is an injective map $f\colon m \hookrightarrow n$ such that $\forall\, i<m\colon \rightsquigarrow_i \;=\; \rightarrow_{f(i)}$; every such map is called an *embedding* of h into p.

Lemma II.4.1. *The subprogram relation is a preorder on the set of programs.*

Proof. Let $\mathcal{P}$ be the set of programs. We show two claims about the subprogram relation.

Reflexivity on $\mathcal{P}$. The identity $\mathsf{id}_n\colon n \hookrightarrow n$ is injective. Moreover, for each $(\rightarrow_i)_{i<n} \in \mathcal{P}$ we have $\forall\, i<n\colon \rightarrow_i \;=\; \rightarrow_{\mathsf{id}_n(i)}$.

Transitivity. Let $(\rightsquigarrow_0, \ldots, \rightsquigarrow_{m-1})$ be a subprogram of $(\rightarrow_0, \ldots, \rightarrow_{n-1})$ and $(\rightarrow_0, \ldots, \rightarrow_{n-1})$ be a subprogram of $(\rightarrowtail_0, \ldots, \rightarrowtail_{k-1})$. Then there are injective maps $f\colon m \hookrightarrow n$ and $g\colon n \hookrightarrow k$ such that $\forall\, i<m\colon \rightsquigarrow_i \;=\; \rightarrow_{f(i)}$ and $\forall\, j<n\colon \rightarrow_j \;=\; \rightarrowtail_{g(j)}$. Then the map $g \circ f$ is injective, and for all $i<m$ we have $\rightsquigarrow_i \;=\; \rightarrow_{f(i)} \;=\; \rightarrowtail_{g(f(i))} \;=\; \rightarrowtail_{(g\circ f)(i)}$. ∎

III. Examples

Before turning to the main results of this paper, we present some small examples (mostly taken from [64]). For simplicity, we show a few binary n-threaded programs whose diameter is $\mathsf{diamax}(n)$ for $n \in \{1, 2, 3\}$.

We therefore give some names to the shared and local states, e.g., $\mathsf{Glob} = \{\alpha, \beta\}$ and $\mathsf{Loc} = \{0, 1\}$, where $\alpha \neq \beta$ are some literals. For brevity in the binary case, we sometimes omit commas and parens when writing thread or program states: we occasionally typeset thread states $(g, l) \in \mathsf{Glob}\times\mathsf{Loc}$ as gl and program states $(g, (l_0, \ldots, l_{n-1})) \in \mathsf{Glob}\times\mathsf{Loc}^n$ as strings $gl_0 \cdots l_{n-1}$.

The (local) diameters of following examples are all computed directly from the definitions; double-checking the numbers is an exercise for the reader.

III.1. $n = 1$

Consider the program with exactly one thread and transitions $\alpha 0 \rightarrow_0 \alpha 1$, $\alpha 1 \rightarrow_0 \beta 1$, and $\beta 1 \rightarrow_0 \beta 0$. The diameter and the local diameter of this program are both 3. The transition graph of the program is depicted below right.

$$\begin{tikzcd} \alpha 0 \arrow[r] & \alpha 1 \arrow[d] \\ \beta 0 & \beta 1 \arrow[l] \end{tikzcd}$$

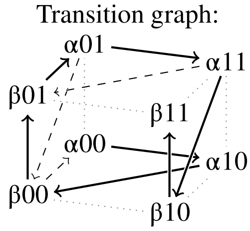
Since the total number of program states is 4, no single-threaded program can have a diameter larger than 3. In total, there are six programs that have diameter 3 and exactly three transitions and whose diameter-realizing paths start in $\alpha 0$.

III.2. $n = 2$

Now we depict some two-threaded programs of maximal diameter.

In the following tables, we omit the index at the arrows in thread transitions. In each transition graph, solid arrows constitute a shortest path from α00 to a state at the largest distance, dashed arrows are program transitions that do not contribute to the path, and dotted gray lines simply help to visualize the set of states as a geometric cube.

A program with a total of 5 thread transitions is depicted below.

Thread 0:	Thread 1:	Transition graph:
$\begin{tikzcd} \alpha 0 \arrow[r] & \alpha 1 \arrow[dl] \\ \beta 0 \arrow[u] & \beta 1 \end{tikzcd}$	$\begin{tikzcd} \alpha 0 & \alpha 1 \arrow[dl] \\ \beta 0 \arrow[r] & \beta 1 \end{tikzcd}$	

It has diameter $7 = \mathsf{d}(\alpha 00, \beta 11)$, and, since the total number of states in any two-threaded program is 8, no two-threaded program can have a larger diameter. Omitting any thread transition would yield a program with a lower diameter. Adding copies of existing threads does not, in general, produce programs with a maximal diameter: $\mathsf{diam}(\to_0, \to_0, \to_1) = \mathsf{diam}(\to_0, \to_1, \to_1) = 8$ and $\mathsf{diam}(\to_0, \to_0, \to_1, \to_1) = 9$, which, as we will see, are less than the lower bound from Theorem V.1.3. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 0, \beta 1)$. Note that for each thread every thread state occurs twice as part of some program states (e.g., the thread state α1 of thread 1 occurs in α01 and α11). Thus, for any source program state, the two occurrences cannot both have distance 7 from this source: if any of these occurrences is reachable from the source, at least one of these occurrences must have distance not exceeding 6. Thus, no two-threaded program can have a local diameter exceeding 6.

We give names to programs for convenient referencing. Figures 1 and 2 depict some programs with a total of 6 and 7 thread transitions, respectively. These programs have diameter 7. Omitting any thread transition from these programs would yield programs with a lower diameter.

Adding thread transitions or copies of existing threads may change the diameter, but this is not always the case.

Let us consider some cases:

In N2T6A, adding $\alpha 0 \to_0 \beta 0$ shrinks the diameter. Instead, adding any combination of the thread transitions $\alpha 1 \to_0 \alpha 0$, $\beta 0 \to_0 \alpha 0$, and $\beta 1 \to_0 \alpha 0$ does not change the diameter. We have $\mathsf{diam}(\to_0, \to_0, \to_1) = 7$ and $\mathsf{diam}(\to_0, \to_1, \to_1) = 8 = \mathsf{diam}(\to_0, \to_0, \to_1, \to_1)$. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 1, \beta 1)$.

The transition graph of N2T6B is strongly connected. Adding any combination of the thread transitions $\beta 1 \to_0 \alpha 1$, $\alpha 1 \to_0 \alpha 0$, $\beta 1 \to_0 \alpha 0$, and $\beta 0 \to_0 \alpha 0$ would not change the diameter, but adding $\alpha 1 \to_0 \beta 0$ would shrink it. Duplicating the threads slightly increases the diameter: $\mathsf{diam}(\to_0, \to_0, \to_1) = \mathsf{diam}(\to_0, \to_1, \to_1) = 8$ and $\mathsf{diam}(\to_0, \to_0, \to_1, \to_1) = 9$. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 1, \beta 0)$.

In N2T6C, adding any combination of the transitions $\alpha 1 \to_0 \alpha 0$, $\beta 0 \to_0 \alpha 0$, and $\beta 1 \to_0 \alpha 0$ would shrink the diameter. Adding the thread transition $\beta 1 \to_0 \alpha 1$ would not change the diameter. We have $\mathsf{diam}(\to_0, \to_0, \to_1) = 7$, $\mathsf{diam}(\to_0, \to_1, \to_1) = 8$, and $\mathsf{diam}(\to_0, \to_0, \to_1, \to_1) = 9$. The local diameter is $5 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 1, \beta 0)$.

The transition graph of N2T7B is strongly connected. Three paths realize distance 7: $\alpha 00 \longrightarrow^* \alpha 01$ (denoted by thick arrows), $\alpha 01 \longrightarrow^* \beta 00$, and $\beta 01 \longrightarrow^* \alpha 00$. Adding

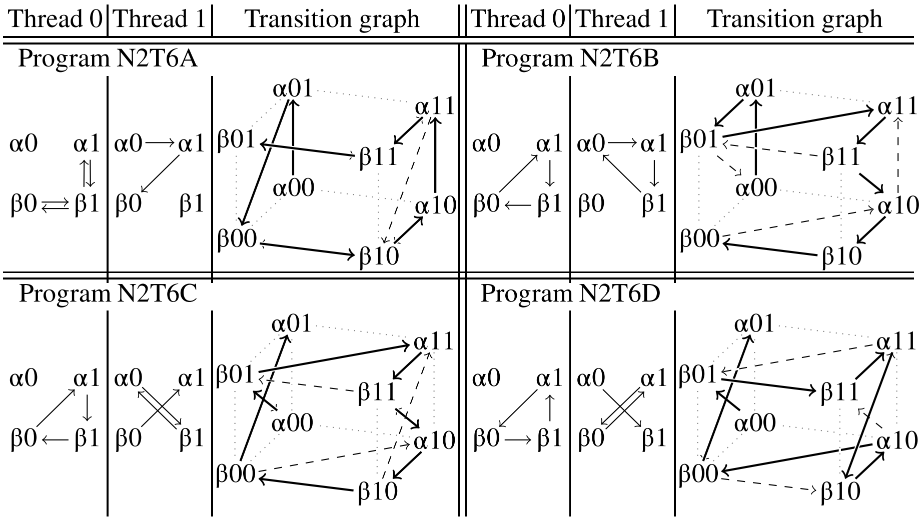


Figure 1: Some two-threaded binary programs with 6 thread transitions.

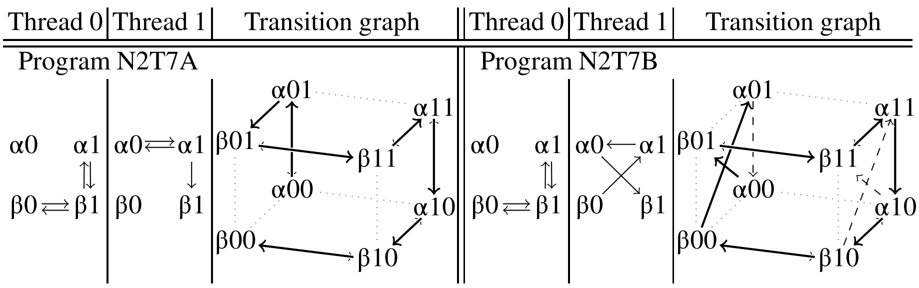


Figure 2: Some two-threaded binary programs with 7 transitions.

any combination of $\alpha 1 \rightarrow_0 \alpha 0$, $\beta 1 \rightarrow_0 \alpha 0$, and $\beta 0 \rightarrow_0 \alpha 0$ would retain the diameter 7, but the only path realizing the longest distance would be $\alpha 01 \longrightarrow^* \beta 00$. Instead, adding $\beta 0 \rightarrow_1 \beta 1$ would keep all three paths realizing the longest distance. Thread duplication increases the diameter in some cases: $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1) = 7$ and $\mathsf{diam}(\rightarrow_0, \rightarrow_1, \rightarrow_1) = 8 = \mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1, \rightarrow_1)$. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 01, 1, \beta 0)$.

III.3. $n = 3$

Consider the program N3T9 from Figure 3.

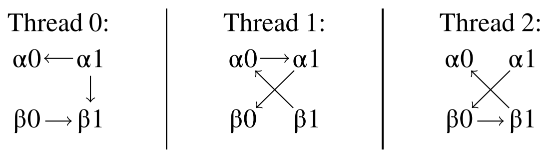


Figure 3: A three-threaded binary program.

Its transition graph is depicted in Figure 4.

This program has local diameter $9 = \mathsf{d}^{\mathsf{loc}}(\alpha 000, 2, \alpha 1)$ and diameter $13 = \mathsf{d}(\alpha 000, \beta 011)$. The transition graph is strongly connected. Adding a copy of each thread would

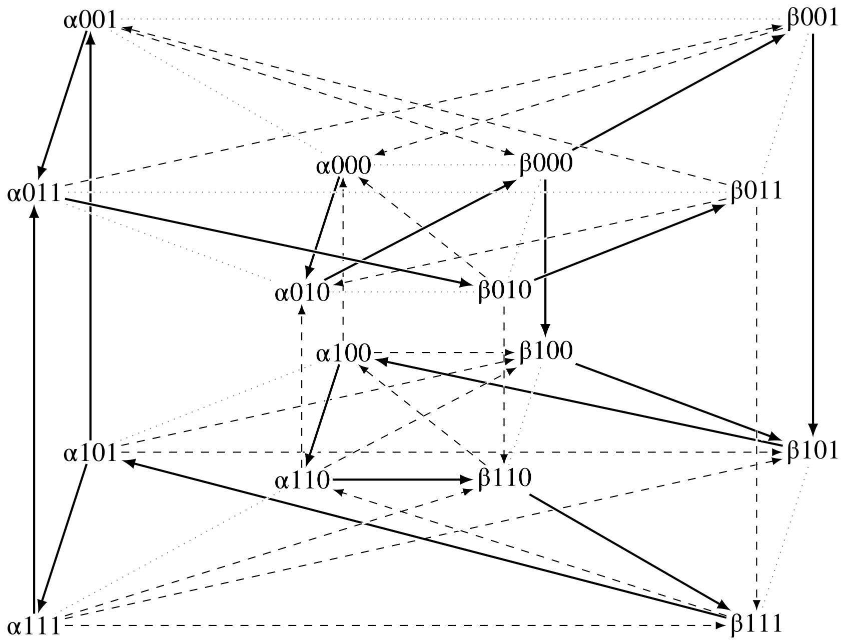


Figure 4: The transition graph of N3T9 representing a path realizing the longest distance.

shrink the diameter: $\mathsf{diam}(\to_0, \to_0, \to_1, \to_1, \to_2, \to_2) = 11$. It can be shown (cf. [63, Theorem 6.2.1.1] or [64, Theorem 5.1.1]) that N3T9 has the maximal diameter among all three-threaded binary programs.

IV. Local diameter

In this section we prove that the local diameter is bounded above by a constant independent of the number of threads, and that the least upper bound is computable by an explicit algorithm. Informally, this means that the shortest counterexamples to violated local safety properties are short, and, given enough computational resources, we can even say how short. In theory, a constant bound is far better than the trivial bound $\mathsf{diamax}^{\mathsf{loc}}(n) \leqslant \mathsf{diamax}(n)$ for all $n \in \mathbb{N}_+$.

(This section represents a strengthening and generalization of the corresponding result from [64].)

From an intuitive standpoint, the proof relies on the observation that the local-distance function is antitone in the program argument (i.e., roughly speaking, adding threads to a program reduces the local distances), as well as on any of the known coverability procedures for Petri nets.

We start with well-known order-theoretic background, continue with new order-theoretic results, and finally apply what we learned to programs.

IV.1. Preliminaries from general order theory

Informally, an antichain is a preorder in which no elements are comparable. More formally, given a preordered set $(P, \lesssim)$, a subset $A \subseteq P$ is called an *antichain* iff $\forall x, y \in A\colon (x \lesssim y \vee y \lesssim x) \Rightarrow x = y$.

Given an arbitrary set I and arbitrary posets $(P_i, \leq_i)$ for $i{\in}I$, the *componentwise partial order* on the product $\prod_{i\in I} P_i$ is given by

$$x \leq y \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall\, i{\in}I\colon x_i \leq_i y_i \qquad\qquad \big(x, y \in \prod_{i\in I} P_i\big)\,.$$

We call a map $\varphi\colon X \to Y$ between preordered sets $(X, \lesssim_X)$ and $(Y, \lesssim_Y)$

- an *order-homomorphism* iff $\forall\, x_1, x_2 \in X\colon x_1 {\lesssim_X} x_2 \Leftrightarrow \varphi(x_1) {\lesssim_Y} \varphi(x_2)$,
- an *order-epimorphism* iff φ is a surjective order-homomorphism, and
- an *order-isomorphism* iff φ is a bijective order-homomorphism. (In general, an isomorphism is defined as a bijective homomorphism such that its inverse is also a homomorphism [21]. In the case of order-isomorphisms, both definitions are equivalent; cf. [14, Lemma 2.2].)

If a bijective order-homomorphism between preordered sets $(X, \lesssim_X)$ and $(Y, \lesssim_Y)$ exists, we call these preordered sets *order-isomorphic* [37, p. 14, Ch. 0].

The following popular result is essential for the whole section:

Proposition IV.1.1 (Dickson's Lemma)**.** *Let $m \in \mathbb{N}_{\geqslant 0}$. Let the set $\mathbb{N}^m_{\geqslant 0}$ of m-tuples of natural numbers be equipped with componentwise partial order over the standard order on natural numbers. Then every antichain in $\mathbb{N}^m_{\geqslant 0}$ is finite.*

This is a special case of Hilbert's basis theorem (cf. Chapitre III, § 2.10, Corollaire 2 in [12]). We give an independent proof below.

Proof. We prove the statement by induction on m. Let $m \in \mathbb{N}_{\geqslant 0}$ be arbitrary, and assume that for every $l{<}m$, all antichains in $\mathbb{N}^l_{\geqslant 0}$ are finite.

Case $m \in \{0, 1\}$. By definition of the empty product, $|\mathbb{N}^0_{\geqslant 0}| = 1$, and $\mathbb{N}^1_{\geqslant 0}$ is order-isomorphic to $\mathbb{N}_{\geqslant 0}$. Thus, every antichain in $\mathbb{N}^m_{\geqslant 0}$ is either empty or a singleton.

Case $m{\geqslant}2$. Let $A \subseteq \mathbb{N}^m_{\geqslant 0}$ be a nonempty antichain. Notice that for each $k \in \mathbb{N}_{\geqslant 0}$ and each $i{<}m$ the set $\big\{b \in \mathbb{N}^{m-1}_{\geqslant 0} \;\big|\; (b_1, \ldots, b_{i-1}, k, b_{i+1}, \ldots, b_{m-1}) \in A\big\}$ is an antichain in $\mathbb{N}^{m-1}_{\geqslant 0}$, hence finite, so $A_{i,k} \overset{\text{def}}{=} \{a \in A \mid a_i = k\}$ is finite. Since A is nonempty, some $a \in A$ exists. Let $B = \bigcup\{A_{i,k} \mid i{<}m \wedge k{\leqslant}a_i\}$. Then B is finite. If A were infinite, some $c \in A \setminus B$ would exist. By construction, a is less than c in the componentwise partial order on $\mathbb{N}^m_{\geqslant 0}$. $\lightning$ ∎

IV.2. Well-foundedness and antitone maps

We recall that a partial order $\leq$ on a set X is *well-founded* iff each nonempty subset of X has a minimal element, i.e., $\forall\, Y{\subseteq}X\colon \big(Y{\neq}\emptyset \Rightarrow \exists y{\in}Y\, \forall z{\in}Y\colon (z{\leq}y \Rightarrow z{=}y)\big)$.

The following fact is also well known:

Proposition IV.2.1. *For each $m \in \mathbb{N}_{\geqslant 0}$, the componentwise partial order on $\mathbb{N}^m_{\geqslant 0}$ is well-founded.*

Proof. We write $\leq$ for the componentwise partial order on $\mathbb{N}^m_{\geqslant 0}$. Let $Y \subseteq \mathbb{N}^m_{\geqslant 0}$ be nonempty. Then there is some $y{\in}Y$ with the minimal 1-norm. Let $z{\in}Y$ be arbitrary such that $z{\leq}y$. Then $z_i \leqslant y_i$ for all $i{\in}m$. Assume for the purpose of contradiction that $z{\neq}y$. Then there is some $j{\in}m$ such that $z_j \neq y_j$. Then $z_j < y_j$. Then $\|z\|_1 < \|y\|_1$, in contradiction to the choice of y. Thus, our assumption was wrong and $z{=}y$. ∎

We combine this fact with Proposition IV.1.1:

Lemma IV.2.2. *Let $\mathbb{N}_{\geqslant 0}$ be equipped with the standard order on natural numbers. Let $m \in \mathbb{N}_{\geqslant 0}$, the set $\mathbb{N}_{\geqslant 0}^m$ be equipped with componentwise partial order, and $f\colon \mathbb{N}_{\geqslant 0}^m \dashrightarrow \mathbb{N}_{\geqslant 0}$ be an antitone partial map. Then:*

a) $\operatorname{img} f$ *is finite.*

b) *Assume that explicit algorithms solving the following problems exist:*
   1) *Decide, given a tuple of pairs $(s_i, a_i)_{i<m} \in (\{\text{‘='}, \text{‘}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^m$, whether some $y \in \operatorname{dom} f$ exists satisfying $\forall\, i<m\colon \big((s_i = \text{‘='} \wedge y_i = a_i) \vee (s_i = \text{‘}\geqslant\text{'} \wedge y_i \geqslant a_i)\big)$.*
   2) *Evaluate f at a point of its domain.*

   Then there is an explicit algorithm determining whether $\operatorname{img} f$ *is empty or not, and, in case of nonemptiness, computing* $\max(\operatorname{img} f)$.

Proof. a) If f is the empty partial map, its image is empty and, therefore, finite.
Let us consider the case of a nonempty f. We write $\preceq$ for the componentwise partial order on $\mathbb{N}_{\geqslant 0}^m$. Let $A \overset{\text{def}}{=} \{x \in \operatorname{dom} f \mid \forall\, y \in \operatorname{dom} f\colon y \preceq x \Rightarrow y{=}x\}$ be the set of the minimal elements of the domain of f. Then A is an antichain, hence finite by Proposition IV.1.1. The partial order $\preceq$ is well-founded by Proposition IV.2.1, and $\operatorname{dom} f$ is nonempty. By the definition of well-foundedness, A is nonempty. Therefore, $\{f(x) \mid x{\in}A\}$ is nonempty; let $M = \max\{f(x) \mid x{\in}A\}$.
It suffices to show that M is the maximum value of f. So let $x \in \operatorname{dom} f$ be arbitrary. Then the set $B \overset{\text{def}}{=} \{y \in \operatorname{dom} f \mid y \preceq x\}$ is nonempty, so it has a minimal element z.
Now we show that z lies in A. From $z{\in}B$ we get $z \in \operatorname{dom} f$. Let $y \in \operatorname{dom} f$ be arbitrary such that $y \preceq z$. From $z{\in}B$ we get $z \preceq x$. By transitivity, $y \preceq x$. From $y \in \operatorname{dom} f$ and $y \preceq x$, we obtain $y{\in}B$. From $y{\in}B$, $y \preceq z$, and the minimality of z, we obtain $y{=}z$. We have shown $z \in \operatorname{dom} f \wedge \forall\, y \in \operatorname{dom} f\colon y \preceq z \Rightarrow y{=}z$. Hence, $z{\in}A$.
From $z{\in}A$ we obtain $f(z) \leqslant M$. From $z \preceq x$ we obtain $f(x) \leqslant f(z)$. By transitivity, $f(x) \leqslant M$.
We have shown that $\operatorname{img} f \subseteq (M{+}1)$, which is a finite ordinal.

b) We will reuse the definitions of A and M given in the proof of part a).
Notice that the algorithm from 1) allows, in particular,
   i) deciding, given an arbitrary set $I \subseteq m$ and a sequence of pairs $(s_i, a_i)_{i\in I} \in (\{\text{‘='}\} \times \mathbb{N}_{\geqslant 0} \cup \{\text{‘}\geqslant\text{'}\} \times \mathbb{N}_{+})^I$, whether some $y \in \operatorname{dom} f$ exists satisfying $\bigwedge_{i\in I} \big((s_i = \text{‘='} \wedge y_i = a_i) \vee (s_i = \text{‘}\geqslant\text{'} \wedge y_i \geqslant a_i)\big)$ (by reusing $(s_i, a_i)_{i\in I}$ and additionally letting $s_i = \text{‘}\geqslant\text{'}$ and $a_i{=}0$ for all $i \in m \setminus I$);
   ii) deciding whether $\operatorname{dom} f$ is empty (by choosing $s_i = \text{‘}\geqslant\text{'}$ and $a_i{=}0$ for all $i<m$);
   iii) deciding, given a vector $a \in \mathbb{N}_{\geqslant 0}^m$, the membership $a \in \operatorname{dom} f$ (by choosing $s_i = \text{‘='}$ for all $i<m$); and
   iv) computing any y from i) if such a y exists (by enumerating all the tuples from $\mathbb{N}_{\geqslant 0}^m$ and, for each tuple, evaluating the constraint $\bigwedge_{i\in I}(\ldots)$ and, if the constraint is satisfied, checking for membership in $\operatorname{dom} f$).

Consider Algorithm 1.
The algorithm first checks whether $\operatorname{dom} f = \emptyset$; if so, $\operatorname{img} f$ is empty. In this case, the procedure returns “empty”.
Otherwise, a maximal (with respect to subset inclusion) antichain in $\operatorname{dom} f$ is constructed, beginning with an arbitrary element $x \in \operatorname{dom} f$. This is done as follows. Given the current antichain stored in the container D, a formula representing the statement “variable y is incomparable with each member of D” is constructed; the equivalent disjunctive normal form (DNF) is stored in F.
During conversion to DNF, we simplify F arithmetically and logically:

Algorithm 1: Finding the maximum value of an antitone function $\mathbb{N}_{\geqslant 0}^m \to \mathbb{N}_{\geqslant 0}$

Output: “empty” or M

Program variables: D, x, grow, F, d, y

1 **if** *any* $x \in \mathsf{dom}\, f$ *exists* **then** `// in this case` $\mathsf{dom}\, f \neq \emptyset$

2   choose any such x;

3   D := {x}; `// D always stores an antichain in dom` f

4   **repeat**

5   grow := false;

6   F := a simplification of a disjunctive normal form of

$$\bigwedge_{a \in \mathrm{D}} \bigvee_{\substack{i,j<m \\ i \neq j}} \Big(\big(\bigvee_{c<a_i} (y_i = c)\big) \wedge y_j \geqslant a_j + 1 \Big);$$

7   **while** F≠false $\wedge$ F *is a nonempty disjunction* $\wedge\, \neg$grow **do**

8   d := any disjunct of F;

9   Remove d from F;

10   **if** $\exists\, y \in \mathsf{dom}\, f$: d **then** `// using the algorithm from i)`

11   Compute any such y;

12   grow := true;

13   D := D $\cup$ {y}

14   **until** $\neg$grow; `// after the loop, D is a maximal antichain in dom` f

15   **return** $\max\{f(a) \mid a \in \mathsf{dom}\, f \wedge \exists\, b \in \mathrm{D}\colon a \leq b\}$ `// computable, finite set`

16 **else return** “empty” `// in this case` $\mathsf{dom}\, f = \emptyset$

- Empty disjuncts of the form $\bigvee_{c<0} (y_i = c)$ are replaced with false.
- Conjunctions of the form $y_i \geqslant c \wedge y_i \geqslant c'$ are replaced with $y_i \geqslant \max\{c, c'\}$ (for $c, c' \in \mathbb{N}_+$ and $i < m$).
- Conjunctions of the form $y_i \geqslant c \wedge y_i = c'$ are replaced with $y_i = c'$ (if $c \leqslant c'$) or false (if $c > c'$) (for $c \in \mathbb{N}_+$, $c' \in \mathbb{N}_{\geqslant 0}$, and $i < m$).
- Conjunctions of the form $y_i = c \wedge y_i = c'$ are replaced with $y_i = c$ (if $c' = c$) or false (if $c' \neq c$) (for $c, c' \in \mathbb{N}_{\geqslant 0}$ and $i < m$).
- The laws of absorption and annihilation (for the Boolean constant false) and associativity and commutativity are used to achieve maximal simplification.

After this simplification, in each disjunct of F, each variable y_i occurs at most once. Each resulting disjunct of F is now of the form $y_{i_1} (= \text{or} \geqslant) a_{i_1} \wedge \cdots \wedge y_{i_k} (= \text{or} \geqslant) a_{i_k}$ (for some $k \geqslant 1$, pairwise disjoint $i_1, \ldots, i_k < m$, and some $a_{i_1}, \ldots, a_{i_k} \in \mathbb{N}_{\geqslant 0}$ such that the right-hand sides of all the $\geqslant$-inequalities are positive). Among these disjuncts, we search, using i), for a disjunct that can be satisfied by some $y \in \mathsf{dom}\, f$. If such a disjunct is found, we extend the antichain (stored in D) with any corresponding $y \in \mathsf{dom}\, f$ and restart the outer loop from line 5. If no such disjunct is found, D already represents a maximal antichain. The outer loop, which extends D to a maximal antichain, terminates because of Proposition IV.1.1.

After the construction of a maximal antichain is finished, we notice that each $a \in A$ must be comparable to some $b \in \mathrm{D}$ (since otherwise D could be extended to a larger antichain) and (because A contains the minimal elements of $\mathsf{dom}\, f$) be less than or equal to this b. So $A \subseteq \{a \in \mathsf{dom}\, f \mid \exists\, b \in \mathrm{D}\colon a \leq b\}$, and the proof of a) implies that evaluating $f|_{\{a \in \mathsf{dom}\, f \mid \exists\, b \in D\colon a \leq b\}}$ suffices to find the maximum of f.

The evaluation of $f|_{\{a\in\mathrm{dom}\, f\,|\,\exists\, b\in D\colon a\leq b\}}$ can be done, e.g., by looping through all $b\in D$ and, for each such b, looping through all $a\leq b$. Since we thereby particularly evaluate $f|_A$, we necessarily encounter the maximal value of f in the process. ■

IV.3. Application of order theory to programs

In the following, let $\mathcal{P}$ be the set of all programs, equipped with the subprogram preorder (cf. Lemma II.4.1). In Lemmas IV.3.1 and IV.3.2, we discuss some structural properties of P.

Lemma IV.3.1. *There is some $k\in\mathbb{N}_+$ and some order-epimorphism $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}^k_{\geqslant 0}\setminus\big\{k\times\{0\}\big\}$, where the codomain is equipped with the componentwise partial order.*

(Since k is an ordinal, the above notation $k\times\{a\}$ means, set-theoretically, simply $\{0,\dots,k{-}1\}\times\{a\} = \{(0,a),\dots,(k{-}1,a)\} = \underbrace{(a,\dots,a)}_{k\text{ times}}$, a vector of k copies of a. If programs with no threads were allowed, the image of φ would include the all-zeros vector.)

Proof. Since a thread transition relation is a subset of $(\mathsf{Glob}\times\mathsf{Loc})^2$, there are $k \overset{\text{def}}{=} 2^{G^2L^2}$ different thread transition relations; we choose some enumeration $t_0,\dots,t_{k-1}$ of these relations. Let

$$\varphi\colon\ \mathcal{P}\ \to\ \mathbb{N}^k_{\geqslant 0}\setminus\big\{k\times\{0\}\big\},\quad (\to_0,\dots,\to_{n-1})\ \mapsto\ \Big(\big|\{\, i{<}n \mid \to_i\, = t_r\,\}\big|\Big)_{r<k}$$

be the map that, loosely speaking, counts how many copies of each of the k thread transition relations there are in a given program and then returns the vector of these counts. Since every program has at least one thread, and since $t_0,\dots,t_{k-1}$ counts all the thread transition relations, the image of φ does indeed not include the zero vector.

Now we prove the claims of the lemma. We write $\preceq$ for the componentwise partial order on $\mathbb{N}^k_{\geqslant 0}\setminus\big\{k\times\{0\}\big\}$.

"φ is an order-homomorphism." By definition, φ is total. Now let $p = (\rightsquigarrow_0,\dots,\rightsquigarrow_{m-1})$ and $q = (\to_0,\dots,\to_{n-1})$ be arbitrary members of $\mathcal{P}$. We are going to show:

"If p is a subprogram of q, then $\varphi(p)\preceq\varphi(q)$." Let p be a subprogram of q via an embedding $f\colon m\hookrightarrow n$. Let $r{<}k$ be arbitrary. The restriction $f|_{\{i<m|\rightsquigarrow_i=t_r\}}$ is still injective, and its image is a subset of $\{i{<}n \mid \to_i\, = t_r\}$. Hence, $\big|\{i{<}m \mid \rightsquigarrow_i\, = t_r\}\big| \leqslant \big|\{i{<}n \mid \to_i\, = t_r\}\big|$. Thus, $(\varphi(p))_r \leqslant (\varphi(q))_r$. Since r was arbitrary, we obtain $\varphi(p)\preceq\varphi(q)$.

"If $\varphi(p)\preceq\varphi(q)$, then p is a subprogram of q." Suppose $\varphi(p)\preceq\varphi(q)$. For each $r{<}k$ let $S_r = \{i{<}m \mid \rightsquigarrow_i\, = t_r\}$ and $S'_r = \{i{<}n \mid \to_i\, = t_r\}$. By the assumption, $|S_r| \leqslant |S'_r|$ $(r{<}k)$. Thus, for each $r{<}k$ there is an injection $f_r\colon S_r \hookrightarrow S'_r$. We view each map f_r as a set of pairs $(r{<}k)$ and define $f = \bigcup_{r<k} f_r$. Note that the sets S_r for $r{<}k$ are pairwise disjoint and that $\bigcup_{r<k} S_r = m$. Thus, f is a mapping $m\to n$. Note that the sets S'_r for $r{<}k$ are also pairwise disjoint. Thus, $f\colon m\hookrightarrow n$ is injective.

Now we prove that $\forall\, i{<}m\colon \rightsquigarrow_i\, = \to_{f(i)}$. Let $i{<}m$ be arbitrary. Then there is some $r{<}k$ such that $\rightsquigarrow_i\, = t_r$. So $i\in S_r$. Then $f(i) = f_r(i)$. From $\mathsf{img}\, f_r \subseteq S'_r$ we obtain $\to_{f(i)}\, = t_r$. Combining, we obtain $\rightsquigarrow_i\, = \to_{f(i)}$.

"φ is surjective." For each nonzero vector $(a_0,a_1,\dots,a_{k-1})\in\mathbb{N}^k_{\geqslant 0}$, one can always create a program with a_0 copies of t_0, a_1 copies of $t_1,\dots,a_{k-1}$ copies of t_{k-1}. ■

Moreover, all order-epimorphisms as in Lemma IV.3.1 are of the same form:

Lemma IV.3.2. *Let $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}^k_{\geqslant 0} \setminus \{k{\times}\{0\}\}$ be an order-epimorphism, where the codomain is equipped with the componentwise partial order. Then $k = 2^{G^2L^2}$, and there is an enumeration $t_0, \ldots, t_{k-1}$ of thread transition relations such that $\varphi\big((\rightarrow_i)_{i<n}\big) = \Big(\big|\{i{<}n \mid \rightarrow_i = t_r\}\big|\Big)_{r<k}$ for all programs $(\rightarrow_i)_{i<n}$. Moreover, φ is computable, and the preimage of each vector under φ is finite and computable.*

Proof. In the following, we write $\preceq$ for the componentwise partial order on $\mathbb{N}^k_{\geqslant 0} \setminus \{k \times\{0\}\}$ and $\prec$ for its irreflexive version. We write e^r for the unit vector $\left(\begin{cases}0, & \text{if } i{\neq}r\\ 1, & \text{if } i{=}r\end{cases}\right)_{i<k}$ $(r{<}k)$.

As a preliminary step, we will show:

Claim 1: For all $n \in \mathbb{N}_+$ and $p, q \in \mathcal{P}$, if p is n-threaded and q is an $(n{-}1)$-threaded subprogram of p, then there is some $r{<}k$ such that $\varphi(q) + e^r = \varphi(p)$.

To prove this, let $n \in \mathbb{N}_+$, p be an n-threaded program, and q be an $(n{-}1)$-threaded subprogram of p via some embedding f. Since φ is an order-homomorphism, $\varphi(q) \preceq \varphi(p)$. If we had $\varphi(p) \preceq \varphi(q)$, then p would be a subprogram of q, implying the existence of an injective map $n \hookrightarrow n{-}1$, $\lightning$. Therefore, $\varphi(q) \prec \varphi(p)$. Thus, there is some $r{<}k$ such that $\varphi(q) + e^r \preceq \varphi(p)$. Since φ is onto, there is some program q' such that $\varphi(q') = \varphi(q) + e^r$. From $\varphi(q) \prec \varphi(q')$ we obtain that q is a subprogram of q', and that q' is not a subprogram of q. Therefore, q' has strictly more than $n{-}1$ threads, i.e., at least n threads. From $\varphi(q') \preceq \varphi(p)$ we obtain that q' is a subprogram of p via some embedding g, and, therefore, q' cannot have more than n threads. Then $g\colon n{\hookrightarrow}n$, so, g is a bijection. Therefore, p is a subprogram of q' via the embedding g^{-1}. So $\varphi(p) \preceq \varphi(q')$. Therefore $\varphi(p) = \varphi(q')$, i.e., $\varphi(p) = \varphi(q) + e^r$, and Claim 1 is proven.

Now we consider another auxiliary statement:

Claim 2: φ maps single-threaded programs to unit vectors.

To prove this, assume for the sake of contradiction that some single-threaded $p{\in}\mathcal{P}$ is mapped to a non-unit vector. Since the zero vector is not in the image of φ, there must be $i, j < k$ such that $i{\neq}j$ and $(\varphi(p))_i \geqslant 1 \leqslant (\varphi(p))_j$. Then $e^i \prec \varphi(p)$. Since φ is onto, some $q{\in}\mathcal{P}$ satisfies $\varphi(q) = e^i$. From $\varphi(q) \prec \varphi(p)$ we obtain that q is a subprogram of p, but not vice versa. But p is single-threaded, so, q must be zero-threaded, which we explicitly excluded in § II.2. Thus, our assumption was wrong, and $\varphi(p)$ is a unit vector, which completes the proof of Claim 2.

Combining Claims 1 and 2, we obtain by induction:

$$\forall\, n \in \mathbb{N}_+,\ \ n\text{-threaded}\ p \in \mathcal{P}\colon\ \ \|\varphi(p)\|_1 = n\,. \tag{2}$$

Since there are exactly $k' \overset{\text{def}}{=} \Big|\mathfrak{P}\big((\mathsf{Glob} \times \mathsf{Loc})^2\big)\Big| = 2^{G^2L^2}$ thread transition relations, there are exactly this many single-threaded programs; they are all incomparable. Thus, the φ-images of the single-threaded programs comprise exactly k' pairwise incomparable unit vectors in $\mathbb{N}^k_{\geqslant 0} \setminus \big\{k{\times}\{0\}\big\}$. Therefore, $k{\geqslant}k'$. If k were strictly greater than k', then $\mathbb{N}^k_{\geqslant 0} \setminus \{k \times \{0\}\}$ would have a unit vector outside the image of single-threaded programs under φ, which, given the fact that φ is onto, would contradict (2). So $k{=}k'$, and the restriction of φ to single-threaded programs is a bijection to the set of unit vectors of $\mathbb{N}^k_{\geqslant 0} \setminus \{k \times \{0\}\}$.

For each $r{<}k$ we define t_r as the unique thread transition relation for which $\varphi((t_r)_{i<1}) = e^r$. Now we are going to prove by induction that $\varphi\big((\rightarrow_i)_{i<n}\big) \overset{!}{=} \Big(\big|\{i{<}n \mid \rightarrow_i = t_r\}\big|\Big)_{r<k}$ for all programs $(\rightarrow_i)_{i<n}$ for all $n \in \mathbb{N}_+$.

To this end, let $n \in \mathbb{N}_+$ be arbitrary, and assume (by inductive hypothesis) that the statement is true for all positive $m<n$. Let $p = (\to_i)_{i<n} \in \mathcal{P}$ be arbitrary.

Case n=1. Choose $s<k$ such that $t_s = \to_0$. Then $\varphi(p) = e^s$. The s^{th} component of e^s is $1 = \big|\{i<1 \mid \to_i = t_s\}\big|$, and every other r^{th} component of e^s for $r \neq s$ is $0 = \big|\{i<1 \mid \to_i = t_r\}\big|$.

Case $n \geqslant 2$. Let $q = (\to_i)_{i<n-1}$. The induction hypothesis implies

$$\varphi(q) \;=\; \Big(\big|\{i < n-1 \mid \to_i = t_r\}\big|\Big)_{r<k}. \tag{3}$$

By Claim 1, there is some $s<k$ such that $\varphi(p) = \varphi(q) + e^s$. There is also some $s'<k$ such that $\to_{n-1} = t_{s'}$. We will show that s and s' coincide.

Since the single-threaded program $(t_{s'})_{i<1}$ is a subprogram of p, we have $\varphi\big((t_{s'})_{i<1}\big) \preceq \varphi(p)$, and, therefore, $e^{s'} \preceq \varphi(p)$. Let $v = \varphi(p) - e^{s'}$. Note that $\|v\|_1 = \|\varphi(p)\|_1 - 1 \overset{(2)}{=} n-1$. Since φ is onto, $\varphi(q') = v$ for some $q' \in \mathcal{P}$. Due to (2), q' must be $(n-1)$-threaded. Let $(\rightsquigarrow_i)_{i<n-1} = q'$. Note that $v_s = (\varphi(q'))_s =$ [induction hypothesis] $\big|\{i < n-1 \mid \rightsquigarrow_i = t_s\}\big|$. From $\varphi(q') \prec \varphi(p)$ we obtain that q' is a subprogram of p. So p contains at least v_s copies of t_s.

Now, assume for the purpose of contradiction that $s \neq s'$. Then $t_s \neq \to_{n-1}$. Together with the definition of q, we obtain that q still has at least v_s copies of t_s. By (3), $(\varphi(q))_s \geqslant v_s$. The definition of s implies $(\varphi(p))_s \geqslant v_s + 1$. The definition of v implies $v_s = (\varphi(p) - e^{s'})_s = (\varphi(p))_s - e^{s'}_s \geqslant (v_s+1) - 0 = v_s + 1$, $\lightning$.

Thus, our assumption is wrong, and s=s'. Therefore, $\to_{n-1} = t_s$. Then $(\varphi(p))_s = (\varphi(q))_s + 1 \overset{(3)}{=} \big|\{i < n-1 \mid \to_i = t_s\}\big| + 1 = \big|\{i<n \mid \to_i = t_s\}\big|$. For $r \in k \setminus \{s\}$, we have $(\varphi(p))_r = (\varphi(q))_r + 0 \overset{(3)}{=} \big|\{i < n-1 \mid \to_i = t_r\}\big| =$ [since $t_r \neq t_s$] $\big|\{i<n \mid \to_i = t_r\}\big|$.

We have proved for all $n \in \mathbb{N}_+$ and all n-threaded $(\to_i)_{i<n} \in \mathcal{P}$ the equality

$$\varphi\big((\to_i)_{i<n}\big) \;=\; \Big(\big|\{i<n \mid \to_i = t_r\}\big|\Big)_{r<k}.$$

The map φ is computable by a loop over all $r<k$.

To show that the preimage of each vector under φ is finite, assume $a \in \mathbb{N}^k_{\geqslant 0} \setminus \big\{k \times \{0\}\big\}$. Each preimage $p \in \varphi^{-1}(a)$ is $\|a\|_1$-threaded due to (2). For each n the number of n-threaded programs is bounded from above by k^n. Therefore, a has no more than $k^{\|a\|_1}$ preimages.

To show that the preimage of a vector under φ is computable, assume again $a \in \mathbb{N}^k_{\geqslant 0} \setminus \big\{k \times \{0\}\big\}$. Construct an arbitrary program $(\to_i)_{i<n}$, where $n = \|a\|_1$, with a_0 copies of t_0, a_1 copies of t_1, ..., a_{k-1} copies of t_{k-1}. For each permutation σ of n, construct the program $(\to_{\sigma(i)})_{i<n}$. All thus created programs together form exactly $\varphi^{-1}(a)$. ∎

(As an aside, notice that Lemma IV.3.2 says that the order-epimorphisms from $\mathcal{P}$ onto the set of non–all-zeros tuples of natural numbers of the same dimension are the same up to permuting the components. This is slightly more general than needed by the following proofs. We added the above characterization to show that it might be difficult to work out a real-world implementation of the algorithm of this section by reducing the dimension.)

The above preparations imply:

Theorem IV.3.3. *Let $f : \mathcal{P} \dashrightarrow \mathbb{N}_{\geqslant 0}$ be an antitone partial map. Then:*

a) $\mathsf{img}\, f$ *is finite*

b) *Assume that there are explicit algorithms solving the following problems:*
   - *Membership of a given program in* $\mathsf{dom}\, f$.
   - *Decide, given functions* $s\colon \mathfrak{P}\big((\mathsf{Glob}{\times}\mathsf{Loc})^2\big) \to \{\text{‘='}, \text{‘}\geqslant\text{'}\}$ *and* $a\colon \mathfrak{P}\big((\mathsf{Glob}{\times}\mathsf{Loc})^2\big) \to \mathbb{N}_{\geqslant 0}$, *whether some* $(\to_i)_{i<n} \in \mathsf{dom}\, f$ *exists satisfying* $\forall \rightsquigarrow\, \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2\colon \big(s(\rightsquigarrow) = \text{‘='} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| = a(\rightsquigarrow)\big) \vee \big(s(\rightsquigarrow) = \text{‘}\geqslant\text{'} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| \geqslant a(\rightsquigarrow)\big)$.
   - *Evaluate* f *at a point of* $\mathsf{dom}\, f$.

   Then there is an explicit algorithm determining whether $\mathsf{img}\, f$ *is empty or not, and, in the case of nonemptiness, computing* $\max(\mathsf{img}\, f)$.

Proof. Let $f\colon \mathcal{P} \dashrightarrow \mathbb{N}_{\geqslant 0}$ be an antitone partial map.

a) Lemma IV.3.1 implies the existence of some $k \in \mathbb{N}_+$ and some order-epimorphism $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}^k_{\geqslant 0} \setminus \{k \times \{0\}\}$, where the codomain is equipped with the componentwise partial order, which we write as $\preceq$.

   Let $A = \{\varphi(p) \mid p \in \mathsf{dom}\, f\}$.

   For $a{\in}A$, we write $\varphi^{-1}(a) = \{p{\in}\mathcal{P} \mid \varphi(p) = a\}$ for the preimage of a under φ till the end of this proof.

   We are going to show that for each $a\,{\in}\,A$ the map f is constant on $(\mathsf{dom}\, f) \cap \varphi^{-1}(a)$. For that, let $p, q \in \mathsf{dom}\, f$ with $\varphi(p) = \varphi(q)$ be arbitrary. Then $\varphi(p) \preceq \varphi(q)$ and $\varphi(p) \succeq \varphi(q)$. Since φ is an order-homomorphism, p is a subprogram of q and vice versa. Since f is antitone, $f(p) \geqslant f(q)$ and $f(p) \leqslant f(q)$, implying $f(p) = f(q)$. Since p and q were arbitrary, we have shown that for each $a{\in}A$ the map $f|_{(\mathsf{dom}\, f)\cap\varphi^{-1}(a)}$ is constant.

   Thus, the map
   $$g\colon A \to \mathbb{N}_{\geqslant 0}, \quad a \mapsto f(p) \text{ for any } p \in (\mathsf{dom}\, f) \cap \varphi^{-1}(a)$$
   is well defined. It is also antitone, so, by Lemma IV.2.2a), $\mathsf{img}\, g$ is finite. Since $\forall\, p \in \mathsf{dom}\, f\colon f(p) = g(\varphi(p))$, we obtain
   $$\mathsf{img}\, f \subseteq \mathsf{img}\, g\,. \tag{4}$$
   Hence, $\mathsf{img}\, f$ is also finite.

b) Consider k, φ, g, and its domain $A = \mathsf{img}(\varphi|_{\mathsf{dom}\, f})$ from above. The very definition of g implies $\mathsf{img}\, g \subseteq \mathsf{img}\, f$, so, using (4), $\mathsf{img}\, g = \mathsf{img}\, f$. We will apply the algorithm from Lemma IV.2.2b) to g. To this end, it suffices to supply the algorithms solving the following problems:

   - Decide, given a tuple of pairs $(s_i, a_i)_{i<m} \in (\{\text{‘='}, \text{‘}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^m$, whether some $y \in \mathsf{dom}\, g$ exists satisfying $\forall\, i < m\colon \big((s_i = \text{‘='} \wedge y_i = a_i) \vee (s_i = \text{‘}\geqslant\text{'} \wedge y_i \geqslant a_i)\big)$. According to Lemma IV.3.2, there is some enumeration of thread transition relations $t_0, \ldots, t_{k-1}$ such that $\varphi\big((\to_i)_{i<n}\big) = \Big(\big|\{i{<}n \mid \to_i = t_r\}\big|\Big)_{r<k}$ for all programs $(\to_i)_{i<n}$.

     So assume that a sequence of pairs $(s_i, a_i)_{i<m} \in (\{\text{‘='}, \text{‘}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^m$ is given. Define $s'\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \{\text{‘='}, \text{‘}\geqslant\text{'}\}$ and $a'\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \mathbb{N}_{\geqslant 0}$ by $s'(t_i) = s_i$ and $a'(t_i) = a_i$ for all $i < m$. Notice that there is some $y \in (\mathsf{dom}\, g) = \mathsf{img}(\varphi|_{\mathsf{dom}\, f})$ satisfying $\forall\, i < m\colon \big((s_i = \text{‘='} \wedge y_i = a_i) \vee (s_i = \text{‘}\geqslant\text{'} \wedge y_i \geqslant a_i)\big)$ iff there is some $(\to_i)_{i<n} \in \mathsf{dom}\, f$ satisfying $\forall \rightsquigarrow\, \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2\colon \big(s'(\rightsquigarrow) = \text{‘='} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| = a'(\rightsquigarrow)\big) \vee \big(s'(\rightsquigarrow) = \text{‘}\geqslant\text{'} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| \geqslant a'(\rightsquigarrow)\big)$:

Input: $a \in A$
Output: $g(a)$
foreach $\mathrm{p} \in \varphi^{-1}(a)$ **do**
  if $\mathrm{p} \in \operatorname{dom} f$ **then return** $f(\mathrm{p})$

Algorithm 2: Evaluate g

"⇒": Given a y as stated, let $(\to_i)_{i<n}$ be an arbitrary member of $(\operatorname{dom} f)\cap\varphi^{-1}(y)$.

"⇐": Given a program $(\to_i)_{i<n}$ as stated, let $y = \varphi\big((\to_i)_{i<n}\big)$.

An algorithm answering whether, given functions $s' \colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \{`{=}', `{\geqslant}'\}$ and $a' \colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \mathbb{N}_{\geqslant 0}$, there is any $(\to_i)_{i<n} \in \operatorname{dom} f$ satisfying $\forall \rightsquigarrow \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2\colon \big(s'(\rightsquigarrow) = `{=}' \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| = a'(\rightsquigarrow)\big) \vee \big(s'(\rightsquigarrow) = `{\geqslant}' \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| \geqslant a'(\rightsquigarrow)\big)$ is provided by the assumption of this theorem.

- Evaluate g at a point of its domain. Since the preimage of a vector under φ is computable according to Lemma IV.3.2, the problem is solvable by Algorithm 2.

■

In the following, states of a particular form play an important role. We call a program state $(g, l) \in \mathsf{Glob}\times\mathsf{Loc}^n$ of any n-threaded program *uniform* if all components of l are the same, i.e., $\forall\, i, j \in n\colon\ l_i = l_j$.

Now we instantiate a particular antitone map in Theorem IV.3.3:

Lemma IV.3.4. *For all $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $\rightsquigarrow \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$ there is some $c \in \mathbb{N}_{\geqslant 0}$ satisfying the following property: for all $n \in \mathbb{N}_+$, all n-threaded programs $(\to_0, \ldots, \to_{n-1})$, and all $i<n$, if $\to_i = \rightsquigarrow$ and $\mathsf{d}^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$, then $\mathsf{d}^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant c$. Moreover, the function mapping a tuple $(g, g', a, a', \rightsquigarrow) \in \mathsf{Glob}\times\mathsf{Glob}\times\mathsf{Loc}\times\mathsf{Loc}\times\mathfrak{P}((\mathsf{Glob}\times\mathsf{Loc})^2)$ to the smallest c satisfying the above property possesses an explicit algorithm.*

Proof. Fix arbitrary $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $\rightsquigarrow \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$. We define $f\colon \mathcal{P} \dashrightarrow \mathbb{N}_{\geqslant 0}$ with $\operatorname{dom} f = \Big\{(\to_i)_{i<n}\in\mathcal{P} \;\Big|\; \exists i<n\colon \to_i = \rightsquigarrow \wedge\, \mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty\Big\}$, $(\operatorname{dom} f) \ni (\to_i)_{i<n} \mapsto \min\Big\{\mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n\times\{a\}), i, (g', a')\big) \;\Big|\; i<n \wedge \rightsquigarrow = \to_i\Big\}$. (The notation $\mathsf{d}^{\mathsf{loc}}_p(\ldots)$ for a program $p = (\to_i)_{i<n}$ has been defined in § II.2.)

Now we will show that f is antitone. Let $p, q \in \operatorname{dom} f$ such that p is a subprogram of q. Let $(\to_i)_{i<n} = p$ and $(\rightarrowtail_i)_{i<m} = q$. Then there is an injective map $h\colon n \hookrightarrow m$ such that $\to_i = \rightarrowtail_{h(i)}$ for all $i<n$. Since $p \in \operatorname{dom} f$, there is some $\hat{\imath}<n$ such that $\rightsquigarrow = \to_{\hat{\imath}}$ and a path σ in the transition graph of p such that $\sigma_0 = (g, n\times\{a\})$, the last state of σ is (g', l') for some $l' \in \mathsf{Loc}^n$ such that $l'_{\hat{\imath}} = a'$, and $\mathsf{length}(\sigma) = f(p)$. Informally, we will now lift the path σ from p to q by adding threads that always stay at the local state a. In the following, let $h^{-1}\colon (\operatorname{img} h) \hookrightarrow\!\!\!\!\rightarrow n$ be the inverse of h on its image. We define a sequence $\hat{\sigma} = (\hat{\sigma}_k)_{k\leqslant f(p)} \in (\mathsf{Glob}\times\mathsf{Loc}^m)^{f(p)+1}$ by $\hat{\sigma}_k \overset{\text{def}}{=} \left(\check{g}^{[k]}, \left(\begin{cases} \check{l}^{[k]}_{h^{-1}(j)}, & \text{if } j \in \operatorname{img} h,\\ a, & \text{otherwise}\end{cases}\right)_{j<m}\right)$, where $(\check{g}^{[k]}, \check{l}^{[k]}) \overset{\text{def}}{=} \sigma_k$, for all $k \leqslant f(p)$. Then $\hat{\sigma}$ is a path in the transition graph of q such that $\hat{\sigma}$ starts in $(g, m\times\{a\})$ and ends in a program state (g', l'') for some $l'' \in \mathsf{Loc}^m$ satisfying $l''_{h(\hat{\imath})} = l'_{\hat{\imath}} = a'$. So $\mathsf{d}^{\mathsf{loc}}_q\big((g, m\times\{a\}), h(\hat{\imath}), (g', a')\big) < \infty$. Notice that $\rightsquigarrow = \to_{\hat{\imath}} = \rightarrowtail_{h(\hat{\imath})}$, so $q \in \operatorname{dom} f$ and $f(q) \leqslant f(p)$. Since p and q were arbitrary, we have shown that f is antitone.

Theorem IV.3.3a) implies that f is bounded from above by some $c \in \mathbb{N}_{\geqslant 0}$.

Now let an arbitrary $n \in \mathbb{N}_+$, an arbitrary n-threaded program $p = (\rightarrow_i)_{i<n}$, an arbitrary $i < n$ such that $\rightarrow_i = \rightsquigarrow$ and $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ be given. Then $p \in \mathsf{dom}\, f$. The definition of f implies the existence of some $\hat\imath < n$ such that $\rightarrow_{\hat\imath} = \rightsquigarrow$ and of some path σ of length $f(p)$ such that σ starts in $(g, n\times\{a\})$ and ends in (g', l') for some $l' \in \mathsf{Loc}^n$ satisfying $l'_{\hat\imath} = a'$. Note that threads i and $\hat\imath$ have the same thread transition relation. Swap the steps of these threads: consider the sequence $\hat\sigma = (\hat\sigma_k)_{k\leqslant f(p)} \in (\mathsf{Glob}\times\mathsf{Loc}^n)^{f(p)+1}$ of program states, defined by $\hat\sigma_k \overset{\text{def}}{=} \left(\check g^{[k]}, \left(\begin{cases} \check l_j^{[k]}, & \text{if } j \notin \{i, \hat\imath\}, \\ \check l_i^{[k]}, & \text{if } j = \hat\imath\,, \\ \check l_{\hat\imath}^{[k]}, & \text{if } j = i \end{cases}\right)_{j<n}\right)$, where $(\check g^{[k]}, \check l^{[k]}) \overset{\text{def}}{=} \sigma_k$, for all $k \leqslant f(p)$. Then $\hat\sigma$ is a path in the transition graph of p of length $f(p)$ that starts in $(g, n\times\{a\})$ and ends in (g', l'') for some $l'' \in \mathsf{Loc}^n$ such that $l''_i = a'$. Since $\mathsf{length}(\hat\sigma) = f(p) \leqslant c$, we obtain $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant c$.

To prove the existence of an explicit algorithm computing the minimal such c as required, let us assume a tuple $(g, g', a, a', \rightsquigarrow)$ from the finite (!) set $S = \mathsf{Glob}\times\mathsf{Glob}\times\mathsf{Loc}\times\mathsf{Loc}\times\mathfrak{P}((\mathsf{Glob}\times\mathsf{Loc})^2)$ and define f as above. We will apply Theorem IV.3.3b). To this end, it suffices to supply the algorithms solving the following problems:

- Membership of a given program in $\mathsf{dom}\, f$.

  Let $p = (\rightarrow_i)_{i<n} \in \mathcal{P}$ be arbitrary. To test whether p belongs to $\mathsf{dom}\, f$, we first search for an $i<n$ such that $\rightarrow_i = \rightsquigarrow$ and return "false" if we find none. If one is found, we check for this i whether $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ by searching in the transition graph of p for a program state that is reachable from $(g, n\times\{a\})$, has the shared part g', and has the local part a' of thread i. We return "true" if we find such a program state and "false" otherwise.

- Decide, given functions $s\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \{\text{'='}, \text{'}\geqslant\text{'}\}$ and $b\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \mathbb{N}_{\geqslant 0}$, whether some $(\rightarrow_i)_{i<n} \in \mathsf{dom}\, f$ exists satisfying $\forall \rightsquigarrow\, \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2\colon \big(s(\rightsquigarrow) = \text{'='} \wedge \big|\{i<n \mid \rightarrow_i = \rightsquigarrow\}\big| = b(\rightsquigarrow)\big) \vee \big(s(\rightsquigarrow) = \text{'}\geqslant\text{'} \wedge \big|\{i<n \mid \rightarrow_i = \rightsquigarrow\}\big| \geqslant b(\rightsquigarrow)\big)$.

  We let $k = \big|\mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big)\big|$ and choose any enumeration $t_0, \ldots, t_{k-1}$ of the thread transition relations.

  Let functions $s\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \{\text{'='}, \text{'}\geqslant\text{'}\}$ and $b\colon \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \to \mathbb{N}_{\geqslant 0}$ be given as input.

  We now create a Petri net simulating multithreaded programs as follows.

  Without loss of generality, suppose $(k\times\mathsf{Loc}) \cap \mathsf{Glob} = \emptyset$; otherwise we rename the members of Loc or Glob to attain this disjointness. Choose a fresh individual start $\notin (k\times\mathsf{Loc}) \cup \mathsf{Glob}$. We construct the set of the Petri net's places as the disjoint union $(k\times\mathsf{Loc}) \,\dot\cup\, \mathsf{Glob} \,\dot\cup\, \{\text{start}\}$.

  Initially, the Petri net has one token in start and $b(t_j)$ tokens in (j, a) for each $j < k$; there are no other tokens in the initial marking.

  The set of transitions is formed as a union of three pairwise disjoint sets. To define the first of these sets, let $I = \{\rightarrowtail\, \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2 \mid s(\rightarrowtail) = \text{'}\geqslant\text{'}\}$; the first Petri-net–transition set has $|I|$ transitions indexed by I, and for each $j < k$ such that $t_j \in I$, the transition with index t_j consumes one token from start, puts one token back into start, and adds one token to the place (j, a). The second Petri-net–transition set comprises a single transition that completely leaves start: this transition consumes a token from start and adds a token to the place g. The third Petri-net–transition set comprises the transitions actually simulating the threads: for each $j < k$ and each thread transition $\big((x, y), (x', y')\big) \in t_j$, the Petri net contains a transition removing one token from (j, y) and one token from x and then placing one token to (j, y') and one token to x'.

After having constructed such a Petri net, we check for the coverability of the marking in which the place g' has one token, for the unique $j < k$ satisfying $t_j = \rightsquigarrow$ the place (j, a') has one token, and all the other places have no tokens. Any coverability procedure (e.g., [48]) would do.

To see the correctness of the construction, notice that the aforementioned Petri net simulates programs with at least $b(\rightarrowtail)$ copies of each $\rightarrowtail \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$ with $s(\rightarrowtail)$ = '$\geqslant$' and exactly $b(\rightarrowtail)$ copies of each $\rightarrowtail \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$ with $s(\rightarrowtail)$ = '='.

- Evaluate f at a point of $\mathsf{dom}\, f$.

  Given an n-threaded program in $\mathsf{dom}\, f$, perform a breadth-first search from $(g, n\times\{a\})$. At each level, watch for the program states with shared part g' and check whether in such a program state any of the threads with the transition relation $\rightsquigarrow$ has a' as the local part. As soon as such a program state is found, stop and return the level number. (Here, we start counting the level numbers of our breadth-first–search tree by assigning 0 to the root.)

Since S is finite, the combined algorithm that takes a member of S and starts the sub-algorithm for the corresponding f is also explicit. ∎

Since there are only finitely many shared states, local states, and thread transition relations, a maximal c from Lemma IV.3.4 can be computed:

Corollary IV.3.5. *There is some $c \in \mathbb{N}_{\geqslant 0}$ such that, for all $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, $n \in \mathbb{N}_+$, all n-threaded programs p, and all $i<n$, if $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$, then $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant c$. Moreover, there is an explicit algorithm constructing the smallest such c.*

Proof. According to Lemma IV.3.4, there is a map $\zeta\colon \mathsf{Glob}\times\mathsf{Glob}\times\mathsf{Loc}\times\mathsf{Loc}\times\mathfrak{P}((\mathsf{Glob}\times\mathsf{Loc})^2) \to \mathbb{N}_{\geqslant 0}$ such that, for all $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, $\rightsquigarrow \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$, $n\in\mathbb{N}_+$, n-threaded programs $p = (\to_i)_{i<n}$, and all $i < n$, if $\to_i = \rightsquigarrow$ and $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$, then $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant \zeta(g, g', a, a', \rightsquigarrow)$. We choose the pointwise smallest such ζ; then an explicit algorithm computing ζ exists. Let $c = \max(\mathsf{img}\,\zeta)$. This value can be explicitly constructed by evaluating ζ at all the points of its finite domain and taking the maximal value.

Now consider an arbitrary program $p = (\to_i)_{i<n}$. For all $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $i < n$ such that $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ we have $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant \zeta(g, g', a, a', \to_i) \leqslant c$.

Thus, $\min\Big\{\hat{c} \in \mathbb{N}_{\geqslant 0} \;\Big|\; \forall\, g, g' \in \mathsf{Glob},\ a, a' \in \mathsf{Loc},\ n \in \mathbb{N}_+,\ n\text{-threaded program } p,\ i<n\colon \mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty \Rightarrow \mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) \leqslant \hat{c}\Big\} \leqslant c$. To show that the last inequality is actually an equality, let $\hat{c}$ be the minimum on the left-hand side. Since $c \in \mathsf{img}\,\zeta$, there are some $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $\rightsquigarrow \subseteq (\mathsf{Glob}\times\mathsf{Loc})^2$ such that $\zeta(g, g', a, a', \rightsquigarrow) = c$.

Case $c=0$. Then $\hat{c} \geqslant c$ holds immediately.

Case $c>0$. Since ζ was chosen pointwise minimal, its definition implies that there is an $n \in \mathbb{N}_+$, an n-threaded program $p = (\to_i)_{i<n}$, and an $i < n$ such that $\to_i = \rightsquigarrow$ and $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ and $\mathsf{d}_p^{\mathsf{loc}}\big((g, n\times\{a\}), i, (g', a')\big) = \zeta(g, g', a,$

$a', \rightsquigarrow)$, i.e., $\mathsf{d}_p^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) = c$. The definition of $\hat{c}$ implies $\mathsf{d}_p^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) \leqslant \hat{c}$, i.e., $c \leqslant \hat{c}$.

In both cases, $\hat{c} \geqslant c$, proving $\min\Big\{\hat{c} \in \mathbb{N}_{\geqslant 0} \;\Big|\; \forall\, g, g' \in \mathsf{Glob},\ a, a' \in \mathsf{Loc},\ n \in \mathbb{N}_+,\ n\text{-threaded program } p,\ i < n\colon \mathsf{d}_p^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) < \infty \Rightarrow \mathsf{d}_p^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) \leqslant \hat{c}\Big\} = c$. ∎

In Corollary IV.3.5, the source program state is uniform. Before we remove this restriction, we argue that local distances stay invariant under renaming of local states. More formally:

Lemma IV.3.6. *Let $p = (\rightarrow_j)_{j<n}$ be a program. For each $j<n$ let h_j be a permutation of Loc. Moreover, for each $j<n$ let a thread transition relation $\rightsquigarrow_j \subseteq (\mathsf{Glob}{\times}\mathsf{Loc})^2$ be defined via*

$$(g, l) \rightsquigarrow_j (g', l') \quad \overset{def}{\Longleftrightarrow} \quad \big(g, h_j(l)\big) \rightarrow_j \big(g', h_j(l')\big)$$

for all $g, g' \in \mathsf{Glob}$ and $l, l' \in \mathsf{Loc}$; let $q = (\rightsquigarrow_j)_{j<n}$.
Then, for all $(g, l) \in \mathsf{State}_p$, $i < n$, $g' \in \mathsf{Glob}$, and $b \in \mathsf{Loc}$, we have

$$\mathsf{d}_q^{\mathsf{loc}}\big((g, l), i, (g', b)\big) \;=\; \mathsf{d}_p^{\mathsf{loc}}\Big(\big(g, (h_j(l_j))_{j<n}\big), i, \big(g', h_i(b)\big)\Big).$$

Proof. Let $(g, l) \in \mathsf{State}_p$, $i<n$, $g' {\in} \mathsf{Glob}$, and $b {\in} \mathsf{Loc}$. We are going to show the equality in question by separating it into two inequalities.

"$\mathsf{d}_q^{\mathsf{loc}}\big((g, l), i, (g', b)\big) \overset{!}{\leqslant} \mathsf{d}_p^{\mathsf{loc}}\Big(\big(g, (h_j(l_j))_{j<n}\big), i, \big(g', h_i(b)\big)\Big)$": If the right-hand side is ∞, the inequality holds trivially. Thus, let us assume from now on that the right-hand side is equal to some $k \in \mathbb{N}_{\geqslant 0}$. Then there is a path $\sigma = \big((\hat{g}^{[r]}, \hat{l}^{[r]})\big)_{r \leqslant k}$ in the transition graph of p such that $\hat{g}^{[0]} = g$, $\hat{l}^{[0]} = \big(h_j(l_j)\big)_{j<n}$, $\hat{g}^{[k]} = g'$, and $\hat{l}_i^{[k]} = h_i(b)$. There is a map $t\colon k {\rightarrow} n$ that tells us which thread takes a step at each time point, i.e., such that for each $r{<}k$ we have $(\hat{g}^{[r]}, \hat{l}_{t(r)}^{[r]}) \rightarrow_{t(r)} (\hat{g}^{[r+1]}, \hat{l}_{t(r)}^{[r+1]})$ and $\forall\, j \in n \backslash \{t(r)\}\colon \hat{l}_j^{[r]} = \hat{l}_j^{[r+1]}$. Then for each $r{<}k$ we have $\Big(\hat{g}^{[r]}, h_{t(r)}\big(h_{t(r)}^{-1}(\hat{l}_{t(r)}^{[r]})\big)\Big) \rightarrow_{t(r)} \Big(\hat{g}^{[r+1]}, h_{t(r)}\big(h_{t(r)}^{-1}(\hat{l}_{t(r)}^{[r+1]})\big)\Big)$, which, using the assumption, implies $\big(\hat{g}^{[r]}, h_{t(r)}^{-1}(\hat{l}_{t(r)}^{[r]})\big) \rightsquigarrow_{t(r)} \big(\hat{g}^{[r+1]}, h_{t(r)}^{-1}(\hat{l}_{t(r)}^{[r+1]})\big)$. Let $\check{l}^{[r]} = \big(h_j^{-1}(\hat{l}_j^{[r]})\big)_{j<n}$ for all $r{\leqslant}k$. Then for each $r{<}k$ we have $(\hat{g}^{[r]}, \check{l}_{t(r)}^{[r]}) \rightsquigarrow_{t(r)} (\hat{g}^{[r+1]}, \check{l}_{t(r)}^{[r+1]})$ and $\forall\, j \in n\backslash\{t(r)\}\colon \check{l}_j^{[r]} = h_j^{-1}(\hat{l}_j^{[r]}) = h_j^{-1}(\hat{l}_j^{[r+1]}) = \check{l}_j^{[r+1]}$. Therefore, the sequence $\check{\sigma} \overset{\text{def}}{=} \Big((\hat{g}^{[r]}, \check{l}^{[r]})\Big)_{r\leqslant k}$ is a walk in the transition graph of q. The walk starts in $\Big(\hat{g}^{[0]}, \big(h_j^{-1}(\hat{l}_j^{[0]})\big)_{j<n}\Big) = \big(\hat{g}^{[0]}, (l_j)_{j<n}\big) = (g, l)$ and ends in the program state $\Big(\hat{g}^{[k]}, \big(h_j^{-1}(\hat{l}_j^{[k]})\big)_{j<n}\Big) = \Big(g', \big(h_j^{-1}(\hat{l}_j^{[k]})\big)_{j<n}\Big)$, whose local part of thread i is $h_i^{-1}(\hat{l}_i^{[k]}) = b$. Note that $\mathsf{length}(\check{\sigma}) = k$. Therefore, $\mathsf{d}_q^{\mathsf{loc}}\big((g, l), i, (g', b)\big) \leqslant k$.

"$\mathsf{d}_q^{\mathsf{loc}}\big((g, l), i, (g', b)\big) \overset{!}{\geqslant} \mathsf{d}_p^{\mathsf{loc}}\Big(\big(g, (h_j(l_j))_{j<n}\big), i, \big(g', h_i(b)\big)\Big)$": Analogously as follows. If the left-hand side is ∞, the inequality holds trivially. Thus, let us assume from now on that the left-hand side is equal to some $k {\in} \mathbb{N}_{\geqslant 0}$. Then there is a path $\sigma = \big((\hat{g}^{[r]}, \hat{l}^{[r]})\big)_{r\leqslant k}$ in the transition graph of q such that $\hat{g}^{[0]} = g$, $\hat{l}^{[0]} = l$, $\hat{g}^{[k]} = g'$, and $\hat{l}_i^{[k]} = b$. There is a map $t\colon k {\rightarrow} n$ that tells us which thread takes a

step at each time point, i.e., such that for each $r{<}k$ we have $(\hat{g}^{[r]}, \hat{l}^{[r]}_{t(r)}) \rightsquigarrow_{t(r)} (\hat{g}^{[r+1]}, \hat{l}^{[r+1]}_{t(r)})$ and $\forall\, j \in n\backslash\{t(r)\}\colon \hat{l}^{[r]}_j = \hat{l}^{[r+1]}_j$. The assumption implies that for each $r{<}k$ we have $\big(\hat{g}^{[r]}, h_{t(r)}(\hat{l}^{[r]}_{t(r)})\big) \rightarrow_{t(r)} \big(\hat{g}^{[r+1]}, h_{t(r)}(\hat{l}^{[r+1]}_{t(r)})\big)$. Let $\check{l}^{[r]} = \big(h_j(\hat{l}^{[r]}_j)\big)_{j<n}$ for each $r{\leqslant}k$. Then for each $r{<}k$ we have $(\hat{g}^{[r]}, \check{l}^{[r]}_{t(r)}) \rightarrow_{t(r)} (\hat{g}^{[r+1]}, \check{l}^{[r+1]}_{t(r)})$ and $\forall\, j \in n\backslash\{t(r)\}\colon \check{l}^{[r]}_j = h_j(\hat{l}^{[r]}_j) = h_j(\hat{l}^{[r+1]}_j) = \check{l}^{[r+1]}_j$. Therefore, the sequence $\check{\sigma} \stackrel{\text{def}}{=} \big((\hat{g}^{[r]}, \check{l}^{[r]})\big)_{r\leqslant k}$ is a walk in the transition graph of p. The walk starts in $\Big(\hat{g}^{[0]}, \big(h_j(\hat{l}^{[0]}_j)\big)_{j<n}\Big) = \Big(g, \big(h_j(l_j)\big)_{j<n}\Big)$ and ends in the program state $\Big(\hat{g}^{[k]}, \big(h_j(\hat{l}^{[k]}_j)\big)_{j<n}\Big) = \Big(g', \big(h_j(\hat{l}^{[k]}_j)\big)_{j<n}\Big)$, whose local part of thread i is $h_i(\hat{l}^{[k]}_i) = h_i(b)$. Note that $\mathsf{length}(\check{\sigma}) = k$. Therefore, $\mathsf{d}^{\mathsf{loc}}_p\Big(\big(g, (h_j(l_j))_{j<n}\big), i, \big(g', h_i(b)\big)\Big) \leqslant k$. ∎

Applying the above result, we obtain:

Theorem IV.3.7. *There is some $c \in \mathbb{N}_{\geqslant 0}$ such that $\forall\, n \in \mathbb{N}_+\colon \mathsf{diamax}^{\mathsf{loc}}(n) \leqslant c$. Moreover, the smallest such c can be constructed by an explicit algorithm.*

Proof. We fix c to be the smallest constant from Corollary IV.3.5. Let $n \in \mathbb{N}_+$. Let $p = (\rightarrow_i)_{i<n}$ be an n-threaded program with the set of program states State; we will show that the local diameter of p is bounded by c.

Let $(g, l) \in \mathsf{State}$, $i{<}n$, $g'{\in}\mathsf{Glob}$, and $b{\in}\mathsf{Loc}$ be arbitrary such that $\mathsf{d}^{\mathsf{loc}}_p\big((g,l), i, (g', b)\big) < \infty$. It suffices to show that $\mathsf{d}^{\mathsf{loc}}_p\big((g,l), i, (g',b)\big) \stackrel{!}{\leqslant} c$. For each $j{<}n$ we let

$$h_j\colon \mathsf{Loc} \hookrightarrow\!\!\!\!\rightarrow \mathsf{Loc}, \quad x \mapsto \begin{cases} x, & \text{if } x \notin \{l_i, l_j\}, \\ l_i, & \text{if } x = l_j\,, \\ l_j, & \text{if } x = l_i \end{cases}$$

be the permutation that swaps l_i with l_j while leaving all other local states invariant. We define

$$(\tilde{g}, \tilde{l}) \rightsquigarrow_j (\tilde{g}', \tilde{l}') \stackrel{\text{def}}{\iff} \big(\tilde{g}, h_j(\tilde{l})\big) \rightarrow_j \big(\tilde{g}', h_j(\tilde{l}')\big)$$

for all $\tilde{g}, \tilde{g}' \in \mathsf{Glob}$, $\tilde{l}, \tilde{l}' \in \mathsf{Loc}$, and $j < n$. We set $q \stackrel{\text{def}}{=} (\rightsquigarrow_j)_{j<n}$. Then $\mathsf{d}^{\mathsf{loc}}_p\big((g,l), i, (g',b)\big) = \mathsf{d}^{\mathsf{loc}}_p\Big(\Big(g, \big(h_j(h_j^{-1}(l_j))\big)_{j<n}\Big), i, \Big(g', h_i\big(h_i^{-1}(b)\big)\Big)\Big) =$ [by Lemma IV.3.6] $\mathsf{d}^{\mathsf{loc}}_q\Big(\Big(g, \big(h_j^{-1}(l_j)\big)_{j<n}\Big), i, \big(g', h_i^{-1}(b)\big)\Big) = \mathsf{d}^{\mathsf{loc}}_q\big((g, n{\times}\{l_i\}), i, (g', h_i^{-1}(b))\big) =$ [since $h_i = \mathsf{id}_{\mathsf{Loc}}$] $\mathsf{d}^{\mathsf{loc}}_q\big((g, n{\times}\{l_i\}), i, (g', b)\big)$. Therefore, $\mathsf{d}^{\mathsf{loc}}_q\big((g, n{\times}\{l_i\}), i, (g', b)\big) < \infty$. Corollary IV.3.5 implies $\mathsf{d}^{\mathsf{loc}}_q\big((g, n{\times}\{l_i\}), i, (g', b)\big) \leqslant c$. Thus, $\mathsf{d}^{\mathsf{loc}}_p\big((g,l), i, (g',b)\big) \leqslant c$.

Since p was arbitrary, we have shown $\max(\mathsf{img}\ \mathsf{diamax}^{\mathsf{loc}}) \leqslant c$. To show that the last inequality is actually an equality, assume for the purpose of contradiction that $\max(\mathsf{img}\ \mathsf{diamax}^{\mathsf{loc}})$ is strictly less than c. Thus, $c > 0$. Recall that $c = \min\Big\{\tilde{c}{\in}\mathbb{N}_{\geqslant 0} \,\Big|\, \forall\, g, g' \in \mathsf{Glob}, a, a' \in \mathsf{Loc}, n{\in}\mathbb{N}_+, n\text{-threaded program } p, i{<}n\colon \mathsf{d}^{\mathsf{loc}}_p\big((g, n{\times}\{a\}), i, (g', a')\big) < \infty \Rightarrow \mathsf{d}^{\mathsf{loc}}_p\big((g, n{\times}\{a\}), i, (g', a')\big) \leqslant \tilde{c}\Big\}$. Since $c{-}1$ does not belong to the set over which the minimum is taken, there is some $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, $n \in \mathbb{N}_+$, n-threaded program p, and $i{<}n$ such that $\mathsf{d}^{\mathsf{loc}}_p((g, n{\times}\{a\}), i, (g', a')) < \infty$ but $\mathsf{d}^{\mathsf{loc}}_p((g, n{\times}\{a\}), i, (g', a'))$

$\not\leqslant c{-}1$. Then $\mathsf{d}^{\mathsf{loc}}_p((g, n{\times}\{a\}), i, (g', a')) \geqslant c$, contradicting our assumption. Thus, our assumption was wrong and $\max(\mathsf{img}\,\mathsf{diamax}^{\mathsf{loc}}) = c$. ■

Due to Theorem IV.3.7, the maximal local diameter for programs can be determined:

Definition and Corollary IV.3.8. $C \stackrel{def}{=} \max(\mathsf{img}\,\mathsf{diamax}^{\mathsf{loc}})$ *exists and is computable by an explicit algorithm.* □

The top level of computing C is "compute $\max(\mathsf{img}\,\zeta)$" as Algorithm 3 demonstrates, where $\zeta\colon \mathsf{Glob}{\times}\mathsf{Glob}{\times}\mathsf{Loc}{\times}\mathsf{Loc}{\times}\mathfrak{P}((\mathsf{Glob}{\times}\mathsf{Loc})^2) \to \mathbb{N}_{\geqslant 0}$ is taken from the proof of Corollary IV.3.5. The function ζ is the pointwise smallest function such that, for all g, $g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $\rightsquigarrow \subseteq (\mathsf{Glob}{\times}\mathsf{Loc})^2$, we have: for all $n \in \mathbb{N}_+$, all n-threaded programs $(\to_0, \ldots, \to_{n-1})$, and all $i{<}n$, if $\to_i = \rightsquigarrow$ and $\mathsf{d}^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) < \infty$, then $\mathsf{d}^{\mathsf{loc}}\big((g, n{\times}\{a\}), i, (g', a')\big) \leqslant \zeta(g, g', a, a', \rightsquigarrow)$.

Algorithm 3: Computing the maximal local diameter C
Input: $\mathsf{Glob}, \mathsf{Loc}$
Output: $C := \max\{\zeta(g, g', a, a', \rightsquigarrow) \mid g, g' \in \mathsf{Glob}, a, a' \in \mathsf{Loc}, \rightsquigarrow \subseteq (\mathsf{Glob} \times \mathsf{Loc})^2\}$

Following the involved lemmas, this algorithm can be written out to any level of detail. The next level, i.e., evaluating ζ, is provided by Algorithm 4.

Algorithm 4: Evaluating ζ at a point.
Input: $g, g' \in \mathsf{Glob},\ a, a' \in \mathsf{Loc},\ \rightsquigarrow \subseteq (\mathsf{Glob} \times \mathsf{Loc})^2$
Output: $\zeta(g, g', a, a', \rightsquigarrow)$
if $\exists\, program\ (\to_i)_{i<n}\colon \exists\, i{<}n\colon \rightsquigarrow = \to_i \wedge \mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}((g, n \times \{a\}), i, (g', a')) < \infty$ **then**
  construct a maximal antichain D of such programs with respect to the subprogram preorder;
  return $\max\{\mathsf{d}^{\mathsf{loc}}_{(\rightarrowtail_i)_{i<n}}((g, n \times \{a\}), i, (g', a')) \mid n \in \mathbb{N}_+ \wedge i < n \wedge \rightarrowtail_i = \rightsquigarrow \wedge \mathsf{d}^{\mathsf{loc}}_p((g, n \times \{a\}), i, (g', a')) < \infty \wedge \exists\, p \in \mathrm{D}\colon (\rightarrowtail_i)_{i<n}\ is\ a\ subprogram\ of\ p\}$
else return 0

Algorithm 4 always terminates. Due to potentially unboundedly large antichains, its running time (and, consequently, the running time of Algorithm 3) is not known to have an elementary upper bound, even if we implement the algorithm in such a way that programs with the smallest number of threads (whenever there is a choice) enter the antichain.

Although this paper does not aim to determine the numerical values of C given numerical values of G and L, three remarks are to be made. First, "explicit" means that our algorithm computing C can be easily converted to a runnable implementation in some real-world programming language (although actually obtaining the numerical representation of C in acceptable time would require more effort); this is strictly better than a pure computability claim (in which case we would know that an algorithm exists but might not know what it is). Second, a general lower bound on C is $GL - 1$: we obtain it by considering a single-threaded program whose transition relation is $\{((g,l), (g, l{+}1)) \in (\mathbb{N}_{\geqslant 0}{\times}\mathbb{N}_{\geqslant 0})^2 \mid g{<}G \wedge l{+}1 < L\} \cup \{((g, L{-}1), (g{+}1, 0)) \mid g \in \mathbb{N}_{\geqslant 0} \wedge g{+}1 < G\}$. Third, a further lower bound stems from the example N3T9 in § III.3: from $\mathsf{diam}^{\mathsf{loc}}(\text{N3T9}) = 9$ we obtain that for $G{=}L{=}2$ we have $C{\geqslant}9$.

V. Diameter

Now we provide a lower and an upper bound on diamax, an upper bound on the diameter of programs from a particular class, and an upper bound on the diameter of a randomly chosen program.

V.1. A lower bound on diamax

Within this section, without loss of generality, let the elements of Glob be $0, \ldots, G{-}1$ and the elements of Loc be $0, \ldots, L{-}1$.

We start with a few special cases.

Lemma V.1.1. *If* $G{=}1{\leqslant}L$*, then* $\mathsf{diamax}(n) = (L{-}1)n$ *for all* $n \in \mathbb{N}_{+}$.

Proof. Assume $G{=}1{\leqslant}L$. Let $n \in \mathbb{N}_{\geqslant 0}$. We split $\mathsf{diamax}(n) \stackrel{!}{=} (L{-}1)n$ into two inequalities.

"$\leqslant$": Take a walk σ of an n-threaded program that realizes the distance $\mathsf{diamax}(n)$, i.e., such that $\mathsf{length}(\sigma) = \mathsf{diamax}(n)$ and σ is a shortest path between its end nodes. The shared state stays constant throughout σ, so, if somewhere in σ two neighboring thread-transitions of different threads are applied, it is always possible to change the order of applying these thread transitions such that the length of the walk and the overall effect of these two thread transitions are retained. With finitely many exchanges of this kind, one can obtain a walk σ' such that, for all i, j with $i{<}j{<}n$, the thread transitions of thread i are used before the thread transitions of thread j, and the final state of σ' is the same as the final state of σ. In σ', the number of used thread transitions of each thread does not exceed $L{-}1$ (otherwise σ' would have a self-intersection, and throwing the loop out would result in a shorter walk between the same end-nodes, contradicting the fact that σ is a shortest walk between its ends). Then $\mathsf{length}(\sigma') \leqslant (L{-}1)n$. Thus, $\mathsf{length}(\sigma) \leqslant (L{-}1)n$.

"$\geqslant$": Consider the n-threaded program $(\to_i)_{i<n}$ such that

$$\to_i \;\stackrel{\text{def}}{=}\; \Big\{\big((0,l),(0,l{+}1)\big) \;\Big|\; l{+}1 < L\Big\} \qquad (i{<}n).$$

Each program transition increases some local component of a state by 1 and leaves the other components unchanged. Thus, traveling from $(0, n{\times}\{0\})$ to $(0, n \times \{L{-}1\})$ takes $(L{-}1)n$ single steps. Therefore, $\mathsf{diamax}(n) \geqslant (L{-}1)n$. ∎

Lemma V.1.2. *If* $L{=}1{\leqslant}G$*, then* $\mathsf{diamax}(n) = G{-}1$ *for all* $n \in \mathbb{N}_{+}$.

Proof. Assume $L{=}1{\leqslant}G$. Let $n \in \mathbb{N}_{+}$. We split $\mathsf{diamax}(n) \stackrel{!}{=} G{-}1$ into two inequalities.

"$\leqslant$": Take a walk σ of an n-threaded program that realizes the distance $\mathsf{diamax}(n)$, i.e., such that $\mathsf{length}(\sigma) = \mathsf{diamax}(n)$ and σ is a shortest path between its end points. All the local states stay constant throughout σ, so inside σ only the shared part changes. There are exactly G shared states; any walk with more than G shared parts would have a self-intersection and could be shortened. Thus, each shared state is used at most once in σ, implying $\mathsf{length}(\sigma) < G$.

"$\geqslant$": Consider an n-threaded program $(\to_i)_{i<n}$ such that

$$\to_0 \;\stackrel{\text{def}}{=}\; \big\{\, \big((g,0),(g{+}1,0)\big) \;\big|\; g{+}1 < G \,\big\} \quad \text{and} \quad \to_i \;\stackrel{\text{def}}{=}\; \emptyset \;\text{ for }\; i \in n \setminus \{0\}\,.$$

Then $\mathsf{d}\Big(\big(0, n{\times}\{0\}\big), \big(G{-}1, n{\times}\{0\}\big)\Big) = G{-}1$. ∎

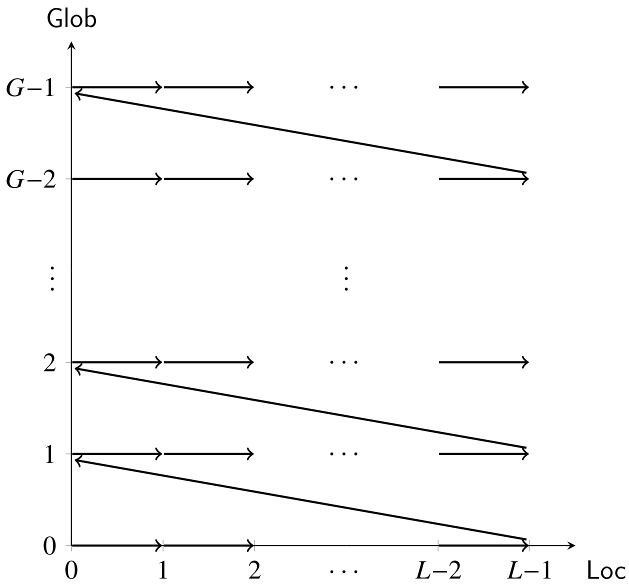
Theorem V.1.3. $\forall\, n{\in}\mathbb{N}_+ :\ \mathsf{diamax}(n) \geqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L.$

Proof. The cases $L{=}1$ or $G{=}1$ have been treated in Lemmas V.1.1 and V.1.2. From now on, consider the case $G, L \geqslant 2$.

Fix $n{\geqslant}1$, and consider the following n-threaded program. We define the set of transitions of thread 0 as

$$\begin{aligned} &\{\, ((g,l),(g,l{+}1)) \mid g < G \wedge l{+}1 < L \,\} && (5)\\ \cup\ &\{\, ((g,L{-}1),(g{+}1,0)) \mid g{+}1 < G \,\}\,. && (6) \end{aligned}$$

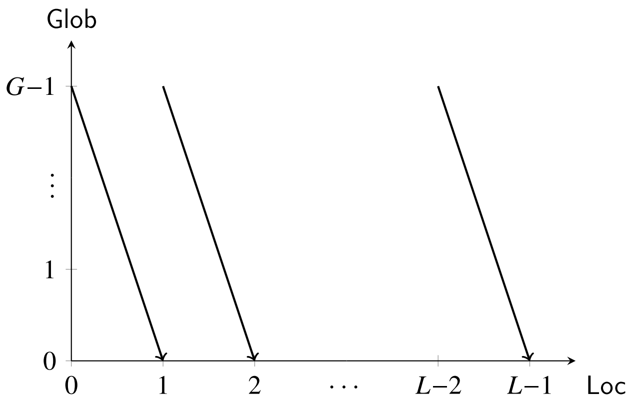
For example, for $G{\geqslant}5$ and $L{\geqslant}5$ these thread transitions can be visualized as follows:



Let the transitions of each thread with index from $n\backslash\{0\}$ be

$$\{((G{-}1,l),(0,l{+}1)) \mid l{+}1 < L\}\,. \qquad (7)$$

For example, for $G{\geqslant}3$ and $L{\geqslant}5$ these thread transitions can be visualized as follows:



Note that for each $i \in \mathbb{N}_{\geqslant 0}$ there is a unique pair $(k, m) \in \mathbb{N}^2_{\geqslant 0}$ such that $i = k((G{-}1)L{+}1){+}m$ and $m \leqslant (G{-}1)L$. Moreover, $G{\geqslant}2$ implies $(\mathbb{N}_{\geqslant 0}{\cap}[0, (G{-}1)L]) = \big(\mathbb{N}_{\geqslant 0}{\cap}[0, L[\big) \mathbin{\dot\cup} \big(\mathbb{N}_{\geqslant 0} \cap [L, (G{-}1)L]\big)$. Using these properties, we define sets $D_i \subseteq \mathsf{State}$ for $i \in \mathbb{N}_{\geqslant 0}$ as follows. For $k \in \mathbb{N}_{\geqslant 0}$ and $m{<}L$, let

$$D_{k((G-1)L+1)+m} \stackrel{\text{def}}{=\!=} \tag{8}$$
$$\left\{(0, l) \in \mathsf{State} \;\middle|\; l_0 = m \;\wedge\; \sum_{t=1}^{n-1} l_t = k\right\}$$

$$\dot\cup \left\{(G{-}1, l) \in \mathsf{State} \;\middle|\; l_0 = m{+}1 \;\wedge\; \sum_{t=1}^{n-1} l_t = k{-}1\right\}, \tag{9}$$

and, for $k, m \in \mathbb{N}_{\geqslant 0}$ such that $L \leqslant m \leqslant (G{-}1)L$, let

$$D_{k((G-1)L+1)+m} \stackrel{\text{def}}{=\!=} \left\{\left(\left\lfloor \frac{m}{L} \right\rfloor, l\right) \in \mathsf{State} \;\middle|\; l_0 = (m \bmod L) \wedge \sum_{t=1}^{n-1} l_t = k\right\}. \tag{10}$$

As a preparatory step, we claim that these sets are disjoint, i.e., $D_j \cap D_i$ is empty for different $i, j \in \mathbb{N}_{\geqslant 0}$. To prove the claim, let $i, j \in \mathbb{N}_{\geqslant 0}$ and $(g, l) \in D_i \cap D_j$ be arbitrary. We are going to show that i and j coincide. Dividing i and j with remainder by $(G{-}1)L + 1$, we obtain quotients $k, k' \in \mathbb{N}_{\geqslant 0}$ and remainders $m, m' \in \mathbb{N}_{\geqslant 0}$ such that $i = k((G{-}1)L + 1) + m$, $j = k'((G{-}1)L + 1) + m'$, and $m, m' \leqslant (G{-}1)L$.

Case $m, m' < L$. Then (g, l) is in the set (8) or (9).

Case $g = 0$ and $m = l_0 = m'$ and $k = \sum_{t=1}^{n-1} l_t = k'$. Then $k((G{-}1)L + 1) + m = k'((G{-}1)L + 1) + m'$, and, therefore, $i = j$.

Case $0 = g = G{-}1$. This would imply $G{=}1$. ↯

Case $g = G{-}1$ and $m{+}1 = l_0 = m'{+}1$ and $k{-}1 = \sum_{t=1}^{n-1} l_t = k'{-}1$. Then $m{=}m'$ and $k{=}k'$. Therefore, $i{=}j$.

Case $m < L \leqslant m'$. By (10), $g = \left\lfloor \frac{m'}{L} \right\rfloor$ and $l_0 = (m' \bmod L)$. From $m'{\geqslant}L$ we obtain $g{>}0$. By (9), $g = G{-}1$ and $l_0 = m{+}1$. Then $m' \geqslant (G{-}1)L$ and $m{+}1 = (m' \bmod L)$. The choice of m' implies $m' = (G{-}1)L$. Therefore, $(m' \bmod L) = 0$. Thus, $m{+}1 = 0$. ↯

Case $m' < L \leqslant m$. By (10), $g = \left\lfloor \frac{m}{L} \right\rfloor$ and $l_0 = (m \bmod L)$. From $m{\geqslant}L$ we obtain $g{>}0$. By (9), $g = G{-}1$ and $l_0 = m'{+}1$. Then $m \geqslant (G{-}1)L$ and $m'{+}1 = (m \bmod L)$. The choice of m implies $m = (G{-}1)L$. Therefore, $(m \bmod L) = 0$. Thus, $m'{+}1 = 0$. ↯

Case $L \leqslant m, m'$. By (10), $\left\lfloor \frac{m}{L} \right\rfloor = g = \left\lfloor \frac{m'}{L} \right\rfloor$, $(m \bmod L) = l_0 = (m' \bmod L)$, and $k = \sum_{t=1}^{n-1} l_t = k'$. Then $m = L\left\lfloor \frac{m}{L} \right\rfloor + (m \bmod L) = L\left\lfloor \frac{m'}{L} \right\rfloor + (m' \bmod L) = m'$ and $k = k'$. Therefore, $i = j$.

Since i, j, and (g, l) were arbitrary, we have shown that the sets D_i for $i \in \mathbb{N}_{\geqslant 0}$ are pairwise disjoint:

$$\forall\, i, j \in \mathbb{N}_{\geqslant 0}\colon \quad D_i \cap D_j \neq \emptyset \;\Rightarrow\; i = j\,. \tag{11}$$

Now, consider the map

$$\mathsf{depth}\colon\; \mathsf{State} \;\to\; \mathbb{N}_{\geqslant 0} \mathbin{\dot\cup} \{\infty\}, \quad (g, l) \;\mapsto\; \mathsf{d}\Big((0,\, n{\times}\{0\}),\; (g, l)\Big)$$

providing the distance of a state from $(0, n{\times}\{0\})$. We claim that the sets D_i and $\mathsf{depth}^{-1}(\{i\})$ are equal ($i \in \mathbb{N}_{\geqslant 0}$) and will prove this claim by natural induction on

i. So let $i \in \mathbb{N}_{\geqslant 0}$ be arbitrary, and assume (as our induction hypothesis) $\forall\, j{<}i\colon D_j = \mathsf{depth}^{-1}(\{j\})$. There is a unique pair $(k, m) \in \mathbb{N}^2_{\geqslant 0}$ such that $i = k((G{-}1)L + 1) + m$ and $m \leqslant (G{-}1)L$.

Case $i{=}0$. We have $D_0 = \{(0, n{\times}\{0\})\} = \mathsf{depth}^{-1}(\{0\})$.

Case $i{\geqslant}1$. We will prove $D_i \overset{!}{=} \mathsf{depth}^{-1}(\{i\})$ by showing the left inclusion and the right inclusion separately.

"$\subseteq$". Let $(g, l) \in D_i$. From (11) we obtain $(g, l) \notin D_j$ for all $j{<}i$. The induction hypothesis implies

$$\forall\, j < i\colon \quad (g, l) \notin \mathsf{depth}^{-1}(\{j\})\,. \tag{12}$$

As the next step, we are going to show that $\mathsf{depth}(g, l)$ does not exceed i. According to the definition of D_i, one of the following situations must hold:

Case $(g, l) \in D_i$ due to (8), i.e., $m{<}L \wedge g{=}0 \wedge l_0{=}m \wedge \sum_{t=1}^{n-1} l_t = k$.

Case $m{=}0$. Since $i{\geqslant}1$, we must have $k{>}0$. From $\sum_{t=1}^{n-1} l_t = k$ we obtain some $\hat{t}$ such that $1{\leqslant}\hat{t}{<}n$ and $l_{\hat{t}}{>}0$. Let $\hat{l} = l[\hat{t} \mapsto l_{\hat{t}}{-}1]$. Note that $G{-}1 = \left\lfloor \frac{(G-1)L}{L} \right\rfloor$, $\hat{l}_0 = 0 = (((G{-}1)L) \bmod L)$, and $\sum_{t=1}^{n-1} \hat{l}_t = k{-}1$. By (10), $(G{-}1, \hat{l}) \in D_{(k-1)((G-1)L+1)+(G-1)L} = D_{i-1}$. By the induction hypothesis, $\mathsf{depth}(G{-}1, \hat{l}) = i{-}1$. By (7), $(G{-}1, \hat{l}) \longrightarrow (0, l)$.

Case $m{\geqslant}1$. Let $\hat{l} = l[0 \mapsto m{-}1]$. From (8) we obtain $(0, \hat{l}) \in D_{k((G-1)L+1)+m-1}$. The induction hypothesis implies $\mathsf{depth}(0, \hat{l}) = i{-}1$. By (5), $(0, \hat{l}) \longrightarrow (0, l)$.

In both cases, $\mathsf{depth}(g, l) \leqslant i$.

Case $(g, l) \in D_i$ due to (9), i.e., $m{<}L \wedge g = G{-}1 \wedge l_0 = m{+}1 \wedge \sum_{t=1}^{n-1} l_t = k{-}1$. Let $\hat{l} = l[0{\mapsto}m]$.

Case $m{=}0$. Note that $G{-}1 = \left\lfloor \frac{(G-1)L}{L} \right\rfloor$, $\hat{l}_0 = 0 = \Big(((G{-}1)L) \bmod L\Big)$, and $\sum_{t=1}^{n-1} \hat{l}_t = k{-}1$. According to (10), $(G{-}1, \hat{l}) \in D_{(k-1)((G-1)L+1)+(G-1)L} = D_{i-1}$.

Case $m{\geqslant}1$. Then $\hat{l}_0 = \underbrace{m-1}_{0\leqslant\cdots<L} {+}1$ and $\sum_{t=1}^{n-1} \hat{l}_t = k{-}1$. By (9), $(G{-}1, \hat{l}) \in D_{k((G-1)L+1)+m-1} = D_{i-1}$.

The induction hypothesis implies $\mathsf{depth}(G{-}1, \hat{l}) = i{-}1$. By (5), $(G{-}1, \hat{l}) \longrightarrow (g, l)$. Therefore, $\mathsf{depth}(g, l) \leqslant i$.

Case $(g, l) \in D_i$ due to (10), i.e., $L{\leqslant}m \wedge g = \left\lfloor \frac{m}{L} \right\rfloor \wedge l_0 = (m \bmod L) \wedge \sum_{t=1}^{n-1} l_t = k$. Note that $g{\geqslant}1$.

Case $l_0{=}0$. Then $m = gL$. Let $\hat{g} = g{-}1$ and $\hat{l} = l[0{\mapsto}L{-}1]$.

Case $\hat{g}{=}0$. Then $(\hat{g}, \hat{l}) = (0, \hat{l}) \overset{\text{by (8)}}{\in} D_{k((G-1)L+1)+L-1} = D_{k((G-1)L+1)+gL-1}$.

Case $\hat{g}{\geqslant}1$. Note that $\hat{g} = g + \left\lfloor -\frac{1}{L} \right\rfloor = \left\lfloor g - \frac{1}{L} \right\rfloor = \left\lfloor \frac{gL-1}{L} \right\rfloor$, $\hat{l}_0 = L{-}1 = ((g{-}1)L + L - 1) \bmod L) = ((gL - 1) \bmod L)$, $\sum_{t=1}^{n-1} \hat{l}_t = k$, and $gL - 1 = (\hat{g}{+}1)L - 1 \geqslant 2L - 1 \geqslant L$. By (10), $(\hat{g}, \hat{l}) \in D_{k((G-1)L+1)+gL-1}$.

From $k((G{-}1)L + 1) + gL - 1 = i{-}1$ we obtain $(\hat{g}, \hat{l}) \in D_{i-1}$. The induction hypothesis implies $\mathsf{depth}(\hat{g}, \hat{l}) = i{-}1$. By (6), $(\hat{g}, \hat{l}) \longrightarrow (g, l)$.

Case $l_0 \geqslant 1$. Let $\hat{l} = l[0 \mapsto l_0-1]$. From $m = L\lfloor\frac{m}{L}\rfloor + \underbrace{(m \bmod L)}_{0<\cdots<L}$ we obtain $m-1 = L\lfloor\frac{m}{L}\rfloor + \underbrace{(m \bmod L)-1}_{0\leqslant\cdots<L}$. Due to the uniqueness of the quotient and the remainder, $\lfloor\frac{m-1}{L}\rfloor = \lfloor\frac{m}{L}\rfloor$ and $((m-1) \bmod L) = (m \bmod L) - 1$. Thus, $g = \lfloor\frac{m-1}{L}\rfloor$ and $\hat{l}_0 = ((m-1) \bmod L)$. Moreover, $\sum_{t=1}^{n-1}\hat{l}_t = k$. From $(m \bmod L) > 0$ and $m \geqslant L$ we obtain $m>L$ and therefore $m-1 \geqslant L$. By (10), $(g,\hat{l}) \in D_{k((G-1)L+1)+m-1} = D_{i-1}$. The induction hypothesis implies $\mathsf{depth}(g,\hat{l}) = i-1$. By (5), $(g,\hat{l}) \longrightarrow (g,l)$.

In both cases we obtain $\mathsf{depth}(g,l) \leqslant i$.

We have shown $\mathsf{depth}(g,l) \leqslant i$. By (12), $\mathsf{depth}(g,l) = i$.

"$\supseteq$". Let $(g,l) \in \mathsf{depth}^{-1}(\{i\})$ be arbitrary. There is some $(\hat{g},\hat{l}) \in \mathsf{depth}^{-1}(\{i-1\})$ such that $(\hat{g},\hat{l}) \longrightarrow (g,l)$. In particular, there is some $\hat{t}<n$ such that $(\hat{g},\hat{l}_{\hat{t}}) \rightarrow_{\hat{t}} (g,l_{\hat{t}})$ and $\forall\, t \in n \setminus \{\hat{t}\}\colon \hat{l}_t = l_t$. The induction hypothesis implies $(\hat{g},\hat{l}) \in D_{i-1}$. We distinguish the following cases.

Case m=0. We obtain $(\hat{g},\hat{l}) \in D_{(k-1)((G-1)L+1)+(G-1)L}$. Since $G \geqslant 2$, we have $L \leqslant (G-1)L$. By (10), $\hat{g} = G-1$, $\hat{l}_0 = 0$, and $\sum_{t=1}^{n-1}\hat{l}_t = k-1$. We consider all the choices of the thread $\hat{t}$ and the corresponding thread transition that made the step:

Case (5), i.e., $\hat{t}=0 \wedge \hat{g} = g \wedge \hat{l}_0+1 = l_0$. Then $g = G-1$, $l_0=1$, and $\sum_{t=1}^{n-1}l_t = k-1$. By (9), $(g,l) \in D_{k((G-1)L+1)}$.

Case (6), i.e., $\hat{t}=0 \wedge \hat{g}+1 = g \wedge \hat{l}_0 = L-1 \wedge l_0 = 0$. However, above we discovered $\hat{l}_0 = 0$. So $0 = L-1$, contradicting $L \geqslant 2$.

Case (7), i.e., $\hat{t} \geqslant 1 \wedge \hat{g} = G-1 \wedge \hat{l}_{\hat{t}}+1 = l_{\hat{t}} \wedge g = 0$. Then $l_0 = 0$ and $\sum_{t=1}^{n-1}l_t = k$. By (8), $(g,l) \in D_{k((G-1)L+1)}$.

Case $0<m<L$. We obtain $(\hat{g},\hat{l}) \in D_{k((G-1)L+1)+m-1}$ with $0 \leqslant m-1 < L$; this can happen only due to (8) or (9):

Case (8), i.e., $\hat{g} = 0 \wedge \hat{l}_0 = m-1 \wedge \sum_{t=1}^{n-1}\hat{l}_t = k$. We consider all the choices of the thread $\hat{t}$ and the corresponding thread transition that made the step:

Case (5), i.e., $\hat{t}=0 \wedge \hat{g} = g \wedge \hat{l}_0+1 = l_0$. Then $g = 0$, $l_0 = m$, and $\sum_{t=1}^{n-1}l_t = k$. By (8), $(g,l) \in D_{k((G-1)L+1)+m}$.

Case (6), i.e., $\hat{t} = 0 \wedge \hat{g}+1 = g \wedge \hat{l}_0 = L-1 \wedge l_0 = 0$. Then $m-1 = L-1$, implying m=L, which contradicts the choice of $m<L$ in this case of the case split.

Case (7), i.e., $\hat{t} \geqslant 1 \wedge \hat{g} = G-1 \wedge \hat{l}_{\hat{t}}+1 = l_{\hat{t}} \wedge g = 0$. Then $0 = \hat{g} = G-1$, contradicting $G \geqslant 2$.

Case (9), i.e., $\hat{g} = G-1 \wedge \hat{l}_0 = m \wedge \sum_{t=1}^{n-1}\hat{l}_t = k-1$. We consider all the choices of the thread $\hat{t}$ and the corresponding thread transition that made the step:

Case (5), i.e., $\hat{t}=0 \wedge \hat{g} = g \wedge \hat{l}_0+1 = l_0$. Then $g = G-1$, $l_0 = m+1$, and $\sum_{t=1}^{n-1}l_t = k-1$. By (9), $(g,l) \in D_{k((G-1)L+1)+m}$.

Case (6), i.e., $\hat{t}=0 \wedge \hat{g}+1 = g \wedge \hat{l}_0 = L-1 \wedge l_0 = 0$. Then $G = g$. $\lightning$

Case (7), i.e., $\hat{t} \geqslant 1 \wedge \hat{g} = G-1 \wedge \hat{l}_{\hat{t}}+1 = l_{\hat{t}} \wedge g = 0$. Then $l_0 = m$ and $\sum_{t=1}^{n-1}l_t = k$. By (8), $(g,l) \in D_{k((G-1)L+1)+m}$.

Case m=L. We obtain $(\hat{g},\hat{l}) \in D_{k((G-1)L+1)+m-1}$ where, due to $L \geqslant 2$,

$0 < m{-}1 < L$. Moreover, $(m{-}1) + 1 = L > \hat{l}_0$. By (9), $(G{-}1, \hat{l}) \notin D_{k((G-1)L+1)+m-1}$, and so $\hat{g} \neq G{-}1$. By (8), $\hat{g} = 0$, $\hat{l}_0 = L{-}1$, and $\sum_{t=1}^{n-1} \hat{l}_t = k$. We consider all the choices of the thread $\hat{t}$ and the corresponding thread transition that made the step:

Case (5), i.e., $\hat{t}{=}0 \wedge \hat{g} = g \wedge \hat{l}_0{+}1 = l_0$. Then $l_0 = L$. $\lightning$

Case (6), i.e., $\hat{t}{=}0 \wedge \hat{g}{+}1 = g \wedge \hat{l}_0 = L{-}1 \wedge l_0 = 0$. Then $g = 1 = \left\lfloor \frac{m}{L} \right\rfloor$, $l_0 = 0 = (m \bmod L)$, and $\sum_{t=1}^{n-1} l_t = k$. By (10), $(g, l) \in D_{k((G-1)L+1)+m}$.

Case (7), i.e., $\hat{t}{\geqslant}1 \wedge \hat{g} = G{-}1 \wedge \hat{l}_{\hat{t}}{+}1 = l_{\hat{t}} \wedge g = 0$. Then $0 = G{-}1$, which contradicts $G{\geqslant}2$.

Case $m{>}L$. We obtain $(\hat{g}, \hat{l}) \in D_{k((G-1)L+1)+m-1}$ where $L \leqslant m{-}1 < (G{-}1)L$. Then $\hat{g} = \left\lfloor \frac{m-1}{L} \right\rfloor$, $\hat{l}_0 = ((m{-}1) \bmod L)$, and $\sum_{t=1}^{n-1} \hat{l}_t = k$. In particular, $\hat{g}{\geqslant}1$. We consider all the choices of the thread $\hat{t}$ and the corresponding thread transition that made the step:

Case (5), i.e., $\hat{t}{=}0 \wedge \hat{g} = g \wedge \hat{l}_0{+}1 = l_0$. Note that $m = (m{-}1) + 1 = \left(L\left\lfloor \frac{m-1}{L} \right\rfloor + \big((m{-}1) \bmod L\big)\right) + 1 = L\hat{g} + \hat{l}_0 + 1 = Lg + l_0$. From $l_0{<}L$ and the uniqueness of the quotient with the remainder we obtain $g = \left\lfloor \frac{m}{L} \right\rfloor$ and $l_0 = (m \bmod L)$. Finally, $\sum_{t=1}^{n-1} l_t = k$. By (10), $(g, l) \in D_{k((G+1)L+1)+m}$.

Case (6), i.e., $\hat{t}{=}0 \wedge \hat{g}{+}1 = g \wedge \hat{l}_0 = L{-}1 \wedge l_0 = 0$. Then $m = (m{-}1) + 1 = \left(L\left\lfloor \frac{m-1}{L} \right\rfloor + \big((m{-}1) \bmod L\big)\right) + 1 = (L\hat{g} + \hat{l}_0) + 1 = L\hat{g} + L - 1 + 1 = L(\hat{g}{+}1) = Lg$. The uniqueness of the quotient with the remainder implies $\left\lfloor \frac{m}{L} \right\rfloor = g$ and $(m \bmod L) = 0 = l_0$. Finally, $\sum_{t=1}^{n-1} l_t = k$. By (10), $(g, l) \in D_{k((G+1)L+1)+m}$.

Case (7), i.e., $\hat{t}{\geqslant}1 \wedge \hat{g} = G{-}1 \wedge \hat{l}_{\hat{t}}{+}1 = l_{\hat{t}} \wedge g{=}0$. Then $G{-}1 = \left\lfloor \frac{m-1}{L} \right\rfloor$. Since $m{-}1 < (G{-}1)L$, we must have $\frac{m-1}{L} < G{-}1$, and therefore $\left\lfloor \frac{m-1}{L} \right\rfloor < G{-}1$. $\lightning$

We have now shown that, in every noncontradictory case, $(g, l) \in D_{k((G-1)L+1)+m} = D_i$.

We have proven

$$\forall\, i \in \mathbb{N}_{\geqslant 0}\colon\ D_i = \mathsf{depth}^{-1}(\{i\})\,.$$

For

$$\hat{d} \overset{\text{def}}{=\!=} \big((n{-}1)(L{-}1) + 1\big)\big((G{-}1)L + 1\big) + L - 2 \tag{13}$$

we have $(G{-}1, n \times \{L{-}1\}) \in D_{\hat{d}}$ by (9). Therefore, $\hat{d} = \mathsf{depth}\big(G{-}1, n \times \{L{-}1\}\big) = \mathsf{d}\Big(\big(0, n{\times}\{0\}\big), \big(G{-}1, n{\times}\{L{-}1\}\big)\Big) \leqslant \mathsf{diamax}(n)$. Note that $\hat{d} = (n{-}1)(L{-}1)((G{-}1)L{+}1){+}(G{-}1)L{+}1{+}L{-}2 = (n{-}1)(L{-}1)(GL{-}L{+}1){+}GL{-}L{+}L{-}1 = n(L{-}1)(GL{-}L{+}1){-}(L{-}1)(GL{-}L{+}1){+}GL{-}1 = n(L{-}1)(GL{-}L{+}1){-}(GL^2{-}L^2{+}L{-}GL{+}L{-}1){+}GL{-}1 = n(L{-}1)(GL - L + 1) - GL^2 + L^2 + GL - 2L + 1 + GL - 1 = n(L{-}1)(GL - L + 1) - GL^2 + L^2 + 2GL - 2L = n(L{-}1)(GL - L + 1) + (-L(G{-}1) + 2(G{-}1))L = n(L{-}1)(GL - L + 1) + (2{-}L)(G{-}1)L$. ■

Using Knuth's Big-Omega notation [50], we obtain: $L{\geqslant}2 \Rightarrow \mathsf{diamax}(n) = \Omega(n)$.

The proof of Theorem V.1.3 cannot be strengthened by better counting using the same program family:

Note V.1.4. The above proof computed the distance $\hat{d}$ between two states of a particular n-threaded program ($n \geqslant 1$). It is worth asking whether there are states at a larger finite distance in this program. Now we show that the proof has reached its own limit, i.e., that it is impossible to obtain any larger finite distance in the transition graph of this program.

To this end, we first claim that, in the context of the above proof, each successor of each state in D_i lies in D_{i+1} ($i \in \mathbb{N}_{\geqslant 0}$). To prove this, let $i \in \mathbb{N}_{\geqslant 0}$ and $(g,l) \in D_i$ be arbitrary. Division with remainder gives us $k \in \mathbb{N}_{\geqslant 0}$ and $m \leqslant (G-1)L$ such that $i = k((G-1)L+1)+m$. Let (g',l') be an arbitrary successor of (g,l). There is some $j < n$ such that $(g,l_j) \to_j (g',l'_j)$ and $\forall \hat{\jmath} \in n \setminus \{j\}\colon l_{\hat{\jmath}} = l'_{\hat{\jmath}}$. We consider three cases concerning how $(g,l) \in D_i$ originated:

Case (8), i.e., $m{<}L \wedge g = 0 \wedge l_0 = m \wedge \sum_{t=1}^{n-1} l_t = k$. Since only thread 0 has thread transitions starting with the shared state 0, we must have $j{=}0$. Thus, $l'_t = l_t$ for $t \in \mathbb{N}_+ \cap [1,n[$, and so $\sum_{t=1}^{n-1} l'_t = k$. Thread 0 has transitions of two kinds:

Case (5), i.e., $g' = g \wedge l'_0 = l_0{+}1$. Then $g'{=}0$ and $m{+}1 = l'_0 < L \leqslant (G-1)L$. By (8), $(g',l') \in D_{k((G-1)L+1)+m+1}$.

Case (6), i.e., $g' = g{+}1 \wedge l_0 = L{-}1 \wedge l'_0 = 0$. From $g' = 1$ and $m{+}1 = l_0{+}1 = L \leqslant (G-1)L$ we obtain $g' = \lfloor \frac{m+1}{L} \rfloor$, $l'_0 = ((m{+}1) \bmod L)$. By (10), $(g',l') \in D_{k((G-1)L+1)+m+1}$.

In both cases above, $(g',l') \in D_{k((G-1)L+1)+m+1} = D_{i+1}$.

Case (9), i.e., $m{<}L \wedge g = G{-}1 \wedge l_0 = m{+}1 \wedge \sum\limits_{t=1}^{n-1} l_t = k{-}1$. Since $g+1 = G$, the transition (6) of thread 0 cannot be taken from (g,l), and so there may be only two cases for j and the thread transition taken:

Case (5), i.e., $j{=}0 \wedge g' = g \wedge l'_0 = l_0{+}1$. From $g' = G{-}1$, $(m{+}1){+}1 = l'_0 \leqslant L \leqslant (G{-}1)L$, and $\sum_{t=1}^{n-1} l'_t = \sum_{t=1}^{n-1} l_t = k-1$ we obtain $(g',l') \in D_{k((G-1)L+1)+m+1}$ by (9).

Case (7), i.e., $j{\geqslant}1 \wedge g' = 0 \wedge l'_j = l_j{+}1$. From $m{+}1 = l'_0 < L$ and $\sum_{t=1}^{n+1} l'_t = 1 + \sum_{t=1}^{n+1} l_t = k$ we obtain $(g',l') \in D_{k((G-1)L+1)+m+1}$ by (8).

In both cases above, $(g',l') \in D_{k((G-1)L+1)+m+1} = D_{i+1}$.

Case (10), i.e., $m{\geqslant}L \wedge g = \lfloor \frac{m}{L} \rfloor \wedge l_0 = (m \bmod L) \wedge \sum_{t=1}^{n-1} l_t = k$. All three cases for j and the transition taken are possible:

Case (5), i.e., $j{=}0 \wedge g = g' \wedge l'_0 = l_0{+}1$. From $j = 0$ we obtain $\sum_{t=1}^{n-1} l'_t = k$. We distinguish two subcases:

Case $m < (G{-}1)L$. So $L < m{+}1 \leqslant (G{-}1)L$. Note that $m{+}1 = \Big(L\lfloor \frac{m}{L} \rfloor + (m \bmod L)\Big) + 1 = Lg + l_0 + 1 = Lg' + l'_0$. From $0 \leqslant l'_0 < L$ and the uniqueness of the quotient with the remainder, we obtain $g' = \lfloor \frac{m+1}{L} \rfloor$ and $l'_0 = \big((m{+}1) \bmod L\big)$. By (10), $(g',l') \in D_{k((G-1)L+1)+m+1}$.

Case $m = (G{-}1)L$. Then $g = G-1 \wedge l_0 = 0$. So $g' = G-1 \wedge l'_0 = 1$. For $k' = k{+}1$ and $m' = 0$ we obtain $l'_0 = m'{+}1 \wedge \sum_{t=1}^{n-1} l'_t = k'{-}1$. By (9), $(g',l') \in D_{k'((G-1)L+1)+m'}$. Note that $k'((G{-}1)L+1)+m' = (k{+}1)((G{-}1)L{+}1){+}0 = k((G{-}1)L{+}1){+}(G{-}1)L{+}1 = k((G{-}1)L{+}1)+m+1$. Thus, $(g',l') \in D_{k((G-1)L+1)+m+1}$.

In both cases above, $(g', l') \in D_{k((G-1)L+1)+m+1} = D_{i+1}$.

Case (6), i.e., $j{=}0 \wedge g{+}1 = g' \wedge l_0 = L{-}1 \wedge l'_0 = 0$. Since $(m \bmod L) = L{-}1 \geqslant 1$, we obtain that m cannot be divisible by L. So $m < (G{-}1)L$. Let $m' = m{+}1$; then $m' \leqslant (G{-}1)L$. Note that $m' = m{+}1 = \left(L\left\lfloor \frac{m}{L} \right\rfloor + (m \bmod L)\right) + 1 = (Lg + l_0) + 1 = Lg + L - 1 + 1 = Lg + L = L(g{+}1) = Lg'$. Therefore, $g' = \left\lfloor \frac{m'}{L} \right\rfloor$ and $0 = (m' \bmod L)$. Thus, $l'_0 = (m' \bmod L)$. Moreover, from $j = 0$ we obtain $\sum_{t=1}^{n-1} l'_t = k$. By (10), $(g', l') \in D_{k((G-1)P+1)+m'} = D_{i+1}$.

Case (7), i.e., $j \geqslant 1 \wedge g = G{-}1 \wedge g' = 0 \wedge l'_j = l_j{+}1$. If m were smaller than $(G{-}1)L$, then we would have $\frac{m}{L} < G-1 = g = \left\lfloor \frac{m}{L} \right\rfloor \leqslant \frac{m}{L}$, $\lightning$. Therefore, $m = (G-1)L$. Thus, $l_0 = 0$. Let $m' = 0$ and $k' = k+1$; then $m' = l_0 = l'_0$ and $\sum_{t=1}^{n-1} l'_t = 1 + \sum_{t=1}^{n-1} l_t = k{+}1 = k'$. By (8), $(g', l') \in D_{k'((G-1)L+1)+m'}$ = [since $k'((G-1)L + 1) + m' = k'((G-1)L + 1) = k((G-1)L + 1) + (G{-}1)L + 1 = k((G{-}1)L + 1) + m + 1 = i{+}1$] D_{i+1}.

In all three cases above, $(g', l') \in D_{i+1}$.

We have shown that successors of states of D_i lie in D_{i+1} ($i \in \mathbb{N}_{\geqslant 0}$), or, more formally:

$$\forall\, i \in \mathbb{N}_{\geqslant 0},\ s \in D_i,\ s' \in \mathsf{State}\colon\ s \longrightarrow s' \ \Rightarrow\ s' \in D_{i+1}\,. \tag{14}$$

Second, we claim that all states lie in $\bigcup_{i \in \mathbb{N}_{\geqslant 0}} D_i$. To prove this, let $(g, l) \in \mathsf{State}$ be arbitrary. For each of the following cases, we find $k, m \in \mathbb{N}_{\geqslant 0}$ so that $m \leqslant (G{-}1)L$ and $(g', l') \in D_{k((G-1)L+1)+m}$:

Case $g = 0$. Choose $m = l_0$ and $k = \sum_{t=1}^{n-1} l_t$. Noting that $m < L$, apply (8).

Case $0 < g < G{-}1$. Choose $m = gL + l_0$ and $k = \sum_{t=1}^{n-1} l_t$. Note that $L \leqslant m \leqslant (G{-}2)L + L - 1 = (G{-}1{-}1)L + L - 1 = (G{-}1)L - L + L - 1 = (G{-}1)L - 1 < (G{-}1)L$ and apply (10).

Case $g = G{-}1 \wedge l_0 = 0$. Choose $m = gL$ and $k = \sum_{t=1}^{n-1} l_t$. Note that $L \leqslant m = (G{-}1)L$ and apply (10).

Case $g = G{-}1 \wedge l_0 \geqslant 1$. Choose $m = l_0{-}1$ and $k = 1 + \sum_{t=1}^{n-1} l_t$. Note that $m < L$ and apply (9).

We have shown

$$\mathsf{State} \subseteq \bigcup_{i \in \mathbb{N}_{\geqslant 0}} D_i\,. \tag{15}$$

Third, we obtain $D_{k((G-1)L+1)+m} = \emptyset$ (at least) for the following $(k, m) \in \mathbb{N}^2_{\geqslant 0}$ such that $m \leqslant (G{-}1)L$:

- $k = (L{-}1)(n{-}1) + 1$ and $m = L{-}1$ according to (8) and (9),
- $k = (L{-}1)(n{-}1) + 1$ and $m \geqslant L$ according to (10),
- $k \geqslant (L{-}1)(n{-}1) + 2$ according to (8), (9), and (10).

Therefore:

$$D_i{=}\emptyset \ \text{ for all } \ i \geqslant ((L{-}1)(n{-}1) + 1)((G{-}1)L + 1) + L - 1\,. \tag{16}$$

Now, let $s, s' \in \mathsf{State}$ be arbitrary such that $s \longrightarrow^* s'$. By (15), there are some $i, j \in \mathbb{N}_{\geqslant 0}$ such that $s \in D_i$ and $s' \in D_j$. By (13) and (16), $i, j \leqslant \hat{d}$. By (14), $\mathsf{d}(s, s') = j{-}i \leqslant j \leqslant \hat{d}$. Since s, s' were arbitrary, the diameter of the program is at most $\hat{d}$. Since the proof of Theorem V.1.3 provided us with two states at this distance, $\hat{d}$ is the exact value of the diameter of the program. □

Of course, $(GL-L+1)(L-1)n+(2-L)(G-1)L$ is only a lower bound and in general not the exact value of the diamax function. We already saw in § III.2 examples, such as N2T6B or N2T7A, of diameter $7 > 6 = 3n = (2\cdot2-2+1)(2-1)n + (2-2)(2-1)\cdot2$ for $n = G = L = 2$. Duplicating existing threads in these examples did not raise the diameter above the lower bound. As we will see in Corollary V.2.2.4, adding arbitrarily many arbitrary threads to these examples would not raise the bound beyond $3n + 1$ anyway. A few more exceptions, i.e., n-threaded programs (for small n; in the binary case, for $n \leqslant 4$) for which the diameter exceeds the lower bound, are known. No such exception is known to have generalizations for infinitely many n that could asymptotically improve Theorem V.1.3. Further, on a set of over $6 \cdot 10^{24}$ n-threaded binary programs with $n \geqslant 5$, no deviations from the lower bound have been observed [62].

V.2. Upper bounds

V.2.1. An upper bound on diamax

We are going to show that diamax is asymptotically majorized by a polynomial function.

We start the proof by fixing an arbitrary program $(\rightarrow_i)_{i<n}$ and an arbitrary state $(g, l) \in \mathsf{State}$ of this program until (but not including) Proposition V.2.1.21. We are going to prove that $\mathsf{d}((g, l), s) \overset{!}{=} \mathsf{O}(n^c)$ for all $s \in \mathsf{State}$, where $c > 1$, and neither c nor the constant hidden in the O-notation depend on g, l, s, n, or the program. From a high-level view, our proof will exploit symmetries between the threads.

As a first step, we "confuse" thread indexes if the local parts corresponding to these thread indexes in the initial state are equal and the threads' transition relations are equal up to self-loops. Formally:

Definition V.2.1.1. Consider the *diagonal* $D \overset{\text{def}}{=} \mathsf{id}_{\mathsf{Glob}\times\mathsf{Loc}}$. Fix for each $i<n$ a permutation $\iota_i\colon \mathsf{Loc}\hookrightarrow\!\!\!\!\rightarrow\mathsf{Loc}$ such that $\iota_i(l_i) = l_0$. (For example, for each $i < n$ choose the transposition $\lambda\, a \in \mathsf{Loc}.$ if $a = l_0$ then l_i elif $a = l_i$ then l_0 else a; we call such a choice *standard*.) Let *the* i^{th} *normalized thread transition relation* be $\rightsquigarrow_i \overset{\text{def}}{=} \{((x, \iota_i(y)), (x', \iota_i(y'))) \mid ((x, y), (x', y')) \in \rightarrow_i\!\setminus D\}$ $(i<n)$. Thread identifiers $i, j < n$ are called *confusable*, written $i{\sim}j$, iff $\rightsquigarrow_i = \rightsquigarrow_j$. □

Note that $\sim$ is an equivalence relation on n.

Next, we define which program states should be considered indistinguishable for our purposes: such states result from each other by re-indexing threads with confusable identifiers, provided that the local parts corresponding to these thread identifiers match properly. Formally:

Definition V.2.1.2. A map $\varphi\colon n \rightarrow n$ is called *confusion-invariant*, or shortly *$\sim$-invariant*, iff $\forall\, i<n\colon i \sim \varphi(i)$. Vectors of local states $\hat{l}, \check{l} \in \mathsf{Loc}^n$ of dimension n are called *confusable*, written $\hat{l} \asymp \check{l}$, iff there is a $\sim$-invariant $\varphi \in (n \hookrightarrow\!\!\!\!\rightarrow n)$ such that $\forall\, i<n$: $\iota_i(\hat{l}_i) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})$. Program states $(\hat{g}, \hat{l}), (\check{g}, \check{l})$ are called *confusable*, written $(\hat{g}, \hat{l}) \approx (\check{g}, \check{l})$, iff $\hat{g}{=}\check{g} \wedge \hat{l}{\asymp}\check{l}$. □

Intuitively speaking, φ in the above definition re-indexes threads with confusable identifiers provided the corresponding local parts match properly.

Example V.2.1.3. Consider the binary case $\mathsf{Glob} = \{\alpha, \beta\}$ and $\mathsf{Loc} = \{0, 1\}$ and a two-threaded program $(\{(\alpha0, \alpha1), (\alpha1, \alpha1)\}, \{(\alpha1, \alpha0), (\beta1, \beta1)\})$. Let the initial state be $(g, l) = (\alpha, (0, 1))$. Choose $\iota_0 = \mathsf{id}_{\mathsf{Loc}}$ and $\iota_1 = \lambda\, a \in \mathsf{Loc}.\ 1-a$. Then $\rightsquigarrow_0 = \{(\alpha0, \alpha1)\}$ and $\rightsquigarrow_1 = \{(\alpha0, \alpha1)\}$. So $0{\sim}1$ and $1{\sim}0$. The transposition $\varphi = \lambda\, i{<}2.\ 1{-}i$, which swaps the indexes of the threads, is $\sim$-invariant. Let $(\hat{g}, \hat{l}) = (\beta, (0,0))$ and $(\check{g}, \check{l}) = (\beta, (1,1))$. From $\iota_0(\hat{l}_0) = 0 = 1{-}1 = \iota_1(\check{l}_1) = \iota_{\varphi(0)}(\check{l}_{\varphi(0)})$ and $\iota_1(\hat{l}_1) = 1{-}0 = 1 = \iota_0(\check{l}_0) = \iota_{\varphi(1)}(\check{l}_{\varphi(1)})$ we get $\hat{l} \asymp \check{l}$. As $\hat{g} = \check{g}$, the program states $(\hat{g}, \hat{l})$ and $(\check{g}, \check{l})$ are confusable. Here, $\approx$ is not a classic bisimulation [7, Definition 7.1]: $(\check{g}, \check{l})$ has a successor program state (namely, itself), whereas $(\hat{g}, \hat{l})$ has no successors. □

The identity on thread indexes is $\sim$-invariant:

Lemma V.2.1.4. $\mathsf{id}_n : n \hookrightarrow\!\!\!\!\rightarrow n$ *is $\sim$-invariant.*

Proof. For any $i{<}n$, we have $i \sim i = \mathsf{id}_n(i)$. ■

The inverses of $\sim$-invariant permutations of n are $\sim$-invariant themselves:

Lemma V.2.1.5. *If $\varphi : n \hookrightarrow\!\!\!\!\rightarrow n$ is $\sim$-invariant, so is φ^{-1}.*

Proof. Let $\varphi : n \hookrightarrow\!\!\!\!\rightarrow n$ be $\sim$-invariant.

Let $i{<}n$ be arbitrary. Let $j \overset{\text{def}}{=} \varphi^{-1}(i)$. Then $j \sim \varphi(j)$. Therefore, $i = \varphi(\varphi^{-1}(i)) = \varphi(j)$ [since $\sim$ is symmetric] $j = \varphi^{-1}(i)$.

Summarizing, $i \sim \varphi^{-1}(i)$ for all $i{<}n$. ■

Composing $\sim$-invariant maps yields a $\sim$-invariant map:

Lemma V.2.1.6. *If $\varphi, \psi : n{\longrightarrow}n$ are $\sim$-invariant, so is $\psi \circ \varphi$.*

Proof. For any $i{<}n$, we have $i \sim \varphi(i) \sim \psi(\varphi(i))$ by Definition V.2.1.2, and, therefore, $i \sim \psi(\varphi(i))$. ■

Confusion of n-vectors of local states is an equivalence relation:

Lemma V.2.1.7. *$\asymp$ is an equivalence relation on* Loc^n.

Proof. We show:

"$\asymp$ is reflexive on Loc^n": Due to Lemma V.2.1.4.

"$\asymp$ is symmetric": Let $\hat{l} \asymp \check{l}$. Choose a $\sim$-invariant $\varphi : n{\hookrightarrow\!\!\!\!\rightarrow}n$ such that $\forall\, i < n\colon \iota_i(\hat{l}_i) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})$. Lemma V.2.1.5 implies that φ^{-1} is $\sim$-invariant. Also, $\forall\, i < n$: $\iota_{\varphi^{-1}(\varphi(i))}(\hat{l}_{\varphi^{-1}(\varphi(i))}) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})$. The surjectivity of φ implies that $\forall\, j < n$: $\iota_{\varphi^{-1}(j)}(\hat{l}_{\varphi^{-1}(j)}) = \iota_j(\check{l}_j)$. So $\check{l} \asymp \hat{l}$.

"$\asymp$ is transitive": Let $\hat{l} \asymp \check{l} \asymp \bar{l}$. There are $\sim$-invariant $\varphi, \psi : n{\hookrightarrow\!\!\!\!\rightarrow}n$ such that $\forall\, i < n$: $\iota_i(\hat{l}_i) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})$ and $\forall\, j < n\colon \iota_j(\check{l}_j) = \iota_{\psi(j)}(\bar{l}_{\psi(j)})$. By Lemma V.2.1.6, $\psi{\circ}\varphi$ is $\sim$-invariant. Also, for all $i < n$ we have $\iota_i(\hat{l}_i) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)}) = \iota_{\psi(\varphi(i))}(\bar{l}_{\psi(\varphi(i))})$. Therefore, $\hat{l} \asymp \bar{l}$. ■

To show that confusion of program states is an equivalence relation, we recall a well-known fact:

Lemma V.2.1.8. *Let X and Y be sets, $\fallingdotseq$ an equivalence relation on X, $\risingdotseq$ an equivalence relation on Y, $Z = X{\times}Y$, and $\div$ a binary relation on Z defined via $(x_1, y_1) \div (x_2, y_2) \overset{def}{\Leftrightarrow} x_1 \fallingdotseq x_2 \wedge y_1 \risingdotseq y_2$ ($x_1, x_2 \in X$, $y_1, y_2 \in Y$). Then:*

a) $\div$ is an equivalence relation on Z, and

b) $\left|Z/_{\div}\right| = \left|X/_{\fallingdotseq}\right| \left|Y/_{\risingdotseq}\right|$.

Proof. The definition immediately implies that $\div$ is reflexive on Z, symmetric and transitive, implying part a). The map $\varphi : Z/_{\div} \to X/_{\fallingdotseq} \times Y/_{\risingdotseq}$, $[(x,y)]_{\div} \mapsto ([x]_{\fallingdotseq}, [y]_{\risingdotseq})$ is well defined and a bijection, implying part b). ■

Lemmas V.2.1.7 and V.2.1.8a) imply directly:

Corollary V.2.1.9. $\approx$ *is an equivalence relation on* State. $\square$

For computing distances, the self-loops in the transition graph are irrelevant, which motivates the following notion:

Definition V.2.1.10. A *simple* thread transition relation is a member of $E \overset{\text{def}}{=} \mathfrak{P}\big((\mathsf{Glob} \times \mathsf{Loc})^2 \setminus D\big)$. $\square$

Above, E stands for "*e*dges". Using this shorthand for simple thread transition relations, we can express the notion of state confusion differently; instead of mentioning bijections we can say that certain sets are equally large:

Lemma V.2.1.11. *For all* $\hat{l}, \check{l} \in \mathsf{Loc}^n$, *we have* $\hat{l} \asymp \check{l}$ *iff* $\forall\, a \in \mathsf{Loc}, \rightrightarrows \in E\colon \big|\{t < n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\}\big| = \big|\{t < n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a\}\big|$.

Proof. Let $\hat{l}, \check{l} \in \mathsf{Loc}^n$ be arbitrary. We show the two directions of the aforementioned bi-implication separately.

"$\Rightarrow$": We assume the left-hand side $\hat{l} {=} \check{l}$. Then a $\sim$-invariant $\varphi\colon n \hookrightarrow\!\!\!\rightarrow n$ exists such that

$$\forall\, i < n\colon \quad \iota_i(\hat{l}_i) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})\,. \tag{17}$$

The $\sim$-invariance of φ means

$$\forall\, i < n\colon \quad i \sim \varphi(i)\,. \tag{18}$$

Let $a \in \mathsf{Loc}$ and $\rightrightarrows \in E$. We claim that the map $\bar{\varphi}\colon \{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\} \to \{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a\}$, $t \mapsto \varphi(t)$ is well defined and a bijection. We prove this claim now:

"$\bar{\varphi}$ is well defined": Let $t {\in} n$ be such that $\rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a$. From (17) we obtain $\iota_{\varphi(i)}(\check{l}_{\varphi(t)}) = a$. From (18) we obtain $\rightsquigarrow_t = \rightsquigarrow_{\varphi(t)}$, implying $\rightsquigarrow_{\varphi(t)} = \rightrightarrows$. Therefore, $\varphi(t) \in \{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a\}$.

"$\bar{\varphi}$ is injective": Follows from the injectivity of φ.

"$\bar{\varphi}$ is surjective": Let $t{<}n$ be given such that $\rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a$. Then $\varphi^{-1}(t) \in n$, and $\rightsquigarrow_{\varphi^{-1}(t)} =$ [due to (18)] $\rightsquigarrow_{\varphi(\varphi^{-1}(t))} = \rightsquigarrow_t = \rightrightarrows$, and $\iota_{\varphi^{-1}(t)}(\hat{l}_{\varphi^{-1}(t)}) =$ [due to (17)] $\iota_{\varphi(\varphi^{-1}(t))}(\check{l}_{\varphi(\varphi^{-1}(t))}) = \iota_t(\check{l}_t) = a$. So $\varphi^{-1}(t) \in \mathsf{dom}\,\bar{\varphi}$. Also, $\bar{\varphi}(\varphi^{-1}(t)) = t$.

We have shown that $\bar{\varphi}$ is a well-defined bijection. Therefore, $|\mathsf{dom}\,\bar{\varphi}| = |\mathsf{img}\,\bar{\varphi}|$.

"$\Leftarrow$": We assume the right-hand side, i.e., $\forall\, a \in \mathsf{Loc}, \rightrightarrows \in E\colon \big|\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\}\big| = \big|\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a\}\big|$. Then for each pair $(\rightrightarrows, a) \in E{\times}\mathsf{Loc}$ there is a bijection $\gamma_{\rightsquigarrow,a}\colon \{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\} \hookrightarrow\!\!\!\rightarrow \{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\check{l}_t) = a\}$. These bijections have pairwise disjoint domains. The images of these bijections are also pairwise disjoint. The union of the domains is n, and the union of the images is also n. Consider the map

$$\varphi \quad \overset{\text{def}}{=} \bigcup_{\rightsquigarrow \in E,\, a \in \mathsf{Loc}} \gamma_{\rightsquigarrow,a}\,.$$

Then φ is a permutation of n. If $i{<}n$ is arbitrary, then $\varphi(i) = \gamma_{\rightsquigarrow_i,\iota_i(\hat{l}_i)}(i)$, and so $\rightsquigarrow_i = \rightsquigarrow_{\gamma_{\rightsquigarrow_i,\iota_i(\hat{l}_i)}(i)} = \rightsquigarrow_{\varphi(i)}$. So φ is $\sim$-invariant. Moreover, $\forall\, i{<}n\colon \iota_i(\hat{l}_i) = \iota_{\gamma_{\rightsquigarrow_i,\iota_i(\hat{l}_i)}(i)}(\check{l}_{\gamma_{\rightsquigarrow_i,\iota_i(\hat{l}_i)}(i)}) = \iota_{\varphi(i)}(\check{l}_{\varphi(i)})$. Thus, $\hat{l} {\asymp} \check{l}$. $\blacksquare$

Lemma V.2.1.11 implies directly:

Corollary V.2.1.12. *For all* $(\hat{g},\hat{l}),(\check{g},\check{l}) \in \mathsf{State}$ *we have* $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$ *iff* $\hat{g}=\check{g} \wedge \big(\forall a\in\mathsf{Loc}, \rightrightarrows\in E\colon \big|\{t<n \mid \rightsquigarrow_t=\rightrightarrows \wedge \iota_t(\hat{l}_t)=a\}\big| = \big|\{t<n \mid \rightsquigarrow_t=\rightrightarrows \wedge \iota_t(\check{l}_t)=a\}\big|\big)$. □

So program states are confusable iff they are the same up to renumbering the threads with confusable identifiers and renaming local states according to the permutations ι_i $(i<n)$.

In the proofs of the following claims, both views of state confusion come in handy.

The next lemma, which says that confusable program states have the same distance from the fixed one, is crucial for the whole section.

Lemma V.2.1.13. $\forall s,s' \in \mathsf{State}\colon\ s\approx s' \ \Rightarrow\ \mathsf{d}((g,l),s)=\mathsf{d}((g,l),s')$.

Proof. We will show the claim by induction on $\min\{\mathsf{d}((g,l),s),\mathsf{d}((g,l),s')\}$, proving $\forall m \in \mathbb{N}_{\geqslant 0} \dot\cup \{\infty\}\colon\ \forall s,s' \in \mathsf{State}\colon\ (s{\approx}s' \wedge \min\{\mathsf{d}((g,l),s),\mathsf{d}((g,l),s')\}=m) \ \Rightarrow\ \mathsf{d}((g,l),s)=\mathsf{d}((g,l),s')$.

So let an arbitrary $m \in \mathbb{N}_{\geqslant 0} \dot\cup \{\infty\}$ be given, and assume that $\forall m'{<}m\colon\ \forall s,s' \in \mathsf{State}\colon\ (s{\approx}s' \wedge \min\{\mathsf{d}((g,l),s),\mathsf{d}((g,l),s')\}=m') \Rightarrow \mathsf{d}((g,l),s)=\mathsf{d}((g,l),s')$. Let $s,s' \in \mathsf{State}$ be given such that $s{\approx}s'$ and $\min\{\mathsf{d}((g,l),s),\mathsf{d}((g,l),s')\}=m$. Three cases can occur: m is zero, m is a positive natural number, or m is infinity.

Case m=0.

Case $\mathsf{d}((g,l),s)=0$. Then $s=(g,l)$. Let $(\check{g},\check{l})=s'$. From $s{\approx}s'$ we obtain $g{=}\check{g}$ and some $\sim$-invariant permutation $\varphi\colon n \hookrightarrow\!\!\!\!\rightarrow n$ such that $\forall i{<}n\colon \iota_i(l_i)=\iota_{\varphi(i)}(\check{l}_{\varphi(i)})$. The definition of ι_i $(i<n)$ implies $\forall i{<}n\colon \iota_i(l_i)=l_0=\iota_{\varphi(i)}(l_{\varphi(i)})$. Therefore, $\forall i{<}n\colon \iota_{\varphi(i)}(\check{l}_{\varphi(i)})=\iota_{\varphi(i)}(l_{\varphi(i)})$. Since $\iota_{\varphi(i)}$ is injective for all $i{<}n$, $\forall i{<}n\colon \check{l}_{\varphi(i)}=l_{\varphi(i)}$. Since φ is onto, $\forall j{<}n\colon l_j{=}\check{l}_j$. So $l{=}\check{l}$. Thus, $s{=}s'$. Hence, $\mathsf{d}((g,l),s)=\mathsf{d}((g,l),s')$.

Case $\mathsf{d}((g,l),s')=0$. We have $s'=(g,l)$. Now, let $(\check{g},\check{l})=s$. From $s{\approx}s'$ we obtain $\check{g}{=}g$ and some $\sim$-invariant permutation $\varphi\colon n \hookrightarrow\!\!\!\!\rightarrow n$ such that $\forall i{<}n\colon \iota_i(\check{l}_i)=\iota_{\varphi(i)}(l_{\varphi(i)})$. The definition of $\sim$ implies $\forall i{<}n\colon \iota_i(l_i)=l_0=\iota_{\varphi(i)}(l_{\varphi(i)})$. Therefore, $\forall i{<}n\colon \iota_i(\check{l}_i)=\iota_i(l_i)$. Since ι_i is injective for all $i{<}n$, $\forall i{<}n\colon \check{l}_i=l_i$. So $\check{l}{=}l$. Thus, $s{=}s'$. Hence, $\mathsf{d}((g,l),s)=\mathsf{d}((g,l),s')$.

Case $0{<}m{<}\infty$. Let $(\check{g},\check{l})=s$ and $(\hat{g},\hat{l})=s'$. From $s{\approx}s'$ we obtain $\check{g}{=}\hat{g}$ and some $\sim$-invariant permutation $\varphi\colon n \hookrightarrow\!\!\!\!\rightarrow n$ such that $\forall i{<}n\colon \iota_i(\check{l}_i)=\iota_{\varphi(i)}(\hat{l}_{\varphi(i)})$.

Case $\mathsf{d}\big((g,l),s\big)=m$. Choose a predecessor $(\check{g}',\check{l}')$ of $(\check{g},\check{l})$ such that

$$\mathsf{d}((g,l),(\check{g}',\check{l}')) \ =\ m-1\,. \tag{19}$$

Let $\hat{l}'=\lambda\, i\in n.\, \iota_i^{-1}(\iota_{\varphi^{-1}(i)}(\check{l}'_{\varphi^{-1}(i)}))$. Then $\forall j<n\colon \iota_{\varphi(j)}(\hat{l}'_{\varphi(j)})=\iota_{\varphi(j)}(\iota^{-1}_{\varphi(j)}(\iota_{\varphi^{-1}(\varphi(j))}(\check{l}'_{\varphi^{-1}(\varphi(j))})))=\iota_j(\check{l}'_j)$. So $(\check{g}',\check{l}')\approx(\check{g}',\hat{l}')$. The induction hypothesis implies $\mathsf{d}((g,l),(\check{g}',\hat{l}'))=m-1$. Choose some $t\in n$ such that $(\check{g}',\check{l}'_t)\rightarrow_t(\check{g},\check{l}_t)\wedge\forall\bar{t}\in n\backslash\{t\}\colon \check{l}'_{\bar{t}}=\check{l}_{\bar{t}}$. Due to (19) and the distance condition in this branch of the case split, $(\check{g}',\check{l}'_t)\neq(\check{g},\check{l}_t)$. Therefore, $(\check{g}',\iota_t(\check{l}'_t))\rightsquigarrow_t(\check{g},\iota_t(\check{l}_t))$. Since $t\sim\varphi(t)$, we have $\rightsquigarrow_t\ =\ \rightsquigarrow_{\varphi(t)}$. So $(\check{g}',\iota_t(\check{l}'_t))\rightsquigarrow_{\varphi(t)}(\check{g},\iota_t(\check{l}_t))$. Thus, $(\check{g}',\iota^{-1}_{\varphi(t)}(\iota_t(\check{l}'_t)))\rightarrow_{\varphi(t)}(\check{g},\iota^{-1}_{\varphi(t)}(\iota_t(\check{l}_t)))$. Knowing in addition that $\hat{l}'_{\varphi(t)}=\iota^{-1}_{\varphi(t)}(\iota_{\varphi^{-1}(\varphi(t))}(\check{l}'_{\varphi^{-1}(\varphi(t))}))=\iota^{-1}_{\varphi(t)}(\iota_t(\check{l}'_t))$ and $\iota^{-1}_{\varphi(t)}(\iota_t(\check{l}_t))=\iota^{-1}_{\varphi(t)}(\iota_{\varphi(t)}(\hat{l}_{\varphi(t)}))=\hat{l}_{\varphi(t)}$, we get $(\check{g}',\hat{l}'_{\varphi(t)})\rightarrow_{\varphi(t)}(\check{g},\hat{l}_{\varphi(t)})$. Moreover, $\forall\bar{t}\in n\backslash\{\varphi(t)\}\colon\ \hat{l}'_{\bar{t}}=\iota_{\bar{t}}^{-1}(\iota_{\varphi^{-1}(\bar{t})}(\check{l}'_{\varphi^{-1}(\bar{t})}))=$ [from $\bar{t}\neq\varphi(t)$ we get

$\varphi^{-1}(\bar{t}) \neq t]$ $\iota_{\bar{t}}^{-1}(\iota_{\varphi^{-1}(\bar{t})}(\check{l}_{\varphi^{-1}(\bar{t})})) = \iota_{\bar{t}}^{-1}(\iota_{\varphi(\varphi^{-1}(\bar{t}))}(\hat{l}_{\varphi(\varphi^{-1}(\bar{t}))})) = \iota_{\bar{t}}^{-1}(\iota_{\bar{t}}(\hat{l}_{\bar{t}})) = \hat{l}_{\bar{t}}$. Combining, $(\check{g}', \hat{l}') \longrightarrow (\check{g}, \hat{l}) = (\hat{g}, \hat{l})$. Thus, $\mathsf{d}((g,l),(\hat{g},\hat{l})) \leqslant m = \min\{\mathsf{d}((g,l),(\check{g},\check{l})), \mathsf{d}((g,l),(\hat{g},\hat{l}))\} \leqslant \mathsf{d}((g,l),(\hat{g},\hat{l}))$. Therefore, $\mathsf{d}((g,l),(\hat{g},\hat{l})) = m = \mathsf{d}((g,l),(\check{g},\check{l}))$.

Case $\mathsf{d}((g,l),s') = m$. Analogously as follows. Choose a predecessor $(\hat{g}', \hat{l}')$ of $(\hat{g}, \hat{l})$ such that

$$\mathsf{d}((g,l),(\hat{g}',\hat{l}')) \;=\; m-1\,. \tag{20}$$

Let $\check{l}' = \lambda\, i{\in}n.\, \iota_i^{-1}(\iota_{\varphi(i)}(\hat{l}'_{\varphi(i)}))$. Due to Lemma V.2.1.5, φ^{-1} is $\sim$-invariant. Moreover, $\forall\, j{<}n\colon \iota_{\varphi^{-1}(j)}(\check{l}'_{\varphi^{-1}(j)}) = \iota_{\varphi^{-1}(j)}(\iota^{-1}_{\varphi^{-1}(j)}(\iota_{\varphi(\varphi^{-1}(j))}(\hat{l}'_{\varphi(\varphi^{-1}(j))}))) = \iota_j(\hat{l}'_j)$. So $(\hat{g}', \hat{l}') \approx (\hat{g}', \check{l}')$. The induction hypothesis implies $\mathsf{d}((g,l), (\hat{g}', \check{l}')) = m{-}1$. Choose some $t{\in}n$ such that $(\hat{g}', \hat{l}'_t) \rightarrow_t (\hat{g}, \hat{l}_t)$ and $\forall\, \bar{t} \in n\backslash\{t\}\colon \hat{l}'_{\bar{t}} = \hat{l}_{\bar{t}}$. Due to (20) and the distance condition in this branch of the case split, $(\hat{g}', \hat{l}'_t) \neq (\hat{g}, \hat{l}_t)$. Therefore, $(\hat{g}', \iota_t(\hat{l}'_t)) \rightsquigarrow_t (\hat{g}, \iota_t(\hat{l}_t))$. Note that $t \sim \varphi^{-1}(t)$; thus, $\rightsquigarrow_t \,=\, \rightsquigarrow_{\varphi^{-1}(t)}$. So $(\hat{g}', \iota_t(\hat{l}'_t)) \rightsquigarrow_{\varphi^{-1}(t)} (\hat{g}, \iota_t(\hat{l}_t))$. Thus, $(\hat{g}', \iota^{-1}_{\varphi^{-1}(t)}(\iota_t(\hat{l}'_t))) \rightarrow_{\varphi^{-1}(t)} (\hat{g}, \iota^{-1}_{\varphi^{-1}(t)}(\iota_t(\hat{l}_t)))$. Knowing in addition that $\check{l}'_{\varphi^{-1}(t)} = \iota^{-1}_{\varphi^{-1}(t)}(\iota_{\varphi(\varphi^{-1}(t))}(\hat{l}'_{\varphi(\varphi^{-1}(t))})) = \iota^{-1}_{\varphi^{-1}(t)}(\iota_t(\hat{l}'_t))$ and $\iota^{-1}_{\varphi^{-1}(t)}(\iota_t(\hat{l}_t)) = \iota^{-1}_{\varphi^{-1}(t)}(\iota_{\varphi(\varphi^{-1}(t))}(\hat{l}_{\varphi(\varphi^{-1}(t))})) = \iota^{-1}_{\varphi^{-1}(t)}(\iota_{\varphi^{-1}(t)}(\check{l}_{\varphi^{-1}(t)})) = \check{l}_{\varphi^{-1}(t)}$, we get $(\hat{g}', \check{l}'_{\varphi^{-1}(t)}) \rightarrow_{\varphi^{-1}(t)} (\hat{g}, \check{l}_{\varphi^{-1}(t)})$. Moreover, $\forall\, \bar{t} \in n\backslash\{\varphi^{-1}(t)\}\colon\ \check{l}'_{\bar{t}} = \iota_{\bar{t}}^{-1}(\iota_{\varphi(\bar{t})}(\hat{l}'_{\varphi(\bar{t})})) =$ [since $\bar{t} \neq \varphi^{-1}(t)$, we have $\varphi(\bar{t}) \neq t$] $\iota_{\bar{t}}^{-1}(\iota_{\varphi(\bar{t})}(\hat{l}_{\varphi(\bar{t})})) = \iota_{\bar{t}}^{-1}(\iota_{\bar{t}}(\check{l}_{\bar{t}})) = \check{l}_{\bar{t}}$. Combining, $(\hat{g}', \check{l}') \longrightarrow (\hat{g}, \check{l}) = (\check{g}, \check{l})$. Thus, $\mathsf{d}((g,l),(\check{g},\check{l})) \leqslant m = \min\{\mathsf{d}((g,l),(\check{g},\check{l})), \mathsf{d}((g,l),(\hat{g},\hat{l}))\} \leqslant \mathsf{d}((g,l),(\check{g},\check{l}))$. Therefore, $\mathsf{d}((g,l),(\check{g},\check{l})) = m = \mathsf{d}((g,l),(\hat{g},\hat{l}))$.

Case $m{=}\infty$. Then both $\mathsf{d}((g,l),s)$ and $\mathsf{d}((g,l),s')$ must be ∞. In particular, they are equal. ∎

From this result we directly conclude:

Lemma V.2.1.14. $\forall\, s{\in}\mathsf{State}\colon \mathsf{d}((g,l),s) < \infty \Rightarrow \mathsf{d}((g,l),s) < \left|{}^{\mathsf{State}}\!/\!_{\approx}\right|$.

Proof. Let $s{\in}\mathsf{State}$ have a finite distance from (g,l). Take a shortest walk $(\sigma^{[0]}, \ldots, \sigma^{[k]})$ in the program's transition graph such that $\sigma^{[0]} = (g,l)$ and $\sigma^{[k]} = s$. Then $k = \mathsf{d}((g,l),s)$. Note that $\forall\, i{\leqslant}k\colon \mathsf{d}((g,l),\sigma^{[i]}) = i$. Lemma V.2.1.13 implies that $\sigma^{[i]} \not\approx \sigma^{[j]}$ for all $i,j \in \mathbb{N}_{\geqslant 0}$ such that $i,j \leqslant k$ and $i{\neq}j$. Thus, the relation $\approx$ has at least $k{+}1$ equivalence classes. That is, $\left|{}^{\mathsf{State}}\!/\!_{\approx}\right| \geqslant \mathsf{d}((g,l),s) + 1 > \mathsf{d}((g,l),s)$. ∎

Computing the index of $\approx$ can be reduced to computing the index of $\asymp$:

Lemma V.2.1.15. $\left|{}^{\mathsf{State}}\!/\!_{\approx}\right| = G\left|{}^{\mathsf{Loc}^n}\!/\!_{\asymp}\right|$.

Proof. Follows from Definition V.2.1.2 and Lemma V.2.1.8b). ∎

To get a polynomial upper bound on the index of $\asymp$, we establish an injection from the set of equivalence classes to another set for which the cardinality will be easy to determine.

Let V be the set of the maps $f\colon E \rightarrow \mathsf{Loc} \rightarrow \mathbb{N}_{\geqslant 0}$ such that, for each simple thread transition relation $\rightrightarrows$, the sequence of values of $f(\rightrightarrows)$ forms a partition of the number of all the threads that have $\rightrightarrows$ as the normalized thread transition relation. Formally:

$$V \;\stackrel{\text{def}}{=}\; \left\{ f \in (E \rightarrow \mathsf{Loc} \rightarrow \mathbb{N}_{\geqslant 0}) \;\middle|\; \forall \rightrightarrows\, \in E\colon \|f(\rightrightarrows)\|_1 = \left|\{t{<}n \mid \rightsquigarrow_t \,=\, \rightrightarrows\}\right| \right\}. \tag{21}$$

Lemma V.2.1.16. *There is an injection from* $\mathsf{Loc}^n\!/\!\asymp$ *to* V.

Proof. Let $\varphi\colon \mathsf{Loc}^n \to E \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}$, $\hat{l} \mapsto \lambda \rightrightarrows \in E.\, \lambda\, a{\in}\mathsf{Loc}.\, \big|\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\}\big|$.

First, we show that $\operatorname{img}\varphi \overset{!}{\subseteq} V$. For that, let $\hat{l}{\in}\mathsf{Loc}^n$ be arbitrary; it suffices to show that $\varphi(\hat{l}) \overset{!}{\in} V$. Note that $\varphi(\hat{l}) \in (E \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0})$. Then $\|\varphi(\hat{l})(\rightrightarrows)\|_1 = \sum_{a\in\mathsf{Loc}} |\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows \wedge \iota_t(\hat{l}_t) = a\}| = |\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows\}|$ for all $\rightrightarrows \in E$. So $\varphi(\hat{l}) \in V$. Since $\hat{l}$ was arbitrary,

$$\operatorname{img}\varphi \subseteq V. \tag{22}$$

Lemma V.2.1.11 implies

$$\forall\, \hat{l}, \check{l} \in \mathsf{Loc}^n\colon\; \hat{l} \asymp \check{l} \;\Leftrightarrow\; \varphi(\hat{l}) = \varphi(\check{l})\,.$$

Due to this fact and (22), the map

$$\psi\colon\; \mathsf{Loc}^n\!/\!\asymp\; \to V,\; [\bar{l}]_\asymp \mapsto \varphi(\bar{l})$$

is well defined and an injection. ■

To estimate the cardinality of V, we employ a generalization of the binomial coefficients in which the upper argument can take arbitrary rational values:

$$\binom{x}{m} \overset{\text{def}}{=\!=} \prod_{i<m} \frac{x-i}{i+1} \quad \text{for } x \in \mathbb{Q} \text{ and } m \in \mathbb{N}_{\geqslant 0},$$

with the convention that the empty product evaluates to 1.

We need a standard combinatorial lemma:

Proposition V.2.1.17. *There are* $\binom{k+m-1}{k}$ *ways to arrange* k *indistinguishable balls into* m *distinguishable baskets. Formally:*

$$\Big|\big\{g\colon m \to \mathbb{N}_{\geqslant 0} \;\big|\; \sum_{c<m} g(c) = k\big\}\Big| \;=\; \binom{k+m-1}{k}.$$

(Remark: in the above formulation, $g(c)$ is the number of balls in basket c, where the baskets are enumerated by integers from 0 up to but not including m.)

Proof. Any arrangement of k indistinguishable balls into m distinguishable baskets can be written as a string over ⊙ and |, e.g., ⊙⊙|⊙||⊙⊙⊙|||⊙|, where ⊙ denotes a ball, | separates the baskets, and there are k symbols ⊙ and $m{-}1$ delimiters |. The example string above, processed from left to right, shows two balls in the first basket, one ball in the second, none in the third, three in the fourth, none in the sixth, none in the seventh, one in the eighth, and none in the ninth. This mapping of arrangements to strings is injective. Moreover, this mapping is also surjective: each string over k symbols ⊙ and $m{-}1$ symbols | can be interpreted as an arrangement of k balls into m baskets. Therefore, it suffices to count the number of such strings, which is $\binom{k+m-1}{k}$. ■

The definition of V in (21) depends on the transitions of the threads. Now we bound $|V|$ by an expression over G, L, and n that does not involve this dependency:

Lemma V.2.1.18. $|V| \;\leqslant\; \max\big\{\prod_{i<2^{GL(GL-1)}} \binom{k_i+L-1}{L-1} \;\big|\; k_0, \ldots, k_{2^{GL(GL-1)}-1} \in \mathbb{N}_{\geqslant 0} \wedge \sum_{i<2^{GL(GL-1)}} k_i = n\big\}$.

Proof. Let $k_{\rightrightarrows} \stackrel{\text{def}}{=} |\{t{\in}n \mid \rightsquigarrow_t = \rightrightarrows\}|$ for each $\rightrightarrows \in E$. Proposition V.2.1.17 implies that $|\{g \in (\mathsf{Loc} \to \mathbb{N}_{\geqslant 0}) \mid \sum_{c \in \mathsf{Loc}} g(c) = k_{\rightrightarrows}\}| = \binom{k_{\rightrightarrows}+L-1}{k_{\rightrightarrows}} = \binom{k_{\rightrightarrows}+L-1}{L-1}$ for each $\rightrightarrows \in E$. Thus,

$$\prod_{\rightrightarrows \in E} \left| \left\{ g \in (\mathsf{Loc} {\to} \mathbb{N}_{\geqslant 0}) \mid \sum_{c \in \mathsf{Loc}} g(c) = k_{\rightrightarrows} \right\} \right| = \prod_{\rightrightarrows \in E} \binom{k_{\rightrightarrows} + L - 1}{L - 1} . \tag{23}$$

The left-hand side of (23) is equal to $\Big| \prod\limits_{\rightrightarrows \in E} \{g \in (\mathsf{Loc} {\to} \mathbb{N}_{\geqslant 0}) \mid \sum\limits_{c \in \mathsf{Loc}} g(c) = k_{\rightrightarrows}\} \Big| = \Big| \{ f \in (E \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}) \mid \forall {\rightrightarrows} {\in} E \colon \sum\limits_{c \in \mathsf{Loc}} f(\rightrightarrows)(c) = k_{\rightrightarrows}\} \Big| = |V|$. Since $\sum\limits_{\rightrightarrows \in E} k_{\rightrightarrows} = n$, the right-hand side of (23) is bounded above by $\max \Big\{ \prod\limits_{\rightrightarrows \in E} \binom{\hat{k}_{\rightrightarrows}+L-1}{L-1} \;\Big|\; (\hat{k}_{\rightrightarrows})_{\rightrightarrows \in E} \in (\mathbb{N}_{\geqslant 0})^E \wedge \sum\limits_{\rightrightarrows \in E} \hat{k}_{\rightrightarrows} = n \Big\} =$ [knowing that $|E| = 2^{G^2L^2 - GL} = 2^{GL(GL-1)}$] $\max \Big\{ \prod_{i < 2^{GL(GL-1)}} \binom{\hat{k}_i+L-1}{L-1} \;\Big|\; (\hat{k}_i)_{i < 2^{GL(GL-1)}} \in (\mathbb{N}_{\geqslant 0})^{2^{GL(GL-1)}} \wedge \sum_{i < 2^{GL(GL-1)}} \hat{k}_i = n \Big\}$. Therefore, $|V| \leqslant \max \Big\{ \prod_{i < 2^{GL(GL-1)}} \binom{\hat{k}_i+L-1}{L-1} \;\Big|\; (\hat{k}_i)_{i < 2^{GL(GL-1)}} \in (\mathbb{N}_{\geqslant 0})^{2^{GL(GL-1)}} \wedge \sum_{i < 2^{GL(GL-1)}} \hat{k}_i = n \Big\}$. ■

Now we prepare to simplify the maximum from Lemma V.2.1.18. Recall that a product of constantly many nonnegative integers with a constant sum is maximal when these integers coincide. We require a similar result:

Lemma V.2.1.19. *Let* $n,m \in \mathbb{N}_{\geqslant 0}$ *and* $t {\in} \mathbb{N}_{+}$. *Then* $\max \Big\{ \prod_{i<t} \binom{k_i+m}{m} \;\Big|\; (k_i)_{i<t} \in (\mathbb{N}_{\geqslant 0})^t \wedge \sum_{i<t} k_i = n \Big\} \leqslant \binom{n/t+m}{m}^t$.

Proof. We are going to prove a generalization of the claim (namely, where k_i are relaxed to be nonnegative rationals) by induction on $|\{i{<}t \mid k_i \neq \frac{n}{t}\}|$. More precisely, we are going to show $\forall r {\in} \mathbb{N}_{\geqslant 0} \colon \forall (k_i)_{i<t} \in \mathbb{Q}^t_{\geqslant 0} \colon \big(|\{i{<}t \mid k_i \neq \frac{n}{t}\}| = r \wedge \sum_{i<t} k_i = n \big) \Rightarrow \prod_{i<t} \binom{k_i+m}{m} \leqslant \binom{n/t+m}{m}^t$.

So let $r \in \mathbb{N}_{\geqslant 0}$ and $(k_i)_{i<t} \in \mathbb{Q}^t_{\geqslant 0}$ be given such that $|\{i{<}t \mid k_i \neq \frac{n}{t}\}| = r$ and $\sum_{i<t} k_i = n$, and the induction hypothesis is satisfied, i.e., $\forall \tilde{r}{<}r \colon \forall (\tilde{k}_i)_{i<t} \in \mathbb{Q}^t_{\geqslant 0} \colon \big(|\{i{<}t \mid \tilde{k}_i \neq \frac{n}{t}\}| = \tilde{r} \wedge \sum_{i<t} \tilde{k}_i = n \big) \Rightarrow \prod_{i<t} \binom{\tilde{k}_i+m}{m} \leqslant \binom{n/t+m}{m}^t$.

If $r{=}0$, then $\forall i{<}t \colon k_i = \frac{n}{t}$, and so $\prod_{i<t} \binom{k_i+m}{m} = \binom{n/t+m}{m}^t$.

So we assume from now on that $r{>}0$. Since $\exists i{<}t \colon k_i \neq \frac{n}{t}$ and $\sum_{i<t} k_i = n$, there must be $i, j < t$ such that $k_i < \frac{n}{t} < k_j$. Let

$$\delta \stackrel{\text{def}}{=} \min \left\{ \frac{n}{t} - k_i,\; k_j - \frac{n}{t} \right\} \quad \text{and} \quad \tilde{k}_h \stackrel{\text{def}}{=} \begin{cases} k_h , & \text{if } i \neq h \neq j , \\ k_h + \delta , & \text{if } h = i , \\ k_h - \delta , & \text{if } h = j \end{cases} \qquad (h < t).$$

Then $\sum_{h<t} \tilde{k}_h = n$ and $\big| \{h{<}t \mid \tilde{k}_h \neq \frac{n}{t}\} \big| < r$. The induction hypothesis implies

$$\prod_{h<t} \binom{\tilde{k}_h + m}{m} \leqslant \binom{n/t + m}{m}^t . \tag{24}$$

Note that $\frac{\binom{\tilde{k}_i+m}{m}\binom{\tilde{k}_j+m}{m}}{\binom{k_i+m}{m}\binom{k_j+m}{m}}$ is well defined and equal to
$\frac{\Big(\prod\limits_{h<m} \frac{\tilde{k}_i+m-h}{h+1}\Big)\Big(\prod\limits_{h<m} \frac{\tilde{k}_j+m-h}{h+1}\Big)}{\Big(\prod\limits_{h<m} \frac{k_i+m-h}{h+1}\Big)\Big(\prod\limits_{h<m} \frac{k_j+m-h}{h+1}\Big)} = \prod\limits_{h<m} \frac{(\tilde{k}_i+m-h)(\tilde{k}_j+m-h)}{(k_i+m-h)(k_j+m-h)} = \prod\limits_{r=1}^{m} \frac{(\tilde{k}_i+r)(\tilde{k}_j+r)}{(k_i+r)(k_j+r)} =$

$\prod_{r=1}^{m} \frac{(k_i+r+\delta)(k_j+r-\delta)}{(k_i+r)(k_j+r)} = \prod_{r=1}^{m} \frac{(k_i+r)(k_j+r)+\delta(k_j+r)-\delta(k_i+r)-\delta^2}{(k_i+r)(k_j+r)} = \prod_{r=1}^{m}\left(1 + \frac{\delta(k_j-k_i)-\delta^2}{(k_i+r)(k_j+r)}\right) =$
$\prod_{r=1}^{m}\left(1 + \delta\frac{k_j-k_i-\delta}{(k_i+r)(k_j+r)}\right) \geqslant$
[since $k_j - k_i - \delta = k_j - \frac{n}{t} + \frac{n}{t} - k_i - \delta \geqslant \delta + \delta - \delta = \delta > 0$]
$\prod_{r=1}^{m}\left(1 + \frac{\delta^2}{(k_i+r)(k_j+r)}\right) \geqslant 1.$
Therefore, $\frac{\binom{k_i+m}{m}\binom{k_j+m}{m}}{\binom{\tilde{k}_i+m}{m}\binom{\tilde{k}_j+m}{m}} \leqslant 1$, and so $\prod_{h<t}\binom{k_h+m}{m} = \frac{\binom{k_i+m}{m}\binom{k_j+m}{m}}{\binom{\tilde{k}_i+m}{m}\binom{\tilde{k}_j+m}{m}}\prod_{h<t}\binom{\tilde{k}_h+m}{m} \leqslant \prod_{h<t}\binom{\tilde{k}_h+m}{m} \leqslant$ [using (24)] $\binom{n/t+m}{m}^t$. ∎

Now we can bound the index of $\approx$ by a less complicated term:

Corollary V.2.1.20. $\left|{}^{\mathsf{State}}\!/\!_{\approx}\right| \leqslant G\binom{n/t+L-1}{L-1}^t$ *where* $t = 2^{GL(GL-1)}$.

Proof. $\left|{}^{\mathsf{State}}\!/\!_{\approx}\right|$ = [according to Lemma V.2.1.8b)] $G\left|{}^{\mathsf{Loc}^n}\!/\!_{\asymp}\right| \leqslant$ [according to Lemma V.2.1.16] $G|V| \leqslant$ [using Lemma V.2.1.18] $G\max\left\{\prod_{i<2^{GL(GL-1)}}\binom{k_i+L-1}{L-1} \;\middle|\; k_0, \ldots, k_{2^{GL(GL-1)}-1} \in \mathbb{N}_{\geqslant 0} \wedge \sum_{i<2^{GL(GL-1)}} k_i = n\right\} \leqslant$ [applying Lemma V.2.1.19 to $m = L-1$ and $t = 2^{GL(GL-1)}$] $G\binom{\frac{n}{2^{GL(GL-1)}} + L - 1}{L-1}^{2^{GL(GL-1)}}$. ∎

Now the computation for a fixed, arbitrary program is over and we turn to the whole class of programs.

We recall a standard, simple combinatorial lemma:

Proposition V.2.1.21. $\forall x \in \mathbb{Q}, m \in \mathbb{N}_{\geqslant 0}\colon \binom{x+m}{m} = \prod_{j=1}^{m}\frac{x+j}{j}$.

Proof. Let $x \in \mathbb{Q}$ and $m \in \mathbb{N}_{\geqslant 0}$. Then $\binom{x+m}{m} = \prod_{i<m}\frac{x+m-i}{i+1} = \frac{\prod_{i<m}(x+m-i)}{\prod_{i<m}(i+1)}$ = [changing the index $i = m-r$ in the enumerator and $i = h-1$ in the denominator] $\frac{\prod_{r=1}^{m}(x+r)}{\prod_{h=1}^{m}h} = \prod_{j=1}^{m}\frac{x+j}{j}$. ∎

This helps to asymptotically bound diamax by a polynomial in n:

Theorem V.2.1.22. $\mathsf{diamax}(n) < G\binom{\frac{n}{2^{GL(GL-1)}} + L - 1}{L-1}^{2^{GL(GL-1)}} = \mathsf{O}\left(n^{(L-1)2^{GL(GL-1)}}\right)$.

Proof. In this proof, we use the fact that both the n-threaded program and the initial state assumed for the prior claims in § V.2.1 were arbitrary.

We have $\mathsf{diamax}(n) = \max\{\mathsf{diam}(p) \mid p$ is an n-threaded program$\} = \max\{(\mathsf{img}\,\mathsf{d}_p)\setminus\{\infty\} \mid p$ is an n-threaded program$\}$ < [using Lemma V.2.1.14] $\max\left\{\left|{}^{\mathsf{State}}\!/\!_{\approx}\right| \;\middle|\; \exists\right.$ an n-threaded program p: State is the set of states of p and $\approx$ is defined as in Definition V.2.1.2 for $\left.p\right\} \leqslant$ [using Corollary V.2.1.20] $G\binom{\frac{n}{2^{GL(GL-1)}} + L - 1}{L-1}^{2^{GL(GL-1)}}$

= [using Proposition V.2.1.21] $G\left(\prod_{r=1}^{L-1}\frac{\frac{n}{2^{GL(GL-1)}} + r}{r}\right)^{2^{GL(GL-1)}} \leqslant$ [for $L = 1$, the last big product is empty and evaluates to 1, and for $L \geqslant 2$, $1 \leqslant r \leqslant L-1$, and $n \geqslant 1$, we have $\frac{\frac{n}{2^{GL(GL-1)}}+r}{r} = \frac{n}{2^{GL(GL-1)}\cdot r} + 1 \leqslant \frac{n}{2^{1\cdot 2(1\cdot 2-1)}\cdot 1} + 1 = \frac{n}{2^2} + 1 = \frac{n}{4} + 1 \leqslant 2n$] $G(2n)^{(L-1)\cdot 2^{GL(GL-1)}}$. ∎

To the best of our knowledge, $\mathsf{O}(n^{(L-1)2^{GL(GL-1)}})$ is the lowest known polynomial upper bound for diamax. Comparing the degree $(L-1)2^{GL(GL-1)}$ to bounds from [78] from a theoretical viewpoint, we feel it comforting to see that our degree is relatively small: it involves *only one* exponentiation operation. Tightening Theorem V.2.1.22 is an open problem, but we expect that the degree could be lowered further by reducing E from Definition V.2.1.10 and considering distance-preserving bisimulations. Sections V.2.2 and V.2.3 will be encouraging: in the setups described there, the degree is provably 1.

Note V.2.1.23. The current degree of the upper bound on diamax has been improved over the previously known degree from [60] by the factor of L. In the binary case we obtain $\mathsf{diamax}(n) = \mathsf{O}\big(n^{1\cdot 2^{4\cdot 3}}\big) = \mathsf{O}\big(n^{2^{12}}\big)$, which is an improvement over the $\mathsf{O}\big(n^{2^{17}}\big)$ bound from [64] by the factor of 32 in the degree. □

V.2.2. Strongly connected subprograms

Now we show a linear upper bound for families of programs that have a subprogram with a strongly connected transition graph. A graph is *strongly connected* if, for each pair of nodes of the graph, there is a walk from the first node to the second node. Slightly abusing the terminology, we say that a program is *strongly connected* if its transition graph is strongly connected.

Models of real-world programs may possess strongly connected subprograms. This may happen when we produce an abstraction of a system. For example, the coarsest abstract thread [53] uses only one local state and changes the shared memory arbitrarily. Another way in which strongly connected subprograms may occur is via the automatic generation of models inside abstraction-refinement loops: it starts with the coarsest thread. Even more-refined thread abstractions may be strongly connected, e.g., when we model a reactive thread running an infinite loop which responds to the environment by changing the shared state in different ways.

For a program with transition relation $\longrightarrow$, we write $\sigma \longrightarrow^{\leqslant k} \sigma'$ iff the state σ' is reachable from σ in at most k steps. Formally:

$$\longrightarrow^{\leqslant 0} \stackrel{\text{def}}{=} \mathsf{id}_{\mathsf{State}} \quad \text{and}$$
$$\longrightarrow^{\leqslant k} \stackrel{\text{def}}{=} \longrightarrow^{\leqslant k-1} \cup \left(\longrightarrow^{\leqslant k-1} \fatsemi \longrightarrow\right) \quad (k \in \mathbb{N}_+)\,,$$

where State is the set of program states and $\fatsemi$ is the left composition.

While the cases of low values of G (or L) will be easy to handle, for the most general case $G \geqslant 2$ we will need some preparation. Let C be the maximal local diameter (cf. Definition and Corollary IV.3.8) for the rest of this section. Now, we determine how long it takes in a strongly connected program to achieve a given shared part in the worst case:

Lemma V.2.2.1. *Let h be an m-threaded strongly connected program with at least two shared states and (g,l) a state of the program. Let $\delta = \min\Big\{C, \big(\text{if } G=2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\big), \mathsf{diam}(h)\Big\}$. Then each shared state can be reached from (g,l) in at most δ steps. Formally: $\big(G\geqslant 2 \wedge h$ is strongly connected $\wedge$ the transition graph of h is $(\mathsf{State},\longrightarrow)\big) \Rightarrow \big(\forall\,(g,l) \in \mathsf{State},\ g'\in\mathsf{Glob}\ \exists\, l' \in \mathsf{Loc}^m\colon\ (g,l) \longrightarrow^{\leqslant \delta} (g',l')\big)$.*

Proof. Let $h = (\rightarrow_i)_{i<m}$, $(g,l) \in \mathsf{State}$ and $g'\in\mathsf{Glob}$.

If $g=g'$, take $l'=l$ and obtain $(g,l) \longrightarrow^* (g',l')$ in zero steps, and $0 \leqslant (G-1)L^m - L^{m-1} + 1$ because of $G\geqslant 2$.

Otherwise, consider the case $g{\neq}g'$ from now on. In particular, $G{\geqslant}2$ in this case. Since the transition graph is strongly connected, there is a path from (g,l) to some program state $(g',\_)$. Among all such paths, we choose a shortest one; let us refer to it as $\sigma = (\sigma^{[j]})_{j\leqslant k}$ for some $k{\geqslant}0$. Since $g{\neq}g'$, we even have $k{\geqslant}1$. Let $(g^{[j]}, l^{[j]}) = \sigma^{[j]}$ for all $j \leqslant k$. Note that $g^{[k]} = g'$; let $l' = l^{[k]}$.

Now we derive the three bounds on $\mathsf{length}(\sigma)$: C, (if $G{=}2$ then L else $(G{-}1)L^m - L^{m-1} + 1$), and $\mathsf{diam}(h)$:

- By the definition of C, we have $\mathsf{d}^{\mathsf{loc}}((g,l), 0, (g', l'_0)) \leqslant C$. According to the definition of local distances, there is a walk $\breve{\sigma} = (\breve{\sigma}^{[j]})_{j\leqslant\check{k}}$ and $\check{l} \in \mathsf{Loc}^m$ such that $\breve{\sigma}^{[0]} = (g,l) \wedge \breve{\sigma}_{\check{k}} = (g', \check{l}) \wedge \check{l}_0 = l'_0 \wedge \check{k} = \mathsf{d}^{\mathsf{loc}}((g,l), 0, (g', l'_0))$. The choice of σ implies $\mathsf{length}(\sigma) \leqslant \mathsf{length}(\breve{\sigma})$, which, in turn, implies $k \leqslant \check{k} \leqslant C$.
- There is some i such that $(g^{[k-1]}, l_i^{[k-1]}) \to_i (g', l_i^{[k]})$ and $l_{\hat{\imath}}^{[k-1]} = l_{\hat{\imath}}^{[k]}$ for all $\hat{\imath} \in n \setminus \{i\}$. By minimality of σ, $g^{[j]} \neq g'$ for all $j{<}k$; in particular, $g^{[k-1]} \neq g'$.

  Case $G{=}2$. Within the path $(\sigma^{[j]})_{j<k}$, the shared state is constant. So every transition there is induced by thread i (since transitions induced by other threads can be skipped, resulting in a shorter path to a program state with the shared part g', in contradiction to the minimality of σ). Thus, only the local state of thread i changes along $(\sigma^{[j]})_{j<k}$. Since a path has no self-intersections, $k{\leqslant}L$.

  Case $G{>}2$. If we had $(g^{[k-1]}, l_i^{[k-1]}) = (g^{[j]}, l_i^{[j]})$ for some $j < k{-}1$, we could have taken the same thread transition $((g^{[k-1]}, l_i^{[k-1]}), (g', l_i^{[k]}))$ earlier in σ in contradiction to its minimality. Therefore, $(g^{[k-1]}, l_i^{[k-1]})$ does not occur as a thread state of thread i in $(\sigma^{[j]})_{j<k-1}$. Thus, $\sigma^{[j]} \in \Big(\big(\mathsf{Glob} \setminus \{g'\}\big) \times \mathsf{Loc}^m\Big) \setminus \Big(\{g^{[k-1]}\} {\times} \big(\mathsf{Loc}^i {\times} \underbrace{\{l_i^{[k-1]}\}}_{\text{at index } i} {\times} \mathsf{Loc}^{m-i-1}\big)\Big)$ for all $j < k{-}1$. So, since $(\sigma^{[j]})_{j<k-1}$ is not self-intersecting, it has at most $(G{-}1)L^m - L^{m-1}$ states. Thus, $k{-}1 \leqslant (G{-}1)L^m - L^{m-1}$, which implies $k \leqslant (G{-}1)L^m - L^{m-1} + 1$.

  In summary, $k \leqslant$ (if $G{=}2$ then L else $(G{-}1)L^m - L^{m-1} + 1$).
- Note that σ is a shortest path from (g,l) to (g',l'). So $k \leqslant \mathsf{diam}(h)$.

Summarizing, $\mathsf{length}(\sigma) \leqslant \min\Big\{C, \big(\text{if } G = 2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\big), \mathsf{diam}(h)\Big\}$. ∎

As an aside concerning the proof above, the treatment of Case $G{>}2$ does not rely on $G{\neq}2$, but also works for $G{=}2$. Following this case in the proof above with $G{=}2$ in mind, we simply obtain the upper bound $(G{-}1)L^m - L^{m-1} + 1 = L^m - L^{m-1} + 1 = (L-1)L^{m-1} + 1$, which coincides with L in case $m{=}1 \vee L{=}1$ and exceeds L otherwise.

For $G{>}2$, it is possible to slightly reduce the upper bound δ from Lemma V.2.2.1 by considering longer suffixes of σ while treating Case $G{>}2$ (than suffixes of length 1 as of now); the resulting upper bound would become an increasingly complicated expression, and, therefore, we are not going to do this. Whether it is possible to significantly reduce δ is an open problem.

As we will see, a strongly connected subprogram is in some sense a "universal helper," since, if h is a strongly connected subprogram of program p, then h may "help" to lift certain sequences of local states of p (that are almost walks in the transition graph p induced by a single thread, but the shared states do not match properly) to actual walks in the transition graph of p:

Lemma V.2.2.2. *Let $G \geqslant 2$ and h be a strongly connected, m-threaded subprogram of an n-threaded program $p = (\rightarrow_i)_{i<n}$ via an embedding f. Let $\delta = \min\Big\{C, \big(\text{if } G = 2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\big), \mathsf{diam}(h)\Big\}$. Let $\longrightarrow$ be the transition relation of p, $i \in n\backslash(\mathsf{img}\, f)$, (g, l) a state of p, $k \in \mathbb{N}_{\geqslant 0}$, $\sigma = \big(\sigma^{[j]}\big)_{j\leqslant k}$ a sequence of local states that starts with l_i and satisfies $\forall\, j<k\colon \big(\_, \sigma^{[j]}\big) \rightarrow_i \big(\_, \sigma^{[j+1]}\big) \vee \sigma^{[j]} = \sigma^{[j+1]}$. Then there is a program state $(\hat{g}, \hat{l})$ of p such that $\hat{l}|_{n\backslash((\mathsf{img}\, f)\cup\{i\})} = l|_{n\backslash((\mathsf{img}\, f)\cup\{i\})}$, $\sigma^{[k]} = \hat{l}_i$, and $(g, l) \longrightarrow^{\leqslant(\delta+1)(L-1)} (\hat{g}, \hat{l})$.*

Proof. If $\sigma^{[0]} = \sigma^{[k]}$, the lemma is proven by setting $(\hat{g}, \hat{l}) = (g, l)$. Thus, from now on we consider the case $\sigma^{[0]} \neq \sigma^{[k]}$.

Informally, we will now shorten σ, producing a sequence θ containing no repetitions. Formally, the contraction is performed by Algorithm 5.

Algorithm 5: Contracting σ into $\mathrm{\theta}$.
Input: σ, k
Program variables: nonnegative integers s, j, sequence $\mathrm{\theta}$ over Loc
Output: s, $\mathrm{\theta}$

```
s := k;
θ := σ;
j := 0;
while j < s do
    if there is some t such that j < t ⩽ s and θ^[t] = θ^[j] then
        let t be the largest integer such that t ⩽ s and θ^[t] = θ^[j];
        remove the subsequence θ^[j+1] ... θ^[t] from θ;
        s := s − (t − j);
    j := j + 1;
```

The following loop invariant holds:
- $\mathrm{j} \leqslant \mathrm{s} = \mathsf{length}(\mathrm{\theta})$,
- $\mathrm{\theta}^{[0]} = \sigma^{[0]} \ \wedge\ \mathrm{\theta}^{[\mathrm{s}]} = \sigma^{[k]}$,
- for all $\bar{\jmath} < \mathrm{j}$, the local state $\mathrm{\theta}^{[\bar{\jmath}]}$ occurs in $\mathrm{\theta}$ exactly once, and
- $\forall\, \bar{\jmath} < \mathrm{s}\colon (\_, \mathrm{\theta}^{[\bar{\jmath}]}) \rightarrow_i (\_, \mathrm{\theta}^{[\bar{\jmath}+1]}) \ \vee\ \mathrm{\theta}^{[\bar{\jmath}]} = \mathrm{\theta}^{[\bar{\jmath}+1]}$.

Thus, we obtain the following postcondition of the program:
- $\mathrm{j} = \mathrm{s} = \mathsf{length}(\mathrm{\theta})$,
- $\mathrm{\theta}^{[0]} = \sigma^{[0]} \ \wedge\ \mathrm{\theta}^{[\mathrm{s}]} = \sigma^{[k]}$,
- for all $\bar{\jmath} < \mathrm{s}$, the local state $\mathrm{\theta}^{[\bar{\jmath}]}$ occurs in $\mathrm{\theta}$ exactly once, and
- $\forall\, \bar{\jmath} < \mathrm{s}\colon (\_, \mathrm{\theta}^{[\bar{\jmath}]}) \rightarrow_i (\_, \mathrm{\theta}^{[\bar{\jmath}+1]})$.

Since in each iteration s−j strictly decreases and stays nonnegative, the program always terminates.

Let θ be the last value of the program variable $\mathrm{\theta}$ and s the last value of the program variable s. As all the elements at positions smaller than s occur in θ exactly once, the last element must occur in θ also exactly once. So θ contains no repetitions at all, implying $s<L$. Since $\theta^{[0]} = \sigma^{[0]} \neq \sigma^{[k]} = \theta^{[s]}$, we must have $s\neq 0$, so $s\geqslant 1$.

According to the postcondition, there are families $\left(\check{e}^{[j]}\right)_{j<s}$ and $\left(\hat{e}^{[j]}\right)_{j<s}$ over Glob such that for each $j<s$ we have $\left(\check{e}^{[j]}, \theta^{[j]}\right) \to_i \left(\hat{e}^{[j]}, \theta^{[j+1]}\right)$. Let $\rightsquigarrow$ be the transition relation of h. Since h is strongly connected, Lemma V.2.2.1 implies that one can recursively construct a sequence $\tau = \left(\tau^{[j]}\right)_{j<s} \in (\mathsf{Loc}^m)^s$ of states of h such that $\left(g, (l_{f(r)})_{r<m}\right) \rightsquigarrow^{\leqslant\delta} \left(\check{e}^{[0]}, \tau^{[0]}\right)$ and $\left(\hat{e}^{[j]}, \tau^{[j]}\right) \rightsquigarrow^{\leqslant\delta} \left(\check{e}^{[j+1]}, \tau^{[j+1]}\right)$ for all $j < s-1$.

Now consider the sequence $\left((\bar{g}^{[j]}, \varphi^{[j]})\right)_{j\leqslant 2s}$ of program states of p, defined as follows:

- $\bar{g}^{[0]} = g$, $\varphi^{[0]} = l$,
- for odd $j < 2s$ let $\bar{g}^{[j]} = \check{e}^{[(j-1)/2]}$, $\varphi_i^{[j]} = \theta^{[(j-1)/2]}$, $\varphi_{f(r)}^{[j]} = \tau_r^{[(j-1)/2]}$ ($r<m$), and $\varphi^{[j]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$,
- for positive even $j \leqslant 2s$ let $\bar{g}^{[j]} = \hat{e}^{[j/2-1]}$, $\varphi_i^{[j]} = \theta^{[j/2]}$, $\varphi_{f(r)}^{[j]} = \tau_r^{[j/2-1]}$ ($r<m$), and $\varphi^{[j]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$.

Note:

- Since $s \geqslant 1$, we obtain $\left(\bar{g}^{[0]}, \left(\varphi_{f(r)}^{[0]}\right)_{r<m}\right) = \left(g, \left(l_{f(r)}\right)_{r<m}\right) \rightsquigarrow^{\leqslant\delta} \left(\check{e}^{[0]}, \tau^{[0]}\right) = \left(\bar{g}^{[1]}, \left(\varphi_{f(r)}^{[1]}\right)_{r<m}\right), \varphi_i^{[0]} = l_i = \sigma^{[0]} = \theta^{[0]} = \varphi_i^{[1]}, \varphi^{[0]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$ $= \varphi^{[1]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$, and so $\left(\bar{g}^{[0]}, \varphi^{[0]}\right) \longrightarrow^{\leqslant\delta} \left(\bar{g}^{[1]}, \varphi^{[1]}\right)$.
- For odd $j < 2s$ we have $\left(\bar{g}^{[j]}, \varphi_i^{[j]}\right) = \left(\check{e}^{[(j-1)/2]}, \theta^{[(j-1)/2]}\right) \to_i \left(\hat{e}^{[(j-1)/2]}, \theta^{[(j+1)/2]}\right) = \left(\bar{g}^{[j+1]}, \varphi_i^{[j+1]}\right)$, $\varphi_u^{[j]} = \tau_{f^{-1}(u)}^{[(j-1)/2]} = \varphi_u^{[j+1]}$ (for all $u \in \mathsf{img}\,f$), $\varphi^{[j]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = \varphi^{[j+1]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$, and so $\left(\bar{g}^{[j]}, \varphi^{[j]}\right) \longrightarrow \left(\bar{g}^{[j+1]}, \varphi^{[j+1]}\right)$.
- For positive even $j < 2s$ we have $\left(\bar{g}^{[j]}, \left(\varphi_{f(r)}^{[j]}\right)_{r<m}\right) = \left(\hat{e}^{[j/2-1]}, \tau^{[j/2-1]}\right)$ [since $j \leqslant 2s-2$, we get $\frac{j}{2} \leqslant s-1$, and so $\frac{j}{2} - 1 < s-1$] $\rightsquigarrow^{\leqslant\delta} \left(\check{e}^{[j/2]}, \tau^{[j/2]}\right) = \left(\bar{g}^{[j+1]}, \left(\varphi_{f(r)}^{[j+1]}\right)_{r<m}\right), \varphi_i^{[j]} = \theta^{[j/2]} = \varphi_i^{[j+1]}, \varphi^{[j]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = \varphi^{[j+1]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$, and so $\left(\bar{g}^{[j]}, \varphi^{[j]}\right) \longrightarrow^{\leqslant\delta} \left(\bar{g}^{[j+1]}, \varphi^{[j+1]}\right)$.
- Finally, $\varphi_i^{[2s]} = \theta^{[s]} = \sigma^{[k]}$ and $\varphi^{[2s]}|_{n\setminus((\mathsf{img}\,f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\,f)\cup\{i\})}$.

The above implies $\left(\bar{g}^{[0]}, \varphi^{[0]}\right) \longrightarrow^* \left(\bar{g}^{[2s]}, \varphi^{[2s]}\right)$; the walk from $\left(\bar{g}^{[0]}, \varphi^{[0]}\right)$ to $\left(\bar{g}^{[2s]}, \varphi^{[2s]}\right)$ goes through all the states of $\left((\bar{g}^{[j]}, \varphi^{[j]})\right)_{j\leqslant 2s}$ in ascending order (and possibly through more states when taking a subwalk from $\left(\bar{g}^{[j]}, \varphi^{[j]}\right)$ to $\left(\bar{g}^{[j+1]}, \varphi^{[j+1]}\right)$ for even nonnegative $j < 2s$). The number of odd positive integers below $2s$ is $\left|\{i \mid i \text{ odd} \wedge 1 \leqslant i \leqslant 2s-1\}\right| = \frac{(2s-1)-1}{2} + 1 = s$. The number of even positive integers below $2s$ is $\left|\{i \mid i \text{ even} \wedge 2 \leqslant i \leqslant 2s-2\}\right| = \frac{(2s-2)-2}{2} + 1 = s-1$. Thus, the length of the walk just mentioned is at most $\delta + s + \delta(s-1) = (\delta+1)s \leqslant (\delta+1)(L-1)$. We set $(\hat{g}, \hat{l}) \stackrel{\text{def}}{=} \left(\bar{g}^{[2s]}, \varphi^{[2s]}\right)$. ∎

Due to Lemma V.2.2.2, a strongly connected subprogram can "help" more than one thread to change shared states "quickly":

Theorem V.2.2.3. *Let $n \geqslant m \geqslant 1$, and let h be a strongly connected, m-threaded subprogram of an n-threaded program p. If G=1, then $\mathsf{diam}(p) \leqslant (L-1)n$. If L =1, then $\mathsf{diam}(p) \leqslant G-1$. If $G\geqslant 2$, then* $\mathsf{diam}(p) \leqslant \left(\min\left\{C, \left(\text{if } G{=}2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\right), \mathsf{diam}(h)\right\} + 1\right)(L-1)(n-m) + \mathsf{diam}(h)$.

Proof. If G=1, the strong connectivity of h is irrelevant, since h cannot exchange information with the remainder of the program, and Lemma V.1.1 gives an upper bound of $(L-1)n$ for the diameter of an n-threaded program.

If L=1, Lemma V.1.2 provides us with an upper bound of $G-1$.

Now we turn to the general case $G \geqslant 2$.

If n=m, both sides of the inequality in question coincide; thus assume from now on that $n>m$. Let $(\to_0, \ldots, \to_{n-1}) = p$, and let $\longrightarrow$ be the transition relation of p. Let f be the embedding of h into p. Let $\zeta = \Big(\min\Big\{C, \big(\text{if } G{=}2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\big),$
$\mathsf{diam}(h)\Big\} + 1\Big)(L-1)$.

Consider program states (g, l), (g', l') of p such that $(g, l) \longrightarrow^* (g', l')$. Let $A = \{i \in n \setminus (\mathsf{img}\, f) \mid l_i{\neq}l'_i\}$ be the set of identifiers of threads outside the embedded program h whose initial and final local states differ.

We start by considering the special case that A is empty. Then $l|_{n\setminus(\mathsf{img}\, f)} = l'|_{n\setminus(\mathsf{img}\, f)}$. Since h is strongly connected, it exhibits a path τ from $\big(g, (l_{f(r)})_{r<m}\big)$ to $\big(g', (l'_{f(r)})_{r<m}\big)$ of length at most $\mathsf{diam}(h)$. We lift τ to a path $\hat{\tau}$ from (g, l) to (g', l') in the transition graph of p by reindexing the local states in τ according to f and adding a tuple of local states $(l_r)_{r\in n\setminus(\mathsf{img}\, f)}$ to each state in τ. Certainly, $\mathsf{length}(\hat{\tau}) \leqslant \mathsf{diam}(h) \leqslant \zeta(n-m) + \mathsf{diam}(h)$, which proves the lemma.

From now on we consider the other case, viz. that A is nonempty. Take any path $\sigma = (\sigma^{[j]})_{j\leqslant k}$ in p from (g, l) to (g', l'). Enumerate the elements of A in ascending order by $i_0 < \cdots < i_{s-1}$ for $s = |A|$. We are going to recursively construct nonempty paths $\varphi_0, \ldots, \varphi_{s-1}$ in the transition graph of p such that the initial state of φ_0 is (g, l), $\mathsf{length}(\varphi_0) \leqslant \zeta$, and for all $j<s$ we have:

- φ_j is nonempty,
- $j > 0 \Rightarrow \mathsf{length}(\varphi_j) < \zeta$,
- the concatenated sequence $\varphi_0 \ldots \varphi_j$ is a walk, and
- the final state of φ_j is of the form $\left(\_, \left(\begin{cases} \_, & \text{if } r \in \mathsf{img}\, f\,, \\ l'_r\,, & \text{if } r \notin \mathsf{img}\, f \ \wedge\ r \leqslant i_j\,, \\ l_r\,, & \text{if } r \notin \mathsf{img}\, f \ \wedge\ i_j < r \end{cases}\right)_{r<n}\right)$.

For this purpose, let $j<s$ be arbitrary, and assume that for all $j'<j$ the paths $\varphi_{j'}$ as above are already constructed.

Case j=0. Notice that $\sigma^{[0]}_{i_0} = l_{i_0}$ and $\forall\, \bar{\jmath} < k\colon\ (\_, \sigma^{[\bar{\jmath}]}_{i_0}) \to_{i_0} (\_, \sigma^{[\bar{\jmath}+1]}_{i_0}) \vee \sigma^{[\bar{\jmath}]}_{i_0} = \sigma^{[\bar{\jmath}+1]}_{i_0}$. By Lemma V.2.2.2, there is a program state $(\hat{g}, \hat{l})$ of p such that $(g, l) \longrightarrow^{\leqslant\zeta} (\hat{g}, \hat{l})$, $\sigma^{[k]}_{i_0} = \hat{l}_{i_0}$, and $\hat{l}|_{n\setminus((\mathsf{img}\, f)\cup\{i_0\})} = l|_{n\setminus((\mathsf{img}\, f)\cup\{i_0\})}$. We define φ_0 as a path of length $\leqslant\zeta$ that takes (g, l) to $(\hat{g}, \hat{l})$. Certainly, φ_0 is nonempty. Now consider an arbitrary $r \in n \setminus (\mathsf{img}\, f)$.

Case $r \leqslant i_0$. If $r \notin A$, we have especially $r \in n \setminus ((\mathsf{img}\, f) \cup \{i_0\})$ and therefore $\hat{l}_r = l_r = l'_r$. If $r \in A$, we have $r = i_0$ and therefore $\hat{l}_r = \hat{l}_{i_0} = \sigma^{[k]}_{i_0} = l'_{i_0} = l'_r$.

Case $i_0 < r$. Then $r \in n \setminus ((\mathsf{img}\, f) \cup \{i_0\})$, and so $\hat{l}_r = l_r$.

Case j>0. Let $(\check{g}, \check{l})$ be the final state of φ_{j-1}, which exists because φ_{j-1} is nonempty. Note that $\check{l}_r = l'_r$ for $r \in n\setminus(\mathsf{img}\, f) \wedge r \leqslant i_{j-1}$, $\check{l}_r = l_r$ for $r \in n\setminus(\mathsf{img}\, f) \wedge r > i_{j-1}$, $\sigma^{[0]}_{i_j} = l_{i_j} = \check{l}_{i_j}$, and $\forall\, \bar{\jmath} < k\colon\ (\_, \sigma^{[\bar{\jmath}]}_{i_j}) \to_{i_j} (\_, \sigma^{[\bar{\jmath}+1]}_{i_j}) \vee \sigma^{[\bar{\jmath}]}_{i_j} = \sigma^{[\bar{\jmath}+1]}_{i_j}$. Lemma V.2.2.2 implies the existence of a program state $(\hat{g}, \hat{l})$ of p such that $(\check{g}, \check{l}) \longrightarrow^{\leqslant\zeta} (\hat{g}, \hat{l})$, $\sigma^{[k]}_{i_j} = \hat{l}_{i_j}$, and $\hat{l}|_{n\setminus((\mathsf{img}\, f)\cup\{i_j\})} = \check{l}|_{n\setminus((\mathsf{img}\, f)\cup\{i_j\})}$. We define φ_j as a path obtained from an evidence path for $(\check{g}, \check{l}) \longrightarrow^{\leqslant\zeta} (\hat{g}, \hat{l})$ by stripping the source state. Now let $r \in n \setminus (\mathsf{img}\, f)$.

Case $r \leqslant i_j$.

Case $r \leqslant i_{j-1}$. Then $r \in n \setminus ((\mathsf{img}\, f) \cup \{i_j\})$, and so $\hat{l}_r = \check{l}_r = l'_r$.

Case $r > i_{j-1}$. If $r \notin A$, we have $r \in n \setminus ((\mathsf{img}\, f) \cup \{i_j\})$ and so $\hat{l}_r = \check{l}_r = l_r = l'_r$. If $r \in A$, then $r = i_j$ and so $\hat{l}_r = \hat{l}_{i_j} = \sigma_{i_j}^{[k]} = l'_{i_j} = l'_r$.

Case $i_j < r$. Then $r \in n \setminus ((\mathsf{img}\, f) \cup \{i_j\})$ and $r > i_{j-1}$, so we have $\hat{l}_r = \check{l}_r = l_r$.

Since $\hat{l}_{i_j} = l'_{i_j} \neq l_{i_j} = \check{l}_{i_j}$, we obtain that φ_j is nonempty.

After construction we thus obtain a walk of length not exceeding ζs from (g, l) to a state of the form $\left(\_, \left(\begin{cases} \_, & \text{if } r \in \mathsf{img}\, f\,, \\ l'_r\,, & \text{if } r \notin (\mathsf{img}\, f) \wedge r \leqslant i_{s-1}\,, \\ l_r\,, & \text{if } r \notin (\mathsf{img}\, f) \wedge i_{s-1} < r \end{cases}\right)_{r<n}\right)$, which is by definition of s and A of the form $\left(\_, \left(\begin{cases} \_, & \text{if } r \in \mathsf{img}\, f\,, \\ l'_r\,, & \text{otherwise.} \end{cases}\right)_{r<n}\right)$. Since h is strongly connected, it exhibits a path τ from the h-components of the last state of φ_{s-1} to $\left(g', (l'_{f(r)})_{r<m}\right)$ of length at most $\mathsf{diam}(h)$. We lift τ to a path $\hat{\tau}$ from the last state of φ_{s-1} to (g', l') in the transition graph of P by reindexing the local states in τ according to f and adding the constant tuple of local states $(l'_r)_{r \in n \setminus (\mathsf{img}\, f)}$ to each state in τ. Certainly, $\mathsf{length}(\hat{\tau}) \leqslant \mathsf{diam}(h)$. Let ψ be $\hat{\tau}$ without its initial state. Then the walk $\varphi_0 \ldots \varphi_{s-1}\psi$ has length at most $\zeta s + \mathsf{diam}(h) \leqslant \zeta(n - m) + \mathsf{diam}(h)$. ■

Let us restate the contents of Theorem V.2.2.3 for program families:

Corollary V.2.2.4. *Let $m \geqslant 1$ and $(p_n)_{n \geqslant m}$ be a family of multithreaded programs such that*
- *the program p_m is strongly connected, and*
- *for all $n \geqslant m$, the program p_n has exactly n threads, and*
- *for all $n \geqslant m$, the program p_m is a subprogram of p_n.*

If $G = 1$, then $\mathsf{diam}(p_n) \leqslant (L-1)n$. If $L = 1$, then $\mathsf{diam}(p_n) \leqslant G-1$. If $G \geqslant 2$, then for $d = \mathsf{diam}(p_m)$ and $\zeta = \left(\min\left\{C, \left(\text{if } G{=}2 \text{ then } L \text{ else } (G-1)L^m - L^{m-1} + 1\right), d\right\} + 1\right)(L-1)$ we have $\forall n \geqslant m\colon \mathsf{diam}(p_n) \leqslant \zeta n - \zeta m + d$. □

Note V.2.2.5 (On the tightness of upper bounds mentioned in Theorem V.2.2.3 and Corollary V.2.2.4)**.** In case G=1, the upper the bound $(L-1)n$ is tight, as can be demonstrated by a program in which the graph of transitions of some thread is a directed Hamiltonian cycle on $\mathsf{Glob}{\times}\mathsf{Loc}$, and the transitions graphs of the other threads are non-circular directed Hamiltonian paths on $\mathsf{Glob}{\times}\mathsf{Loc}$.

In case L=1, the upper the bound $G-1$ is also tight, as can be demonstrated by a program in which the graph of transitions of some thread is a directed Hamiltonian cycle on $\mathsf{Glob}{\times}\mathsf{Loc}$, and the thread transition relations of the other threads are empty.

In case $G, L \geqslant 2$ and m=1, the upper bound $\left(\min\left\{C, \left(\text{if } G{=}2 \text{ then } L \text{ else } (G-1)L\right), \mathsf{diam}(h)\right\} + 1\right)(L-1)(n-1) + \mathsf{diam}(h) \leqslant \left((G-1)L + 1\right)(L-1)(n-1) + \mathsf{diam}(h) \leqslant (GL-L+1)(L-1)(n-1)+GL-1 = (GL-L+1)(L-1)n-(GL^2-L^2+L-GL+L-1) + GL - 1 = (GL - L + 1)(L-1)n - GL^2 + L^2 - 2L + GL + 2GL = (GL - L + 1)(L-1)n + (-GL + L - 2 + 2G)L = (GL - L + 1)(L-1)n + ((2-L)G + L - 2)L =$

$(GL - L+1)(L{-}1)n + (2{-}L)(G{-}1)L$ is likely to be tight. We expect that it can be demonstrated by an adaptation of the family from the proof of Theorem V.1.3, in which thread 0 additionally obtains a reverse transition for each transition it already has; we skip the corresponding proof in this paper.

For the most general case of $G, L, m \geqslant 2$, the question whether the bound is tight is open. □

Other examples for Theorem V.2.2.3 and Corollary V.2.2.4 can be obtained by adding more threads to N2T6B, N2T7B, or N3T9 from § III: each of these programs is strongly connected.

Notice that the upper bounds in Theorem V.2.2.3 and Corollary V.2.2.4 are (sub-) linear in n, as the coefficient with which n is multiplied can be bounded above by $(C{+}1)(L{-}1)$, which does not depend on the family of programs.

Also remark that for $G{\geqslant}2$, the if-then-else expression inside the aforementioned coefficient can be simplified to its "else" branch which is at least as large as the "then" branch. The cost paid is a loosening of the inequality for $G{=}2$.

V.2.3. Probabilistic Analysis

(We thank Steffen Borgwardt for the basic insight of this section.)

We proceed by considering a random process of creating a multithreaded program in which each transition of each thread is chosen independently with a probability that depends only on the thread transition.

Due to Lemmas V.1.1 and V.1.2, in case $1 \in \{G, L\}$ the diameter of an n-threaded program is at most linear in n with probability 1 regardless of the random process. So from here to the end of this section, we consider the general case $G, L \geqslant 2$.

First, we show that the existence of a thread with a special set of transitions leads to a polynomial bound on the diameter which is linear in n.

Lemma V.2.3.1. *If the graph of thread transitions of some thread in a program is strongly connected, then the diameter of the program does not exceed* $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. *Formally: for every n-threaded program $p = (\to_i)_{i<n}$, we have* $\big(\exists\, i < n\ \forall\, g, g' \in \mathsf{Glob},\ l, l' \in \mathsf{Loc}\colon (g, l) \to_i^* (g', l')\big) \ \Rightarrow\ \mathsf{diam}(p) \leqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$.

Proof. Consider an n-threaded program $p = (\to_i)_{i<n}$ such that for some $i{<}n$, the graph $(\mathsf{Glob}{\times}\mathsf{Loc}, \to_i)$ is strongly connected. Notice that this graph has GL nodes, so any path (which is, by definition, not self-intersecting) in this graph has at most $GL-1$ edges. In particular, if h is the program consisting of only the thread $\to_i$, then $\mathsf{diam}(h) \leqslant GL-1$. Theorem V.2.2.3 implies $\mathsf{diam}(p) \leqslant (\min\{(G{-}1)L, GL{-}1\}{+}1)(L{-}1)(n{-}1){+}GL{-}1 = (\min\{GL{-}L, GL{-}1\}{+}1)(L{-}1)(n{-}1){+}GL{-}1 \leqslant ((GL{-}L){+}1)(L{-}1)(n{-}1){+}GL{-}1 = (GL - L + 1)(L{-}1)n - (GL - L + 1)(L{-}1) + GL - 1 = (GL - L + 1)(L{-}1)n - (GL^2 - L^2 + L - GL + L - 1) + GL - 1 = (GL - L + 1)(L{-}1)n - (GL^2 - L^2 - GL + 2L - 1) + GL - 1 = (GL - L + 1)(L{-}1)n - GL^2 + L^2 + GL - 2L + 1 + GL - 1 = (GL{-}L{+}1)(L{-}1)n{-}GL^2{+}L^2{+}2GL{-}2L = (GL{-}L{+}1)(L{-}1)n{+}({-}GL{+}L{+}2G{-}2)L = (GL{-}L{+}1)(L{-}1)n{+}({-}L(G{-}1){+}2(G{-}1))L = (GL{-}L{+}1)(L{-}1)n{+}(2{-}L)(G{-}1)L$. ■

Lemma V.2.3.1 is our main ingredient in proving a linear diameter of programs for which the number of threads becomes sufficiently large—given a rather general probability distribution for the existence of thread transitions as mentioned before.

Theorem V.2.3.2. *Let* $\pi\colon (\mathsf{Glob}{\times}\mathsf{Loc})^2 \to [0, 1]$ *be a probability distribution on thread transitions for which* $\forall\, g, g' \in \mathsf{Glob},\ l, l' \in \mathsf{Loc}\ \exists\, k \in \mathbb{N}_{\geqslant 0},\ (\tilde{l}_i)_{i=0}^k \in (\mathsf{Loc}^n)^{k+1},\ (\tilde{g}_i)_{i=0}^k \in$

Glob^{k+1}: $(g,l) = (\tilde{g}_0, \tilde{l}_0) \wedge (g', l') = (\tilde{g}_k, \tilde{l}_k) \wedge \forall i{<}k\colon \pi((\tilde{g}_i, \tilde{l}_i), (\tilde{g}_{i+1}, \tilde{l}_{i+1})) > 0$. *Then* $\lim_{n\to\infty} \mathrm{Prob}\big(\mathsf{diam}(\textit{an n-threaded random program}) \leqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L\big) = 1$.

Proof. The probability for a thread to satisfy the prerequisites for Lemma V.2.3.1, i.e., to have a strongly connected graph of thread transitions, is strictly positive. With growing n, the probability that a random n-threaded program does not contain such a thread approaches 0. ∎

Note V.2.3.3. In the above claims, the bound $(GL - L + 1)(L{-}1)n + (2{-}L)(G{-}1)L$ on the diameter is linear in the number of threads, and can be simplified to a mathematically larger but conceptually easier expression GL^2n.

Proof. Let $e = (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$.

Case $L{=}1$. Then $e = 0n + (G{-}1) < GL^2n$.

Case $L{\geqslant}2$. Then $e < (GL)Ln + 0 = GL^2n$. ∎

The probability distribution above is given, e.g., when the probability of each thread transition is positive, no matter how small it is.

Informally, the consequence is that trying to find programs with "large" diameter by generating n-threaded programs, for instance, uniformly at random would most likely fail for large n: we would likely get only at-most–linear values. In contrast, the search from [64], for example, is nonrandom, structured, and informed.

Theorem V.2.3.2 models the risks of a large number of threads being affected externally. Examples are radiation in high-performance computing and row-hammering attacks on server memory [49]. Assuming that the operating system with the scheduler is stored in better-protected memory (which is a reasonable requirement on system builders) and that user-space threads are less protected, and assuming sufficiently long running-times, irreparable bit-flips are likely to turn the threads into random pieces of code while still maintaining their parallel execution. In particular, Theorem V.2.3.2 is an indication that if an "error" program state ever becomes reachable from the current execution state due to bit-flips, it is also likely to become reachable "quickly," i.e., in time linear in n. Our argument even extends to deterministic threads (in which each thread state has at most one successor, and which can be actually executed as opposed to their nondeterministic counterparts) if we additionally assume that the probability of each thread transition is strictly below 1: then, a program asymptotically almost surely contains a thread whose transition graph is a Hamiltonian cycle on $\mathsf{Glob} \times \mathsf{Loc}$ (such a cycle is both deterministic and strongly connected), and we can apply Theorem V.2.2.3.

Of course, the usual caveats apply: it is a simplification of the reality that the probability of a thread transition depends only on the thread transition itself and that the executions are sequentially consistent in the interleaving semantics. So a closer examination of hardware-near execution models might be required to find out whether the above claim applies to more-realistic machine code, revealing additional risks of long-running, massively parallel software.

VI. Complexity of (non-)reachability

Now we determine the complexity of proving or refuting the reachability of a target program state or a target thread state from a source program state.

For the purpose of defining these reachability problems as formal languages, we assume without loss of generality the following:

- A shared state is an ordinal below G, and it is encoded in binary occupying exactly $\lfloor \log G \rfloor + 1$ bits.
- A local state is an ordinal below L, and it is encoded in binary occupying exactly $\lfloor \log L \rfloor + 1$ bits.
- A thread identifier is an ordinal below n, and it is encoded in binary occupying exactly $\lfloor \log n \rfloor + 1$ bits.

If some binary number actually needs fewer bits than allocated, it is padded with zeros.

In the language definitions below, "a program over G and L" is a program such that its set of shared states is G, and its set of local states is L (we emphasize this to create a textual difference to the language definitions in § II.3):

$$\mathsf{PReach} \overset{\text{def}}{=} \{(p,s,s') \mid p \text{ is a program over } G \text{ and } L \ \wedge\ s,s' \in \mathsf{State}_p \ \wedge\ \mathsf{d}_p(s,s') < \infty\},$$

$$\mathsf{PNonReach} \overset{\text{def}}{=} \{(p,s,s') \mid p \text{ is a program over } G \text{ and } L \ \wedge\ s,s' \in \mathsf{State}_p \ \wedge\ \mathsf{d}_p(s,s') = \infty\},$$

$$\mathsf{PReach}^{\mathsf{loc}} \overset{\text{def}}{=} \left\{(p,s,i,\tau) \,\middle|\, \begin{array}{l} p \text{ is a program over } G \text{ and } L \ \wedge\ s \in \mathsf{State}_p \\ \wedge\ i<n \ \wedge\ \tau \in \mathsf{Glob}\times\mathsf{Loc} \ \wedge\ \mathsf{d}^{\mathsf{loc}}_p(s,i,\tau) < \infty \end{array}\right\}, \quad \text{and}$$

$$\mathsf{PNonReach}^{\mathsf{loc}} \overset{\text{def}}{=} \left\{(p,s,i,\tau) \,\middle|\, \begin{array}{l} p \text{ is a program over } G \text{ and } L \ \wedge\ s \in \mathsf{State}_p \\ \wedge\ i<n \ \wedge\ \tau \in \mathsf{Glob}\times\mathsf{Loc} \ \wedge\ \mathsf{d}^{\mathsf{loc}}_p(s,i,\tau) = \infty \end{array}\right\}.$$

The letter P starting the names of the languages above indicates that G and L are the hidden constant *p*arameters of these languages; the number of threads is implicitly existentially quantified in the class terms.

First, we deal with program-state–to–program-state reachability:

Theorem VI.1. $\mathsf{PReach}, \mathsf{PNonReach} \in \mathsf{NSpace}(\log n)$.

Proof. From a high-level view, our algorithm is going to perform a nondeterministic search in the Petri net obtained by a symmetry reduction of an input program using state confusion $\approx$ in its reformulation from Corollary V.2.1.12. The Turing machine M that we are going to construct will implement this search.

Given an input on the read-only Turing tape, M first syntactically checks whether the input is a properly encoded triple of a program and two of its states. Then M determines the number of threads n and stores it in binary on the working tape. If n=1, the problem is solved by looking up in a constant table; thus, we assume $n \geqslant 2$ from now on.

To explain our procedure further, we let $(\rightarrow_i)_{i<n} = p$ be the program, $(g,l) = s$ the source program state, and $(g',l') = s'$ the target one. We fix the standard permutations ι_i ($i<n$) from Definition V.2.1.1. Notice that the value of a standard permutation for given $i<n$ and $a \in \mathsf{Loc}$ can be computed on the working tape using $\mathsf{O}(\lfloor \log n \rfloor + 1)$ space (where L is considered constant). Next, we allocate $L2^{GL(GL-1)}$ counters of $\lceil \log(n+1) \rceil$ bits each on the working tape of M. We will refer to these counters as $c_{\rightrightarrows,a}$ for $\rightrightarrows\ \in E$ and $a \in \mathsf{Loc}$, where E has been defined in Definition V.2.1.10. In the following state exploration, a newly reached program state $(\hat{g}, \hat{l})$ will be tracked in these counters: $c_{\rightrightarrows,a}$ will store $|\{t<n \mid \rightsquigarrow_t\, = \,\rightrightarrows \wedge \iota_t(\hat{l}_t) = a\}|$ for each $\rightrightarrows\ \in E$ and $a \in \mathsf{Loc}$. (Note that the query $\rightsquigarrow_t\, = \,\rightrightarrows \wedge \iota_t(\hat{l}_t) = a$ can be decided by constructing $\rightsquigarrow_t$ and $\iota_t(\hat{l}_t)$ on the working tape and comparing them to $\rightrightarrows$ and a; one can reuse temporary space on the working tape for doing this for all $t<n$, consuming $\mathsf{O}(\lfloor \log n \rfloor + 1)$ space in total, where G and L are considered constant.) The machine also allocates $\lceil \log(G{+}1) \rceil$ bits for storing the shared state $\hat{g}$. Initially, the shared state g will be stored there, while each counter $c_{\rightrightarrows,a}$ will be initialized to $|\{t<n \mid \rightsquigarrow_t\, = \,\rightrightarrows \wedge \iota_t(l_t) = a\}|$ for $\rightrightarrows\ \in E$ and $a \in \mathsf{Loc}$. In addition,

we allocate an extra counter, called v, which stores the number of visited equivalence classes wrt. $\approx$; this counter is initialized to 1.

After initialization, M performs a nondeterministic search in a loop as follows. First of all, M checks whether v exceeds $G(2n)^{(L-1)2^{GL(GL-1)}}$. If so, M halts, since it has visited some equivalence class twice (according to the proof of Theorem V.2.1.22). Otherwise, M continues and checks whether s' (or any state equivalent to s') has been reached. To do this, M checks, according to Corollary V.2.1.12, whether g' is equal to the current shared part and l' has exactly as many threads of different classes as described by the counters $c_{\rightrightarrows,a}$ for $\rightrightarrows \in E$ and $a \in \mathsf{Loc}$. If it is the case, M accepts: then an execution from s to s' exists. Otherwise, M nondeterministically chooses a thread index $i<n$, chooses a non–self-loop thread transition $((\hat{g},a),(\hat{g}',a')) \in \rightarrow_i \setminus D$, and constructs the "normalized" form $((\hat{g},\iota_i(a)),(\hat{g}',\iota_i(a'))) \in \rightsquigarrow_i$. Examining the counters and the shared-state variable, M determines whether the thread transition is applicable: M tests whether the shared-state variable stores $\hat{g}$ and whether $c_{\rightsquigarrow_i,\iota_i(a)} > 0$. If the result of this test is negative, M halts, otherwise M updates the shared-state variable and the counters according to the thread transition. Such an update stores $\hat{g}'$, decrements $c_{\rightsquigarrow_i,\iota_i(a)}$, and increments $c_{\rightsquigarrow_i,\iota_i(a')}$. Moreover, M increments v. After that, M goes to the loop start.

To show that all program states reachable from s are explored by M, note that M examines an overapproximating abstraction:

- confusable states are not distinguished, and
- at each loop iteration, all applicable non-loop thread transitions are available for the nondeterministic choice, so M considers all successors of the current state.

To show that only program states that are reachable from s are explored, notice that, in each loop iteration, if a currently considered program state $\tilde{s}$ is reachable from s, all program states that are confusable with $\tilde{s}$ are also reachable from s according to Lemma V.2.1.13. So the analysis has no opportunity to jump to an unreachable program state from $\tilde{s}$.

Thus, M explores exactly the states reachable from s.

Throughout any execution of M, the sum $\sum_{\rightrightarrows \in E,\, a \in \mathsf{Loc}} c_{\rightrightarrows,a}$ remains constant; therefore, $\mathsf{O}(\log n)$ space suffices for each of the $L2^{GL(GL-1)}$ counters. Taking into account also the shared-state variable of size $\lceil \log(G{+}1) \rceil$ on the working tape and temporary counters of for navigating through the input as well as for computing the values of ι, M consumes $\mathsf{O}(\log n)$ space on the working tape regardless of the computation branch and always halts (whether accepting or not). Since the input size is proportional to n, we have shown $\mathsf{PReach} \in \mathsf{NSpace}(\log n)$.

Since $\mathsf{NSpace}(\log n)$ is closed under complementation [44, 86], the non-reachability problem also belongs to $\mathsf{NSpace}(\log n)$. ■

The nondeterministic logarithmic space complexity of reachability is low but occurs sometimes; see, e.g., [4, Fig. 2].

For program-state–to–thread-state reachability, recall that low-complexity problems tend to be sensitive to the input format. So we now provide the details of the encoding of $\mathsf{PReach}^{\mathsf{loc}}$ and $\mathsf{PNonReach}^{\mathsf{loc}}$ into bitstrings. A thread transition relation is encoded using a bitstring of length $|(\mathsf{Glob} \times \mathsf{Loc})^2| = G^2L^2$. An n-threaded program p is stored as a list of thread transition relations in a self-delimited way. (For example, insert 0 between each pair of thread transition relations and append 1 at the end. This encoding uses $(G^2L^2+1)n$ bits. Using a slightly more compact self-delimiting encoding [57] would not change the complexity class.) An element of Loc^n is represented as a string of $n(\lfloor \log L \rfloor + 1)$ bits assuming that n is known. A program state $(g, l) \in \mathsf{Glob} \times \mathsf{Loc}^n$ is formed by prepending the encoding of g to the encoding of l, again assuming that n is known. The number i from the third component of the quadruple $(p, s, i, \tau) \in \mathsf{PReach}^{\mathsf{loc}} \cup \mathsf{PNonReach}^{\mathsf{loc}}$ is stored in binary and not in a self-delimiting way in the bits right after s. The encoding of i is followed by an encoding of τ using the final $(\lfloor \log G \rfloor + 1) + (\lfloor \log L \rfloor + 1)$ bits.

We expect that slight encoding variations (e.g., in the self-delimiting encoding of p) will not change the complexity class.

Theorem VI.2. $\mathsf{PReach}^{\mathsf{loc}}, \mathsf{PNonReach}^{\mathsf{loc}} \in \mathsf{NC}^1$.

Proof. We will describe of a logspace-uniform family of circuits that will recognize $\mathsf{PReach}^{\mathsf{loc}}$ similarly to how the machine M from the proof of Theorem VI.1 does it. Given $m \in \mathbb{N}_{\geqslant 0}$, we now describe a bounded–fan-in circuit with m inputs, polynomial size in m, and $\mathrm{O}(\log m)$ depth that recognizes all words in $\mathsf{PReach}^{\mathsf{loc}}$ of length m. A part of this circuit guesses $n < m$ by a balanced "or" tree and then checks, by a logarithmically deep circuit, that p is indeed a description of an n-threaded program (with the aforementioned self-delimited encoding, a balanced "and" tree can check that positions $(G^2L^2 + 1)j - 1$ contain zeros for positive $j < n$ and one for $j = n$). For each valid choice of n, interpret the $\lfloor \log G \rfloor + 1 + (\lfloor \log L \rfloor + 1)n$ bits following the description of p as a description of $s \in \mathsf{Glob} \times \mathsf{Loc}^n$. Then interpret all the bits at positions $(G^2L^2+1)n+\lfloor \log G \rfloor+1+(\lfloor \log L \rfloor+1)n = (G^2L^2+\lfloor \log L \rfloor+2)n+\lfloor \log G \rfloor+1$ till $m - ((\lfloor \log G \rfloor + 1) + (\lfloor \log L \rfloor + 1)) - 1 = m - \lfloor \log G \rfloor - \lfloor \log L \rfloor - 3$ as the binary encoding of i. Interpret the remaining positions $m - \lfloor \log G \rfloor - \lfloor \log L \rfloor - 2$ till $m - 1$ as τ. Check that $i < n$ by a logarithmically deep circuit. (For example, convert n into binary by a recursive three-integers–to–two-integers addition, find the most significant bit position in which the binary representations of n and i differ, and check that in that position, the bit of the representation of n is larger than the corresponding bit of i.)

Now, we fix an arbitrary pair $(n, i) \in \mathbb{N}_+ \times \mathbb{N}_{\geqslant 0}$ such that $i < n$. We are left with the problem of describing an NC^1 circuit that determines whether $\mathsf{d}_p^{\mathsf{loc}}(s, i, \tau) < \infty$. We are going to show how a Turing machine with a read-only input tape and a space-bounded read-write output tape, which we will construct and call M^{loc}, would decide this problem. In the following explanation, $(\rightarrow_i)_{i<n} = p$ will be the program, $(g, l) = s$ the source program state, and $(g', l') = \tau$ the target thread state. The machine M^{loc} allocates $2^{GL(GL-1)}$ binary counters on the working tape; each binary counter has a width of $\lfloor \log C \rfloor + 1$ bits, where C is the maximal local diameter (cf. Definition and Corollary IV.3.8). These counters will be referred to as $\tilde{c}_{\rightrightarrows}$ ($\rightrightarrows \in E$) and initialized by $\tilde{c}_{\rightrightarrows} := \min\{C, |\{t{<}n \mid \rightsquigarrow_t = \rightrightarrows\}|\}$ ($\rightrightarrows \in E$). (Again, counting can be done with a recursive three-integers–to–two-integers addition, which is performed till the bit width of C is exceeded; taking the minimum can be done by finding the most significant bit in which two numbers differ.) The machine also copies the shared state g, the normalized thread transition relation $\rightsquigarrow_i$, and the "normalized" target thread state $\tilde{\tau} = (g', \iota_i(l'))$ to the working tape. In the abstract, M^{loc} would need to explore the state space of a program that starts in the shared state g and has $\tilde{c}_{\rightrightarrows} \leqslant C$ threads starting in the local state l_0 and having the transition relation $\rightrightarrows$ (for $\rightrightarrows \in E$); the goal is to decide on the reachability of $\tilde{\tau}$ in any thread with the transition relation $\rightsquigarrow_i$. Instead of actually performing the search, M^{loc} uses the fact that the number of decision questions of the above kind is finite (namely, below $G \cdot (C + 1)^{2^{GL(GL-1)}} \cdot 2^{GL(GL-1)} \cdot GL$), so M^{loc} simply looks up in a table stored in the machine state. (Notice: although we do not know the actual yes/no answers to these decision questions, we still know that such a finite table of answers exists.)

To see that M^{loc} is sound with respect to the reachability question for the original program p, note that if M^{loc} answers "reachable", then the thread state τ of the i^{th} thread is indeed reachable from s in p: adding more threads beyond C that do not take any steps would not destroy reachability.

To see that M^{loc} is complete with respect to the reachability question for the original program p, note that if in p there is a walk from the program state s to the thread state τ of the thread i, there is also a path of length not exceeding C from s to the thread state τ of the thread i. Therefore, at most C threads of any equivalence class from ${}^{n}\!/\!_{\sim}$ take steps in this path; all the other threads do not perform any steps. Throwing out these other "lazy" threads, we obtain a path from a "sub-state" of s to the thread state τ of thread i in a program in which there are no more than C threads of any equivalence class from ${}^{n}\!/\!_{\sim}$. The existence of this path will be reported by M^{loc} due to its construction.

Thus, M^{loc} recognizes $\mathsf{PReach}^{\mathsf{loc}}$.

Since M^{loc} takes bounded working-tape space, M^{loc} can be converted into a finite automaton, which uses zero space on the working tape. The existence of an accepting path in the finite automaton from the initial state to the final state can be decided with an NC^1 circuit using the standard recursive divide-and-conquer construction.

As for $\mathsf{PNonReach}^{\mathsf{loc}}$, the proof is almost the same as above, except that M^{loc}, instead of asking $\mathsf{d}_p^{\mathsf{loc}}(s, i, \tau) < \infty$, now asks $\mathsf{d}_p^{\mathsf{loc}}(s, i, \tau) = \infty$. ∎

We do not expect the complexity to drop significantly below NC^1, since checking whether a unary-encoded number is larger than a binary-encoded number ($n > i$ in the problem formulation) cannot be performed by a finite automaton.

VII. Discussion and conclusion

In the present paper, we tackled the case of arbitrary but constant shared and local state space. We derived the bounds $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L \leqslant \mathsf{diamax}(n) < G\binom{n/t+L-1}{L-1}^t$, where $t = 2^{GL(GL-1)}$, on the maximum diameter of an n-threaded program with G shared states and L local states for all n. Notice that the exponent does not depend on the number of threads. The lower bound was proven by constructing an infinite family of explicit programs such that the n^{th} program has n threads and diameter $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. The upper bound is both a strengthening and a generalization of the corresponding bound from [64]. The mathematical computations behind these bounds are tight in the following sense: if better yet simple bounds exist, they would require genuinely new proof ideas. Moreover, we discovered the following results:

- a polynomial upper bound for the diameter for a subclass of multithreaded programs; this bound is linear in n and, in certain cases, agrees with the lower bound;
- a polynomial upper bound for a rather general class of probability distributions and a randomly chosen program; this bound is linear in n and matches the lower bound;
- an upper bound on the local diameter that does not depend on n; the function mapping G and L to the least upper bound is computable, and the computation algorithm can be readily extracted from the proofs;
- (non-)reachability of target program states from source program states is decidable in $\mathsf{NSpace}(\log n)$;
- (non-)reachability of target thread states from source program states is decidable in NC^1 (which strengthens and generalizes the corresponding result from [64]).

(Sub-)polynomial characterizations of this type are desirable as they, besides being aesthetically pleasing, give hope for practical algorithms.

(Besides, in the unparametrized case, we formally proved PSpace-completeness of reachability and nonreachability of program states and thread states in multithreaded programs with respect to logspace reductions.)

So far, no infinite family of n-threaded programs with constant shared and local space sizes and superlinear diameter (in n) is known. This is in stark discrepancy to the presence of examples of sequential programs whose diameter is exponential in the number of variables and in extreme discrepancy to the presence of Petri nets computing non-primitive recursive functions [69]. The polynomial bound on diamax implies that the distances in the transition graph of a program are at most polylogarithmic in the size of this graph (whereas the general upper bound on the distances in an arbitrary graph is linear rather than polylogarithmic).

From a purely theoretical viewpoint, searching for counterexamples (to certain non-reachability properties) in certain programs with depth-first search (DFS) can be improved now: the search depth can be bounded by $\mathsf{O}(n^c)$ for some $c > 1$ not depending on n for nonlocal properties and by a constant for local properties. Thus, the worst-case time for look-ups in the DFS stack can be improved from linear to logarithmic or constant in n. If all reachable states up to depth $\mathsf{O}(n^c)$ for nonlocal properties are explored, full coverage holds even if the bug finder at hand thinks that more reachable states might still exist, but does not know it for sure due to, for example, not having the ability or enough space to store already visited states. Similarly, if all reachable states up to some constant depth independent of n are explored, full coverage for local non-reachability properties is achieved, making bounded verification complete. These search-depths bounds for non-local and local properties are asymptotically negligible relative to the number of states. Moreover, if a good upper bound on diamax (or C for local properties) is ever provided, the modulus of the difference between this upper bound and the (e.g., default worst-case) search depth of a bug finder might be used to better estimate the quality of the bug finder. This would help to coarsely rank the bug finders.

Low complexity of (non-)reachability has some asymptotic ramifications in terms of classical complexity theory (again, n being the only variable). First, reachability queries are probably efficiently parallelizable in n due to $\mathsf{NC}^1 \subseteq \mathsf{NSpace}(\log(\text{input size})) \subseteq \mathsf{NC}^2$. Second, finding bugs in even huge input programs is probably tractable due to using only $\mathsf{O}(\log^2 n)$ additional memory cells (because of $\mathsf{NC}^1 \subseteq \mathsf{NSpace}(\log n) \subseteq \mathsf{DSpace}(\log^2 n)$).

From an algorithmic viewpoint, our proofs of the complexity bounds demonstrate that symmetry reduction in tools is more than just a heuristic: when suitably performed, it diminishes the resource consumption also in the worst case.

To the practical tool builder, on the one hand, our results show that implementations could consider paying a little bit more attention to avoiding the state-space explosion in n in the worst case (since the exponential blow-up in n is theoretically avoidable in the setting described). On the other hand, large or unknown constants in our upper bounds emphasize a well-known warning: theoretical advances primarily say only that certain improvements could probably be achieved asymptotically in principle, not that any advance translates into practice directly.

Concerning research on verification, our results show that, in certain natural mathematical setups, the exponential state-explosion phenomenon in the number of threads has reduced or no bearing on the reachability analysis for parallel programs. Further, if a modeling step bounds the sizes of the shared and thread-local state spaces in the abstraction when they are unbounded in the concrete, unexpected mathematical artifacts may emerge. One such artifact is the state-explosion problem turning out to be asymptotically a nonproblem, informally speaking. Depending on the application, this artifact might be considered helpful (e.g., if small values of the fixed parameters turn out to allow for specialized but fast verifiers, for instance, using caching) or detrimental (e.g., hidden large constants might fool the developers). Also, the widely stated claim that the exponential blow-up of the state space in the number of threads is a major obstacle to verification should not be *blindly* interpreted in a mathematically straightforward way. Sometimes this blow-up is an empirical phenomenon occurring in the tools that do not perform symmetry reduction properly or at all, and sometimes the set of programs on which the state explosion is observed simply does not have constant local and shared state spaces.

As a next theoretical-research step, we plan to tighten the aforementioned inequalities. While we do not know whether diamax can be majorized, for instance, by a linear function, we expect that the upper bound could be lowered in principle; one way could start with determining a possibly large class of threads that do not occur in diamax-realizing programs and subtracting this class while defining E in Definition V.2.1.10 and another way could be using distance-preserving bisimulations to reduce the index of $\approx$. An interesting research direction is searching for an explicit upper bound on the maximal local diameter C (perhaps, using [75]). Locating the general and local reachability problems in the hierarchy of parametric complexity classes is a further, orthogonal line of research. Finally, a mathematical study of the state-space explosion problem in the number of parallel components is important also for other models of parallel computation.

Acknowledgments

This work would have been impossible without prior research contributions of Steffen Borgwardt from the University of Colorado Denver, USA. Funding: This work was supported by the project "Verbundprojekt SpesML: SysML Workbench für die SPES Methodik" at Software and Systems Engineering Research Group, led by Manfred Broy at TUM, Germany. The funding source had no involvement in the conduct of the research, preparation, or submission of the article.

References

[0] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. "Subcubic Equivalences Between Graph Centrality Problems, APSP and Diameter." In: *SODA*. 2015, pp. 1681–1697.

[1] Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. "General Decidability Theorems for Infinite-State Systems." In: *LICS*. 1996, pp. 313–321.

[2] Parosh Aziz Abdulla, Frédéric Haziza, and Lukáš Holík. "All for the price of few." In: *VMCAI*. Springer. 2013, pp. 476–495.

[3] Noga Alon, Zvi Galil, and Oded Margalit. "On the Exponent of the All Pairs Shortest Path Problem." In: *J. Comput. Syst. Sci.* (1997).

[4] Rajeev Alur, Sampath Kannan, and Mihalis Yannakakis. "Communicating Hierarchical State Machines." In: *ICALP'99*. 1999, pp. 169–178.

[5] Patrick L. Anderson and Ilhan Kubilay Geckil. *Northeast blackout likely to reduce US earnings by $6.4 billion*. Tech. rep. http://www.andersoneconomicgroup.com/Portals/0/upload/Doc544.pdf. Aug. 2003.

[6] James Aspnes, Faith Ellen Fich, and Eric Ruppert. "Relationships between broadcast and shared memory in reliable anonymous distributed systems." In: *Distributed Computing* 18.3 (Feb. 2006), pp. 209–219.

[7] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[8] Henk Pieter Barendregt and Erik Barendsen. *Introduction to λ-calculus*. Tech. rep. Department of Computer Science, Radboud University of Nijmegen, 2000.

[9] Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. "Context-aware counter abstraction." In: *FMSD* (2010).

[10] Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. "Symbolic counter abstraction for concurrent software." In: *CAV*. 2009.

[11] Ahmed Bouajjani. "Lecture 1: Verification of concurrent programs. Part 1: decidability and complexity results" (VTSA). Talk. MPII, Saarbrücken, Sept. 6, 2012. URL: `http://resources.mpi-inf.mpg.de/departments/rg1/conferences/vtsa12/slides/bouajjani/Bouajjani_L1_VTSA12.pdf#page=6` (visited on 10/25/2019).

[12] Nicolas Bourbaki. *Algèbre commutative, chapitres 1 à 4*. Springer, 1985. ISBN: 978-3-540-33937-3.

[13] Michael C. Browne, Edmund Melson Clarke, and Orna Grümberg. "Reasoning about networks with many identical finite-state processes." In: *Inf. Comput.* (1989).

[14] Manfred Broy. *Logische und methodische Grundlagen der Programm- und Systementwicklung. Datenstrukturen, funktionale, sequenzielle und objektorientierte Programmierung*. In collab. with Alexander Malkis. Springer, 2019. ISBN: 978-3-658-26301-0.

[15] Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. "On the complexity of bounded context switching." In: vol. 87. LIPIcs. 2017.

[16] Peter Chini, Roland Meyer, and Prakash Saivasan. "Fine-grained complexity of safety verification." In: *TACAS*. Springer, 2018, pp. 20–37.

[17] Fan Chung. "Diameters of communication networks." In: *Proc. Sympos. Appl. Math. 34 Amer. Math. Soc.*. 1986, pp. 1–18.

[18] Fan Chung. "The diameter and Laplacian eigenvalues of directed graphs." In: *Electr. J. Comb.* 13.1 (2006).

[19] Edmund Melson Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Dec. 1999. ISBN: 978-0-262-03270-4.

[20] Edmund Melson Clarke and Orna Grümberg. "Avoiding the state explosion problem in temporal logic model checking." In: *PODC*. 1987.

[21] CWoo. *Homomorphism between partial algebras. Isomorphism*. Planetmath. Mar. 22, 2013. URL: `http://planetmath.org/homomorphismbetweenpartialalgebras#S0.SS0.SSSx1.p5`.

[22] Peter Dankelmann. "The diameter of directed graphs." In: *J. Comb. Theory, Ser. B* 94.1 (2005), pp. 183–186.

[23] George Bernard Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, 1963.

[24] Xiaotie Deng, Horace Ho-Shing Ip, Ken Chee-keung Law, Jianping Li, Weimin Zheng, and Shanfeng Zhu. "Parallel models and job characterization for system scheduling." In: *ICCS*. 2001.

[25] Raymond Devillers, Evgeny Erofeev, and Thomas Hujsa. "Synthesis of weighted marked graphs from constrained labelled transition systems." In: *ATAED@Petri Nets/ACSD*. 2018, pp. 75–90.

[26] Alastair F. Donaldson and Alice Miller. "Automatic symmetry detection for Promela." In: *Journal of Automated Reasoning* 41.3–4 (2008).

[27] Ernest Allen Emerson and Vineet Kahlon. "Reducing model checking of the many to the few." In: *CADE*. 2000, pp. 236–254.

[28] Ernest Allen Emerson and Aravinda Prasad Sistla. "Symmetry and Model Checking." In: *FMSD* 9.1/2 (1996), pp. 105–131.

[29] Javier Esparza. "Keeping a crowd safe: on the complexity of parameterized verification (corrected version)." In: (2014). Successor to a STACS 2014 paper.

[30] Javier Esparza, Pierre Ganty, and Rupak Majumdar. "Parameterized verification of asynchronous shared-memory systems." In: *JACM* 63.1 (2016).

[31] Javier Esparza, Pierre Ganty, and Tomás Poch. "Pattern-based verification for multithreaded programs." In: *TOPLAS* 36.3 (2014).

[32] Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. "Inductive data flow graphs." In: *SIGPLAN Not.* 48.1 (2013).

[33] Cormac Flanagan, Stephen Neal Freund, Shaz Qadeer, and Sanjit Arunkumar Seshia. "Modular verification of multithreaded programs." In: *TCS* 338.1-3 (2005).

[34] Cormac Flanagan and Shaz Qadeer. "Thread-modular model checking." In: *SPIN*. Vol. 2648. LNCS. Springer, 2003, pp. 213–224.

[35] Patrice Godefroid. *Partial-order methods for the verification of concurrent systems – an approach to the state-explosion problem*. LNCS. Springer, 1996.

[36] Google LLC. *Search results for thread identifier*. A subset of results from `http://www.google.com/search?q=%22char%20thread_id;%22+OR+%22char%20threadId;%22`. 2020.

[37] George A. Grätzer. *Universal Algebra*. Second edition with updates. Springer, 2008. ISBN: 978-0-387-77486-2.

[38] Thomas Anton Henzinger, Ranjit Jhala, and Rupak Majumdar. "Race checking by context inference." In: *PLDI*. 2004, pp. 1–13.

[39] Graham Higman. "Ordering by divisibility in abstract algebras." In: *Proc. London Math. Soc. (3)* 2 (1952), pp. 326–336.

[40] Gerard Johan Holzmann. "The model checker SPIN." In: *IEEE TSE* 23.5 (May 1997), pp. 279–295.

[41] Gerard Johan Holzmann. *The SPIN model checker: primer and reference manual*. Addison-Wesley, 2003. ISBN: 0-321-22862-6.

[42] John Edward Hopcroft, Rajeev Motwani, and Jeffrey David Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2001.

[43] Qiang-Sheng Hua, Haoqiang Fan, Ming Ai, Lixiang Qian, Yangyang Li, Xuanhua Shi, and Hai Jin. "Nearly optimal distributed algorithm for computing betweenness centrality." In: *ICDCS*. 2016, pp. 271–280.

[44] Neil Immerman. "Nondeterministic space is closed under complementation." In: *SIAM J. Comput.* 17.5 (1988), pp. 935–938.

[45] Insomnia Security. *Concurrency Vulnerabilities*. July 11, 2011. URL: `http://owasp.org/www-pdf-archive/OWASP_NZDay_2011_BrettMoore_ConcurrencyVulnerabilities.pdf` (visited on 05/31/2020).

[46] Neil Deaton Jones. *Computability and Complexity*. 2006.

[47] Alexander Kaiser, Daniel Kroening, and Thomas Wahl. "Dynamic cutoff detection in parameterized concurrent programs." In: *CAV*. 2010, pp. 645–659.

[48] Richard Manning Karp and Raymond Edward Miller. "Parallel program schemata." In: *JCSS* 3.2 (1969), pp. 147–195. ISSN: 0022-0000.

[49] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors." In: *ISCA'14*. 2014.

[50] Donald Ervin Knuth. "Big omicron and big omega and big theta." In: *SIGACT News* 8.2 (Apr. 1976).

[51] Dexter Kozen. "Lower bounds for natural proof systems." In: *FOCS*. IEEE, 1977, pp. 254–266.

[52] Salvatore La Torre, Parthasarthy Madhusudan, and Gennaro Parlato. "Model checking parameterized concurrent programs using linear interfaces." In: *CAV*. 2010.

[53] Shuvendu K. Lahiri, Alexander Malkis, and Shaz Qadeer. "Abstract Threads." In: *VMCAI*. 2010, pp. 231–246.

[54] Eric Lehman, Frank Thomson Leighton, and Albert Ronald da Silva Meyer. *Mathematics for computer science*. June 6, 2018. ISBN: 978-988-8407-06-4.

[55] Jérôme Leroux. "Presburger Vector Addition Systems." In: *LICS*. 2013.

[56] Nancy Gail Leveson and Clark Savage Turner. "Investigation of the Therac-25 accidents." In: *IEEE Computer* 26.7 (1993), pp. 18–41.

[57] Ming Li and Paul M. B. Vitányi. "Kolmogorov complexity and its applications." In: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Ed. by Jan van Leeuwen. 1990, pp. 187–254.

[58] Boris Dmitrievich Lubachevsky. "An approach to automating the verification of compact parallel coordination programs. I." In: *Acta Inf.* (1984).

[59] Alexander Malkis. "Cartesian Abstraction and Verification of Multithreaded Programs." PhD thesis. Albert-Ludwigs-Universität Freiburg, 2010.

[60] Alexander Malkis. "Reachability in Multithreaded Programs Is Polynomial in the Number of Threads." In: *PDCAT*. Version with proofs available from `http://wwwbroy.in.tum.de/~malkis/Malkis-ReachabilityInMultithreadedProgramsIsPolynomialInTheNumberOfThreads_techrep.pdf` or `http://wwwbroy.in.tum.de/~malkis/Malkis-ReachabilityInMultithreadedProgramsIsPolynomialInTheNumberOfThreads_techrep.ps`. 2019.

[61] Alexander Malkis. "Reachability in Parallel Programs Is Polynomial in the Number of Threads." In: *JPDC* (2022). In press.

[62] Alexander Malkis. *Sequence A290642 at OEIS*. Aug. 9, 2017. URL: `http://oeis.org/A290642`.

[63] Alexander Malkis and Steffen Borgwardt. "Binary Multithreaded Programs Have Low Reachability Complexity and Small Diameters." A full version of [64], submitted to JCSS. 2017.

[64] Alexander Malkis and Steffen Borgwardt. "Reachability in Binary Multithreaded Programs Is Polynomial." In: *ICDCS*. IEEE, June 2017. DOI: `10.1109/ICDCS.2017.290`.

[65] Alexander Malkis and Andreas Podelski. *Refinement With Exceptions*. Research rep. University of Freiburg, 2008. URL: `http://www.sec.in.tum.de/~malkis/MalkisPodelski-refinementWithExceptions_techrep.pdf`.

[66] Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko. *Thread-modular verification and Cartesian abstraction*. Research rep. Updated version. 2006. URL: `http://wwwbroy.in.tum.de/~malkis/MalkisPodelskiRybalchenko-threadModVerAndCartAbs.pdf`.

[67] Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko. "Thread-modular verification is Cartesian abstract interpretation." In: *ICTAC*. Ed. by Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone. Vol. 4281. Lecture Notes in Computer Science. Springer, 2006, pp. 183–197. ISBN: 3-540-48815-4.

[68] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems. Safety*. Springer, 1995. ISBN: 978-0-387-94459-3.

[69] Ernst Wilhelm Mayr and Albert Ronald da Silva Meyer. "The complexity of the finite containment problem for Petri Nets." In: *JACM* 28.3 (1981), pp. 561–576.

[70] Scott Owens, Susmit Sarkar, and Peter Sewell. *A Better x86 Memory Model: x86-TSO (Extended Version)*. Tech. rep. UCAM-CL-TR-745. Univ. of Cambridge, 2009.

[71] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000. ISBN: 978-0-89871-464-7.

[72] Amir Pnueli, Jessie Xu, and Lenore D. Zuck. "Liveness with $(0, 1, \infty)$-Counter Abstraction." In: *CAV*. 2002, pp. 107–122.

[73] Kevin Poulsen. *Software bug contributed to blackout*. Originally available at `http://www.securityfocus.com/news/8016`. Feb. 12, 2004. URL: `http://www.theregister.com/2004/02/12/software_bug_contributed_to_blackout`.

[74] Kevin Poulsen. *Tracking the blackout bug*. Originally available at `http://www.securityfocus.com/news/8412`. Apr. 8, 2004. URL: `http://www.theregister.com/2004/04/08/blackout_bug_report`.

[75] Charles Weill Rackoff. "The covering and boundedness problems for vector addition systems." In: *TCS* (1978).

[76] Petr Ročkai. "Model checking software." PhD thesis. Masaryk university, Jan. 2015.

[77] Walter John Savitch. "Relationships between nondeterministic and deterministic tape complexities." In: *J. Comput. Syst. Sci.* 4.2 (1970).

[78] Sylvain Schmitz. "Complexity hierarchies beyond elementary." In: *TOCT* 8.1 (2016), 3:1–3:36.

[79] Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen. "x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors." In: *CACM* 53.7 (2010), pp. 89–97.

[80] Stephen Frederick Siegel and George Sam'l Avrunin. "Improving the precision of INCA by eliminating solutions with spurious cycles." In: *IEEE TSE* (2002).

[81] Michael Sipser. *Introduction to the theory of computation*. 3rd ed. Cengage learning, 2013.

[82] Aravinda Prasad Sistla and Edmund Melson Clarke. "The complexity of propositional linear temporal logics." In: *JACM* 32.3 (1985).

[83] Aravinda Prasad Sistla and Steven M. German. "Reasoning about systems with many processes." In: *JACM*. 1992.

[84] Aravinda Prasad Sistla and Steven M. German. "Reasoning with many processes." In: *LICS*. 1987, pp. 138–152.

[85] Herb Sutter. "The free lunch is over: a fundamental turn toward concurrency in software." In: *Dr. Dobb's Journal* 30.3 (Mar. 2005).

[86] Róbert Szelepcsényi. "The method of forced enumeration for nondeterministic automata." In: *Acta Inf.* 26.3 (1988), pp. 279–284.

[87] Bolesław Karol Szymański. "A simple solution to Lamport's concurrent programming problem with linear wait." In: *ICS*. ACM, 1988.

[88] The MITRE corporation. *CWE category: concurrency issues*. May 2017. URL: `http://cwe.mitre.org/data/definitions/557.html`.

[89] tombom. *Interrupting query sends wrong thread ID*. Sept. 21, 2017. URL: `http://dba.stackexchange.com/questions/186545`.

[90] Douglas Brent West. *Introduction to graph theory*. 2nd ed. Prentice Hall, 2001. ISBN: 0-13-014400-2.

[91] Wikipedia. *List of PSpace-complete problems*. June 2019. URL: `http://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems`.

[92] Wikipedia. *Peterson's algorithm*. 2019. URL: `http://en.wikipedia.org/wiki/Peterson's_algorithm`.

[93] Haoxian Zhao. "Using the Karp-Miller tree construction to analyse concurrent finite-state programs." MA thesis. Trinity College, Oxford, UK, 2009.