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

Alexander Malkis
Technical University of Munich (TUM), Boltzmannstr. 3, 85748 Garching, Germany

***Abstract*—Reachability in multithreaded programs is an important yet inherently difficult problem, even if they are finite-state and equipped with the interleaving semantics. So far, the complexity of this problem as a function of the number of threads n, while keeping the maximal size of the thread-local memory and the size of shared memory bounded by a constant, has been explored poorly. We close this gap by measuring aspects such as (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 distance from any program state to any thread-local state, and (iii) the computational complexity of bug finding. We prove that all these are majorized by a polynomial in n and, in certain cases, by a linear, logarithmic, or even constant function in n. Such bounds shed more light on how the widely expressed claim, that one of the major obstacles to analyzing concurrent programs is the exponential state explosion in the number of threads, should (and should not) be understood.**

***Index Terms*—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**

0. Introduction

Since 2004, the CPU-clock limit has been stagnating, which has increased the demand for multithreading by a quantum leap [0]. 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 [1, 2]; 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 [3, 4], and the effects of failures can be devastating [5–8]. 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 [9]. Deciding such properties of programs in practice often incurs the infamous state-explosion problem, which is the phenomenon whereby the number of program states is exponential in the number of threads (hereinafter n) of the program analyzed [10–15]. It is a practical "problem" because the program analyzer often runs out of resources while reasoning about these states and fails to deliver a conclusive answer. It is possible to show that deciding reachability for finite-state multithreaded programs equipped with interleaving semantics is PSpace-complete in the overall input length (cf. generally Lem. 3.2.3 in [16]). Since PSpace is a very robust class [17, § 8.2, Ex. 8.4] containing a wealth of complete decision problems [18, 19], the PSpace-completeness characterization is rather coarse. It reveals no details about the exponential blow-up with respect to n from a theoretical viewpoint.

To study the state-explosion phenomenon, we consider a parametrized setting with a variable number of threads n and two constant parameters: the maximal size of 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 formally later) and the complexity of two natural reachability problems. Informally speaking, is the growth fast (as suggested by the state explosion occurring in the tools) or slow (not atypical 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. These results generalize and partially sharpen the previous findings of [20] (this discusses *binary* programs, in which the two aforementioned constant parameters are both 2) and are motivated by experiments [21].

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, a leading Web-search

engine 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 modeling memory limitations in dynamic systems. (A typical bug example is a thread-identifier overflow [22]: a server starts a new thread upon a new query from a client while using fixed space for thread identifiers. After the server runs sufficiently long, 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 modeling Edge Computing, in which parallel computations (e.g., in the cloud) are given to a variable number of small-size computational nodes (say, embedded and mobile devices) which are accessing 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 semantics.

Before proceeding, 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 diameter of the program. 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.

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). Thus, we concentrate on the function that, given a natural number n, returns the largest diameter over all programs with n threads. We will call this function diamax. To the best of our knowledge, nothing is known about this function yet except [20]. Certainly, diamax is majorized by the size of the state space, which is singly exponential in n. We show much more: a linear lower bound and a polynomial upper bound, thereby positively solving Open Problem 1.12.3 from [23]. Our upper bound is both a generalization and a tightening of the upper bound from [20]. Furthermore, 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 will also show that the program-state–to–program-state (non-)reachability problem belongs to the complexity class $\mathsf{NSpace}(\log n)$.

The above notion of the (maximal) diameter is based on the program-state–to–program-state distance and thus targets "nonlocal" properties concerning more than one thread, such as deadlock freedom or mutual exclusion. Still, many interesting nonreachability properties (of, say, real operating-system code) are "local," meaning that they are, to simplify, of the form "a particular state of a particular thread does not occur in any execution" (a state of a thread, shortly, a *thread state*, is 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. 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. 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 this 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). (A formal definition will appear in § I.2.) 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. These lower and upper bounds also apply to searches for local bugs in program models (and not only in original programs), e.g., in the inner loops of the CEGAR schemes.

We will 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. (That is, we can actually write, to any level of detail, an algorithm computing the least upper bound.) One may derive the existence of the bound through a careful interpretation and extension of results in the literature (cf. § VI); we propose a mostly self-contained proof of the existence of the bound and its explicit construction in § III. Moreover, we will show that the program-state–to–thread-state (non-)reachability problem belongs to the complexity class NC^1 [17, Def. 10.38] (again, considering the shared and thread-local memories bounded). The membership in this class is both a generalization and an improvement over the corresponding bound from [20]. The results on diameters and local diameters are, 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 [20]).

- Constructively bounding the maximum local diameter for the parametrized case from above by an explicitly computable value independent of n (Def. and Cor. III.9).
- Bounding diamax between a linear and a polynomial function (Thms. IV.1.1 and IV.2.1.13). Restricted to the binary case, the polynomial degree is lower than a previously known one (Note IV.2.1.14).
- A class of programs for which we show a linear upper bound in n on the diameter (Thm. IV.2.2.3). This upper bound matches the lower bound up to a constant.
- For rather general probability distributions on thread transitions, the diameter of a program is asymptotically almost surely at most linear in n (Thm. IV.2.3.2).
- Deciding whether a program state is reachable from another program state is possible in $\mathsf{NSpace}(\log n)$ (Thm. V.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 (Thm. V.2).

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

Limitations

First, it is not the goal of this paper to empirically measure or improve contemporary techniques for bug finding or verification; none of the algorithms from the proofs are meant to be used directly in practice. 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 would break the limits of this paper and is relegated to a later report [24].) 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, track or even optimize them if the corresponding proof methods permit to do it easily. Whether these constants are "optimal enough" is subjective and does not matter much for the purpose of classification: 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., [25]). 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 [26, §§ 7, 8] or be anywhere between linear [27, Ex. 13] and, say, $n^{\mathsf{O}(n)}$ [28]). Moreover, some parametrized programs (say, Readers-Writers) come with two or more independent variable parameters which together determine the number of threads 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 another. As opposed to that, this paper aims to deal with only one variable parameter and generically with the whole class of multithreaded programs rather than particular families of multithreaded programs.

I. Preliminaries

We now introduce the formal notation used throughout the paper.

I.1. General conventions

In logical statements, the symbol $\Rightarrow$ means implication and $\Leftrightarrow$ means bi-implication.

Let $\mathbb{N}_+$ be the set of positive integers, $\mathbb{N}_{\geqslant 0}$ the set of nonnegative integers, $\mathbb{Q}$ the set of rationals, and $\mathbb{Q}_{\geqslant 0}$ 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 $\forall\, i, j\colon i{<}j \Leftrightarrow i{\in}j$. (It will help us to get rid of additional notation for the set of thread identifiers such as Tid often seen in the literature [29]: 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 "$\forall\, i \in n \setminus \{j\}\colon \ldots$" are simpler than "$\forall\, i \in \mathbb{N}_{\geqslant 0}\colon (i < n \wedge i \neq j) \Rightarrow \ldots$.")

Our map-constructor is right-associative, meaning that $X \to Y \to Z$ 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 [30]; 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 $X \leftrightarrowtail Y$ 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}. The domain and the image of a map f are denoted by $\mathsf{dom}\, f$ and $\mathsf{img}\, f$, respectively. For finite functions mapping into some number domain (naturals, rationals, ...), $\|\cdot\|_1$ denotes the 1-norm and $\|\cdot\|_\infty$ the maximum norm: $\|f\|_1 \stackrel{\text{def}}{=} \sum_{i \in \mathsf{dom}\, f} |f(i)|$ and $\|f\|_\infty \stackrel{\text{def}}{=} \max\{|f(i)| \mid i \in \mathsf{dom}\, f\}$, where $\max \emptyset \stackrel{\text{def}}{=} 0$.

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 ith element as σ_i or $\sigma^{[i]}$; the whole sequence is written as $\big(\sigma_i\big)_{i \in I}$ or $\big(\sigma^{[i]}\big)_{i \in I}$, respectively. If the index set is some initial segment of natural numbers, we sometimes write $i{<}n$ or $i{\leqslant}n$ instead of $i{\in}n$ or $i \in n{+}1$ in the right subscript position; the version with the weak inequality "$\leqslant$" additionally implies the nonemptiness of σ. The context determines which notation is most convenient.

By a slight abuse of notation, a plain number in the right upper 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. 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.

The *index of an equivalence relation* is the number of the equivalence classes.

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

I.2. Program notation

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

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

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 [31].) 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

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

The *transition graph* induced by p is a directed graph $(\mathsf{State}, \longrightarrow)$ where

$$(g,l) \longrightarrow (g',l') \;\stackrel{\text{def}}{\Leftrightarrow}\; \exists\, i{<}n\colon \begin{pmatrix} (g,l_i) \to_i (g',l'_i) \\ \wedge\; \forall\, j \in n\backslash\{i\}\colon l_j{=}l'_j \end{pmatrix}$$

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 $\forall\, i,j \in m\colon s_i{=}s_j \Rightarrow i{=}j$. The *length* of a nonempty walk is the number of times the walks takes an edge:

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

where "$\leqslant$" in the subscript ensures 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):

$$\mathsf{d}(s,s') \;\stackrel{\text{def}}{=}\; \min\left\{ m \;\middle|\; \begin{array}{l} \exists\,\text{walk } (s_0,\ldots,s_m) \text{ in } (\mathsf{State},\longrightarrow) \\ \text{such that } s_0{=}s \wedge s_m{=}s' \end{array} \right\},$$

where $\min\emptyset \stackrel{\text{def}}{=} \infty$. 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., $\mathsf{d}^{\mathsf{loc}}_p(\ldots)$ means the local distance inside program p.

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):

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

where again $\min\emptyset \stackrel{\text{def}}{=} \infty$. The *diameter* of a transition graph as above is the largest realizable finite distance:

$$\mathsf{diam}(\mathsf{State},\longrightarrow) \;\stackrel{\text{def}}{=}\; \max\big((\mathsf{img}\,\mathsf{d}) \setminus \{\infty\}\big)\,.$$

The *diameter* of a program p is the diameter of its transition graph:

$$\mathsf{diam}(p) \;\stackrel{\text{def}}{=}\; \mathsf{diam}(\text{transition graph of } p)\,.$$

Since we are interested in the dependency of the measured quantities on n, we fix Glob, Loc, $G \stackrel{\text{def}}{=} |\mathsf{Glob}|$, and $L \stackrel{\text{def}}{=} |\mathsf{Loc}|$ for the rest of the paper. Thus, we write programs such as p above more shortly as

$$p = (\to_0, \ldots, \to_{n-1})$$

or, even more compactly,

$$p \;=\; (\to_i)_{i<n}\,.$$

We say that a program $(\rightsquigarrow_i)_{i<m}$ is a *subprogram* of a program $(\to_i)_{i<n}$ if there is an injective map $f\colon m{\hookrightarrow}n$ such that $\forall\, i{<}m\colon \rightsquigarrow_i \;=\; \to_{f(i)}$. Note that the subprogram relation is a preorder on the set of programs.

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

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

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

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

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

$$\mathsf{diamax}^{\mathsf{loc}}(n) \;\stackrel{\text{def}}{=}\; \max\left\{ \mathsf{diam}^{\mathsf{loc}}(p) \;\middle|\; \begin{array}{l} p \text{ is an } n\text{-threaded} \\ \text{program} \end{array} \right\}.$$

II. Examples

Before turning to the main results of this paper, we present some small examples (mostly taken from [20]). For simplicity, we show a few binary n-threaded programs whose diameter is $\mathsf{diamax}(n)$ for $n \in \{1,2,3\}$.

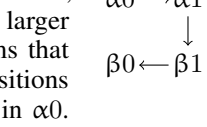
We therefore give some names to the shared and local states, say, $\mathsf{Glob} = \{\alpha,\beta\}$ and $\mathsf{Loc} = \{0,1\}$, where $\alpha{\neq}\beta$ are some literals. For brevity in the binary case, we will sometimes omit commas and parens when writing thread or program states: we will occasionally 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.

II.1. $n = 1$

Consider the program with exactly one thread and transitions $\alpha0 \to_0 \alpha1$, $\alpha1 \to_0 \beta1$, and $\beta1 \to_0 \beta0$. The diameter and the local diameter of this program are both 3. The transition graph of the program is depicted below right.

Since the total number of program states is 4, no single-threaded program can have a larger diameter. In total, there are six programs that have diameter 3 and exactly three transitions and whose diameter-realizing paths start in $\alpha0$.

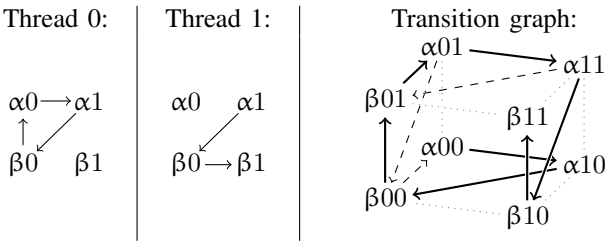


II.2. $n = 2$

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 $\alpha 00$ 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.



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: $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1) = \mathsf{diam}(\rightarrow_0, \rightarrow_1, \rightarrow_1) = 8$ and $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1, \rightarrow_1) = 9$, which, as we will see, are less than the lower bound from Thm. IV.1.1. 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 $\alpha 1$ of thread 1 occurs in $\alpha 01$ and $\alpha 11$). 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. Figs. 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 $\alpha 0 \rightarrow_0 \beta 0$ shrinks the diameter. Instead, adding any combination of the thread transitions $\alpha 1 \rightarrow_0 \alpha 0$, $\beta 0 \rightarrow_0 \alpha 0$, and $\beta 1 \rightarrow_0 \alpha 0$ does not change the diameter. We have $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1) = 7$ and $\mathsf{diam}(\rightarrow_0, \rightarrow_1, \rightarrow_1) = 8 = \mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1, \rightarrow_1)$. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 1, \beta 1)$.

The transition graph of N2T6B is strongly connected. Adding any combination of the thread transitions $\beta 1 \rightarrow_0 \alpha 1$, $\alpha 1 \rightarrow_0 \alpha 0$, $\beta 1 \rightarrow_0 \alpha 0$, and $\beta 0 \rightarrow_0 \alpha 0$ would not change the diameter, but adding $\alpha 1 \rightarrow_0 \beta 0$ would shrink it. Duplicating the threads slightly increases the diameter: $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1) = \mathsf{diam}(\rightarrow_0, \rightarrow_1, \rightarrow_1) = 8$ and $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1, \rightarrow_1) = 9$. The local diameter is $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 00, 1, \beta 0)$.

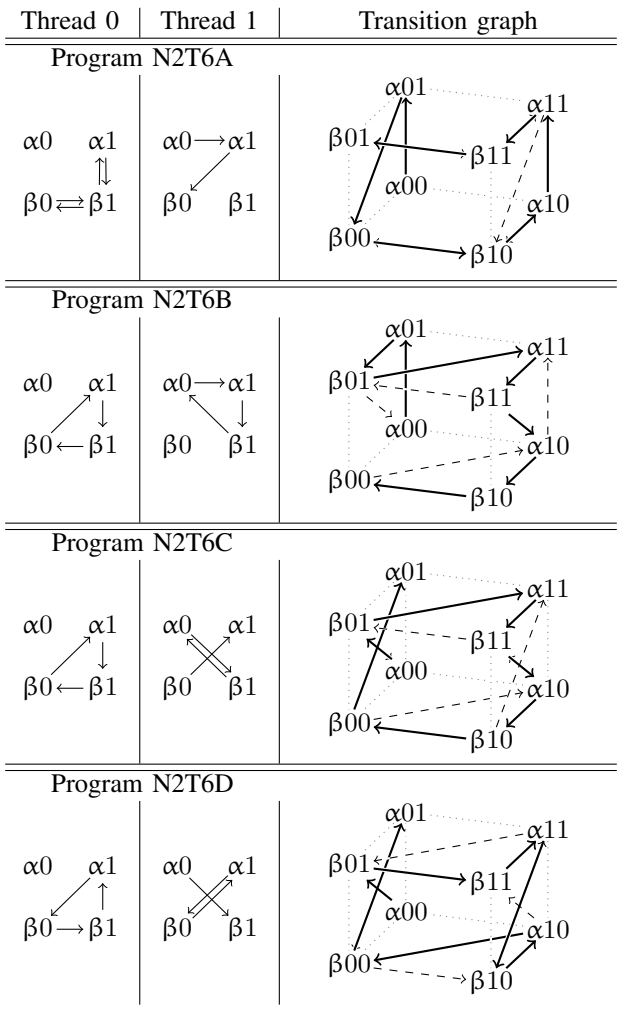


Figure 1. Some two-threaded binary programs with 6 thread transitions.

In N2T6C, adding any combination of the transitions $\alpha 1 \rightarrow_0 \alpha 0$, $\beta 0 \rightarrow_0 \alpha 0$, and $\beta 1 \rightarrow_0 \alpha 0$ would shrink the diameter. Adding the thread transition $\beta 1 \rightarrow_0 \alpha 1$ would not change the diameter. We have $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1) = 7$, $\mathsf{diam}(\rightarrow_0, \rightarrow_1, \rightarrow_1) = 8$, and $\mathsf{diam}(\rightarrow_0, \rightarrow_0, \rightarrow_1, \rightarrow_1) = 9$. 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 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 $6 = \mathsf{d}^{\mathsf{loc}}(\alpha 01, 1, \beta 0)$.

II.3. $n = 3$

Consider the program N3T9 from Fig. 3.

Its transition graph is depicted in Fig. 4.

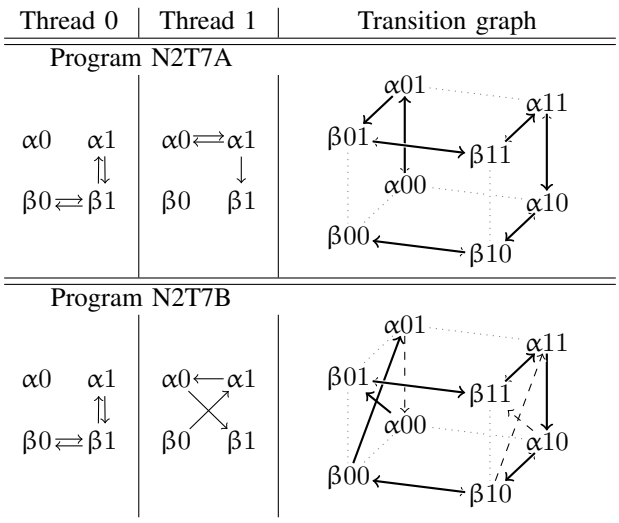


Figure 2. Some two-threaded binary programs with 7 transitions.

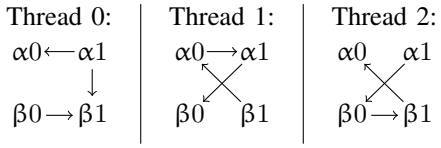


Figure 3. A three-threaded binary program.

This program has local diameter $9 = \mathsf{d}^{\mathsf{loc}}(\alpha000, 2, \alpha1)$ and diameter $13 = \mathsf{d}(\alpha000, \beta011)$. The transition graph is strongly connected. Adding a copy of each thread would shrink the diameter: $\mathsf{diam}(\to_0, \to_0, \to_1, \to_1, \to_2, \to_2) = 11$. It can be shown [20, Prop. 6.2.1.1] that N3T9 has the maximal diameter among all three-threaded binary programs.

III. 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. From a theoretic viewpoint, a constant bound will be far better than the trivial bound $\mathsf{diamax}^{\mathsf{loc}}(n) \leqslant \mathsf{diamax}(n)$ for all $n \in \mathbb{N}_+$.

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.

To develop this idea into a proof, we start with well-known terminology.

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

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

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

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

- *order-homomorphism* iff $\forall\, x_1, x_2 \in X\colon x_1 \lesssim_X x_2 \Leftrightarrow \varphi(x_1) \lesssim_Y \varphi(x_2)$, and
- *order-epimorphism* iff φ is a surjective order-homomorphism.

The following popular result is essential for the whole section:

Proposition III.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.*

Proof idea. A special case of Hilbert's basis theorem (cf. Chapitre III, § 2.10, Corollaire 2 in [32]). ■

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

Note that for each $m \in \mathbb{N}_{\geqslant 0}$, the componentwise partial order on $\mathbb{N}^m_{\geqslant 0}$ is well-founded. We combine this fact with Prop. III.1 now:

Lemma III.2. *Let $\mathbb{N}_{\geqslant 0}$ be equipped with the standard order on natural numbers. Let $m \in \mathbb{N}_{\geqslant 0}$, the set $\mathbb{N}^m_{\geqslant 0}$ be equipped with componentwise partial order, and $f\colon \mathbb{N}^m_{\geqslant 0} \dashrightarrow \mathbb{N}_{\geqslant 0}$ be an antitone partial map. Then:*

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

b) Assume that explicit algorithms solving the following problems exist:

1) Decide, given an arbitrary set $I \subseteq m$ and a sequence of pairs $(s_i, a_i)_{i\in I} \in (\{\text{`='}, \text{`}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$, whether some $y \in \mathsf{dom}\, f$ exists satisfying $\bigwedge_{i\in I} \big((s_i = \text{`='} \wedge y_i = a_i) \vee (s_i = \text{`}\geqslant\text{'} \wedge y_i \geqslant a_i)\big)$.

2) Evaluate f at a point of its domain.

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

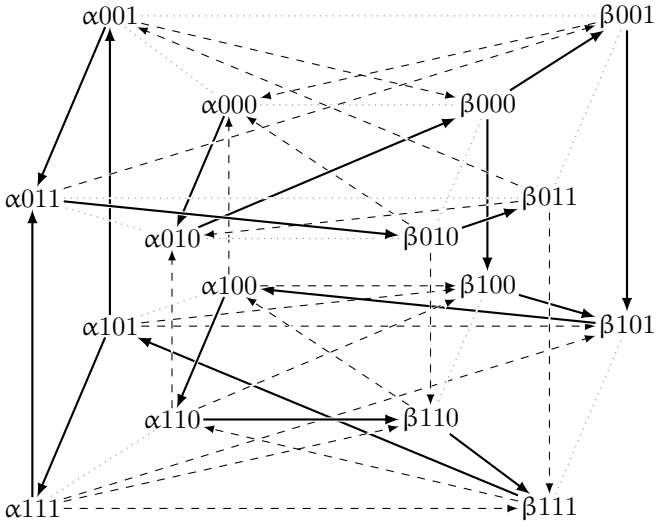


Figure 4. N3T9: transition graph and a path realizing the longest distance.

Proof idea. In the interesting case that $\mathsf{dom}\, f \neq \emptyset$, consider the antichain of minimal elements of $\mathsf{dom}\, f$ and let M be the maximal value of f on this antichain. All the values of f are bounded by M from above; there are finitely many such values. As for computability, construct any antichain in $\mathsf{dom}\, f$ first, and then enumerate the tuples below that antichain. ■

In the following, let $\mathcal{P}$ be the set of all programs, equipped with the subprogram preorder (cf. Lem. B.1).

Lemma III.3. *There is some $k \in \mathbb{N}_+$ and some order-epimorphism $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}_{\geqslant 0}^k \setminus \{k \times \{0\}\}$, where the codomain is equipped with the componentwise partial order.*

(Since k is an ordinal, the notation $k \times \{a\}$ means, set-theoretically, simply $\{0, \ldots, k-1\} \times \{a\} = \{(0,a), \ldots, (k-1,a)\} = \underbrace{(a, \ldots, a)}_{k \text{ times}}$, 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 idea. Let $k = \left|\mathfrak{P}((\mathsf{Glob} \times \mathsf{Loc})^2)\right|$ be the number of different thread transition relations. We choose some enumeration $t_0, \ldots, t_{k-1}$ of these relations. Let φ map a program $(\to_i)_{i<n}$ to $\Big(\big|\{i<n \mid \to_i = t_r\}\big|\Big)_{r<k}$. ■

Moreover, all order-epimorphisms as in Lem. III.3 are of the same form:

Lemma III.4. *Let $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}_{\geqslant 0}^k \setminus \{k \times \{0\}\}$ be an order-epimorphism, where the codomain is equipped with the componentwise partial order. Then $k = 2^{G^2 L^2}$, and there is some enumeration $t_0, \ldots, t_{k-1}$ of thread transition relations 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 $(\to_i)_{i<n}$. Moreover, φ is computable, and the preimage of each vector under φ is finite and computable.*

Proof idea. Induction on the number of threads. ■

The above preparations imply:

Theorem III.5. *Let $f\colon \mathcal{P} \dashrightarrow \mathbb{N}_{\geqslant 0}$ 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 an arbitrary set $I \subseteq \mathfrak{P}\big((\mathsf{Glob} \times \mathsf{Loc})^2\big)$ and an I-indexed sequence of pairs $(s_\rightsquigarrow, a_\rightsquigarrow)_{\rightsquigarrow \in I} \in (\{\text{`}=\text{'}, \text{`}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$, whether some $(\to_i)_{i<n} \in \mathsf{dom}\, f$ exists satisfying $\bigwedge_{\rightsquigarrow \in I} \Big(\big(s_\rightsquigarrow = \text{`}=\text{'} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| = a_\rightsquigarrow\big) \vee \big(s_\rightsquigarrow = \text{`}\geqslant\text{'} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| \geqslant a_\rightsquigarrow\big)\Big)$.*
- *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 idea. Consider φ from Lems. III.3 and III.4. Let $A = \{\varphi(p) \mid p \in \mathsf{dom}\, f\}$. The map $g\colon A \to \mathbb{N}_{\geqslant 0}$, $a \mapsto f(p)$ for any $p \in (\mathsf{dom}\, f) \cap \varphi^{-1}(a)$ is well defined. Its image is finite due to Lem. III.2a) and is equal to $\mathsf{img}\, f$. Apply Lem. III.2b). ■

In the following, states of a particular form will 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 can use the aforementioned Thm. III.5:

Lemma III.6. *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 idea. Fix arbitrary $g, g' \in \mathsf{Glob}$, $a, a' \in \mathsf{Loc}$, and $\rightsquigarrow \subseteq (\mathsf{Glob} \times \mathsf{Loc})^2$. Define $f\colon \mathcal{P} \dashrightarrow \mathbb{N}_{\geqslant 0}$ with domain $\Big\{(\to_i)_{i<n} \in \mathcal{P} \;\Big|\; \exists\, i<n\colon \to_i = \rightsquigarrow \wedge\, \mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n \times \{a\}), i, (g', a')\big) < \infty\Big\}$ such that f maps a program $(\to_i)_{i<n}$ from that domain to $\min\Big\{\mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n \times \{a\}), i, (g', a')\big) \;\Big|\; i<n \wedge \rightsquigarrow = \to_i\Big\}$. Apply Thm. III.5 to f. ■

Since there are only finitely many shared states, local states, and thread transition relations, a maximal c from Lem. III.6 can be computed:

Corollary III.7. *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 $\mathsf{d}^{\mathsf{loc}}_p\big((g, n \times \{a\}), i, (g', a')\big) < \infty$, then $\mathsf{d}^{\mathsf{loc}}_p\big((g, n \times \{a\}), i, (g', a')\big) \leqslant c$. Moreover, there is an explicit algorithm constructing the smallest such c.*

In Cor. III.7, the source program state is uniform. Generalizing, if an arbitrary initial program state (g, l) with a finite distance to a thread state (g', a') of thread i is given, we rename the local states of all the threads except the i^{th} one to obtain a (different, in general) program with the same local distance from $(g, n \times \{l_i\})$ to (g', a') in thread i, and apply the above result. We obtain:

Theorem III.8. *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.*

Due to Thm. III.8, the maximal local diameter for programs can be determined:

Definition and Corollary III.9. $\mathcal{C} \overset{def}{=\!=} \max(\mathsf{img}\, \mathsf{diamax}^{\mathsf{loc}})$ *exists and is computable by an explicit algorithm.*

The explicit algorithm can be determined by unrolling the proofs to the necessary level of detail; for example, the top level of the algorithm is "compute $\max(\mathsf{img}\, \zeta)$," where ζ is defined in the proof of Cor. III.7 in the appendix.

Although this paper does not aim to determine the numerical value of $\mathcal{C}$, three remarks are to be made. First, "explicit" means that our algorithm computing $\mathcal{C}$ can be easily converted to a runnable implementation in some real-world programming language (although actually obtaining the numerical representation of $\mathcal{C}$ 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 $\mathcal{C}$ is $GL-1$: we obtain it by considering a single-threaded program whose transition relation is $\{((g,l),(g,l{+}1)) \in (\mathbb{N}_{\geqslant 0}\times\mathbb{N}_{\geqslant 0})^2 \mid g{<}G \wedge l{+}1 < L\} \cup \{((g,L{-}1),(g{+}1,0)) \in (\mathbb{N}_{\geqslant 0}\times\mathbb{N}_{\geqslant 0})^2 \mid g{+}1 < G\}$. Third, a further lower bound stems from the example N3T9 in § II.3: from $\mathsf{diam}^{\mathsf{loc}}(\text{N3T9}) = 9$ we obtain that for $G = L = 2$ we have $\mathcal{C} \geqslant 9$.

IV. 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 random program.

IV.1. A lower bound on diamax

Theorem IV.1.1. $\forall n{\in}\mathbb{N}_+\colon \mathsf{diamax}(n) \geqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$.

Proof idea. Without loss of generality, suppose $\mathsf{Glob} = \{0,\ldots,G{-}1\}$ and $\mathsf{Loc} = \{0,\ldots,L{-}1\}$. Fix $n{\geqslant}1$. Consider the following n-threaded program. Let the transitions of thread 0 be $\{((g,l),(g,l{+}1)) \mid g{<}G \wedge l{+}1 < L\} \cup \{((g,L{-}1),(g{+}1,0)) \mid g{+}1 < G \wedge l{<}L\}$. Let the transitions of each thread $i \in n\backslash\{0\}$ be $\{((G{-}1,l),(0,l{+}1)) \mid l{+}1 < L\}$. Then $\mathsf{d}((0,(0)_{i<n}),(G{-}1,(L{-}1)_{i<n})) = (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. ■

Using Knuth's Big-Omega notation [33], we obtain: $L{\geqslant}2 \Rightarrow \mathsf{diamax}(n) = \Omega(n)$.

The proof of Thm. IV.1.1 cannot be strengthened by better counting using the same program family. In Note D.3 we show that the diameter of the n-threaded program from this family is exactly $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. In this sense, the proof of Thm. IV.1.1 has reached its limit.

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 § II.2 examples, say, N2T6B or N2T7A, of diameter $7 > 6 = 3n = (2{\cdot}2{-}2{+}1)(2{-}1)n + (2{-}2)(2{-}1){\cdot}2$ 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 Thm. IV.2.2.3, adding arbitrarily many arbitrary threads to these examples would not raise the bound beyond linear anyway. A few more exceptions are known, i.e., n-threaded programs (for small n) for which the diameter is known to exceed the lower bound; no such exception is known to have generalizations for infinitely many n that could asymptotically improve Thm. IV.1.1. Further, on a set of over $5 \cdot 10^{24}$ n-threaded binary programs with $n \geqslant 5$, no deviations from the lower bound have been observed [21].

IV.2. Upper bounds

IV.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 $(\to_i)_{i<n}$ and an arbitrary state $(g,l) \in \mathsf{State}$ of this program until (but not including) Thm. IV.2.1.13. We are going to prove that $\mathsf{d}((g,l),s) = \mathsf{O}(n^c)$ 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, consider the *diagonal* $D \stackrel{\text{def}}{=} \mathsf{id}_{\mathsf{Glob}\times\mathsf{Loc}}$. Thread identifiers $i,j < n$ are called *confusable*, written $i{\sim}j$, iff $l_i = l_j \wedge {\to_i}\backslash D = {\to_j}\backslash D$. 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 are equal. Formally:

Definition IV.2.1.1. A map $\varphi\colon n \to n$ is called $\sim$*-invariant* iff $\forall i{<}n\colon i \sim \varphi(i)$. Program states $(\hat{g},\hat{l}),(\check{g},\check{l})$ are called *confusable*, written $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$, iff $\hat{g}{=}\check{g} \wedge \exists$ $\sim$-invariant $\varphi \in (n \hookrightarrow n)\colon\ \forall i{<}n\colon \hat{l}_i = \check{l}_{\varphi(i)}$. □

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

Example IV.2.1.2. Consider the binary case $\mathsf{Glob} = \{\alpha,\beta\}$ and $\mathsf{Loc} = \{0,1\}$ and a two-threaded program $(\{(\alpha 1,\alpha 1)\}, \{(\beta 0,\beta 0)\})$ having self-loops only. Let the initial state be $(g,l) = (\alpha,(0,0))$. The thread indexes 0 and 1 are confusable. The transposition $\varphi = \lambda i{<}2.\, 1{-}i$, which swaps the indexes of the threads, is $\sim$-invariant. Let $(\hat{g},\hat{l}) = (\beta,(0,1))$, and $(\check{g},\check{l}) = (\beta,(1,0))$. Then $\hat{g} = \check{g}$, $\hat{l}_0 = \check{l}_{\varphi(0)}$, and $\hat{l}_1 = \check{l}_{\varphi(1)}$. Therefore, $(\hat{g},\hat{l})$ and $(\check{g},\check{l})$ are confusable. Here, $\approx$ is not a classic bisimulation [15, Def. 7.1]: $(\check{g},\check{l})$ has a successor program state (namely, itself), whereas $(\hat{g},\hat{l})$ has no successors. □

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

Lemma IV.2.1.3. *If $\varphi\colon n \hookrightarrow n$ is $\sim$-invariant, so is φ^{-1}.*

Confusion of program states is an equivalence relation:

Lemma IV.2.1.4. *$\approx$ is an equivalence relation on* State.

We write $\mathsf{State}/_{\approx}$ for the set of equivalence classes.

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

Definition IV.2.1.5. A *loopless* thread transition relation is a member of $E \stackrel{\text{def}}{=} \mathfrak{P}\big((\mathsf{Glob}{\times}\mathsf{Loc})^2 \setminus D\big)$. □

Above, E stands for "*e*dges". Using this shorthand for loopless 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 IV.2.1.6. *For all $\hat{g},\check{g} \in \mathsf{Glob}$ and $\hat{l},\check{l} \in \mathsf{Loc}^n$, we*

have $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$ *iff* $\Big(\hat{g}{=}\check{g} \wedge \Big(\forall a,b \in \mathsf{Loc},\rightsquigarrow{\in}E\colon \big|\{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \hat{l}_t{=}b\}\big| = \big|\{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \check{l}_t{=}b\}\big|\Big)\Big)$.

Proof idea. Two sets have the same cardinality iff a bijection between them exists. ■

So program states are confusable iff they are the same up to renumbering the threads with confusable identifiers.

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

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

Lemma IV.2.1.7. $\forall\, s,s' \in \mathsf{State}\colon\ s{\approx}s' \Rightarrow \mathsf{d}((g,l),s) = \mathsf{d}((g,l),s')$.

Proof idea. Induction on $\min\{\mathsf{d}((g,l),s),\mathsf{d}((g,l),s')\}$. ■

From this result we directly conclude:

Lemma IV.2.1.8. $\forall\, s \in \mathsf{State}\colon \mathsf{d}((g,l),s) < \infty \Rightarrow \mathsf{d}((g,l),s) < \big|\mathsf{State}/_{\approx}\big|$.

Proof idea. Take a shortest path from (g,l) to s. Lem. IV.2.1.7 implies that any two program states on this path are nonconfusable. Hence, the index of the equivalence relation $\approx$ exceeds the path length. ■

It is unclear how to determine the index of $\approx$ from its definition. To get a polynomial upper bound on this index, we first establish a bijection between the set of equivalence classes and another set for which the cardinality will be easier to determine directly.

To this end, let V be the set of all maps

$$f\colon (\mathsf{Loc}\times E) \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}$$

such that, for all $a{\in}\mathsf{Loc}$ and all $\rightsquigarrow\, \in E$, we have

$$\|f(a,\rightsquigarrow)\|_1 \;=\; \big|\{t{<}n \mid l_t = a \wedge {\to_t}\backslash D = {\rightsquigarrow}\}\big|$$

(i.e., informally speaking, the sequence of values of $f(a,\rightsquigarrow)$ forms a partition of the number of all the threads starting in a and having the thread transition relation $\rightsquigarrow$ up to self-loops).

Lemma IV.2.1.9. *There is a bijection between* $\mathsf{State}/_{\approx}$ *and* $\mathsf{Glob}\times V$.

Proof idea. Construct a surjection from State to $\mathsf{Glob}{\times}V$ which delivers the same values for two states iff they are confusable (using Lem. IV.2.1.6 to prove it). Lift this surjection to a bijection from $\mathsf{State}/_{\approx}$ to $\mathsf{Glob}{\times}V$. ■

(An aside has to be made. The proof of our upper bound on diamax will use the fact that the aforementioned bijection—as any bijection—is injective. But we will not use the fact that the bijection is also surjective. The existence of a surjection only shows the difficulty of potential future attempts to tighten our upper bound on diamax.)

To estimate the cardinality of $\mathsf{Glob}{\times}V$, we employ a generalization of the binomial coefficients in which the upper argument can take arbitrary rational values:

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

with the convention that the empty product evaluates to 1.

The definition of the set from Lem. IV.2.1.9 depends on the transitions relations of the program. Now we bound the cardinality of this set by an expression over G, L, and n, which doesn't involve the aforementioned dependency:

Lemma IV.2.1.10. $|\mathsf{Glob}\times V| \leqslant G\cdot\max\big\{\prod_{i<L\cdot 2^{GL(GL-1)}} \binom{k_i+L-1}{L-1} \;\big|\; k_0,\ldots,k_{L\cdot 2^{GL(GL-1)}-1} \in \mathbb{N}_{\geqslant 0} \wedge \sum_{i<L\cdot 2^{GL(GL-1)}} k_i = n\big\}$.

Proof idea. Use the fact that there are $\binom{k+L-1}{L-1}$ ways to arrange k indistinguishable balls into L distinguishable baskets. ■

Recall that a product of constantly many nonnegative integers with a constant sum is maximal when these integers coincide. To simplify the maximum from Lem. IV.2.1.10, we will require a similar result:

Lemma IV.2.1.11. *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 idea. Generalize the claim to $k_i \in \mathbb{Q}_{\geqslant 0}$ $(i{<}t)$ and prove it by induction on $\big|\{i{<}t \mid k_i \neq \frac{n}{t}\}\big|$. ■

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

Corollary IV.2.1.12. $\big|\mathsf{State}/_{\approx}\big| \leqslant G\binom{n/t+L-1}{L-1}^t$ *where* $t = L\cdot 2^{GL(GL-1)}$.

Proof idea. Combine Lems. IV.2.1.9 to IV.2.1.11. ■

This helps to bound diamax by a polynomial in n:

Theorem IV.2.1.13. $\mathsf{diamax}(n) < G\binom{\frac{n}{L\cdot 2^{GL(GL-1)}}+L-1}{L-1}^{L\cdot 2^{GL(GL-1)}} = \mathsf{O}\big(n^{L(L-1)2^{GL(GL-1)}}\big)$.

Proof idea. Follows from Cor. IV.2.1.12 and Lem. IV.2.1.8. ■

To the best of our knowledge, $\mathsf{O}(n^{L(L-1)2^{GL(GL-1)}})$ is the first known *explicit* polynomial upper bound for diamax. Comparing the degree $L(L-1)2^{GL(GL-1)}$ to bounds from [34] from a theoretical viewpoint, we feel it comforting to see that our degree is relatively small: it involves *only one* exponentiation operation. Tightening Thm. IV.2.1.13 is an open problem, but we expect that the degree could be lowered further by reducing E from Def. IV.2.1.5. §§ IV.2.2 and IV.2.3 will be encouraging: in the setups described there, the degree will be provably 1.

Note IV.2.1.14. In the binary case we obtain $\mathsf{diamax}(n) = \mathsf{O}\big(n^{2\cdot 1\cdot 2^{4\cdot 3}}\big) = \mathsf{O}\big(n^{2^{13}}\big)$, improving over the $\mathsf{O}\big(n^{2^{17}}\big)$ bound from [20]. The proof of the $\mathsf{O}\big(n^{2^{13}}\big)$ bound shows that a diamax-realizing program has at most $2^{12} = 4096$ different thread transition relations. The reduction of the degree by a factor of 16 is a substantial and necessary step in the following sense: if anyone aims to understand the structure of diamax-realizing programs, they have to consider the transition relations of the threads manually or semi-automatically. We do not believe it to be doable directly for $2^{16} = 65536$ relations as in [20], but we believe that 4096 cases could be reduced after further optimizations to a manageable number (and, even in the worst case, such an inspection would be

more elementary than known monster explorations, such as [35]). □

IV.2.2. Strongly connected subprograms

Now we show a linear upper bound for the family of programs that have a subprogram with a strongly connected transition graph.

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:

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

where $\fatsemi$ is the left composition.

For the rest of this section, let $\mathcal{C}$ be the maximal local diameter (cf. Def. and Cor. III.9).

Lemma IV.2.2.1. *Let h be an m-threaded program whose transition graph is strongly connected and (g,l) a state of the program. Then every shared state can be reached from (g,l) in at most $\min\{\mathcal{C},(G{-}1)L^m,\mathsf{diam}(h)\}$ steps. Formally: $\forall\,(g,l) \in \mathsf{State},\, g'{\in}\mathsf{Glob}\ \exists\, l'{\in}\mathsf{Loc}^m\colon\ (g,l) \longrightarrow^{\leqslant \min\{\mathcal{C},(G-1)L^m,\mathsf{diam}(h)\}} (g',l')$.*

Proof idea. Take a shortest path from (g,l) to a program state with the shared part g'. Each state from $(\mathsf{Glob}\backslash\{g'\}) \times \mathsf{Loc}^m$ occurs in the path at most once. ■

Due to Lem. IV.2.2.1, we can change the shared state 'quickly' in a program with a strongly connected transition graph.

We define a *universal helper* of a program p as a subprogram h of p such that the transition graph of h is strongly connected. The name 'universal helper' is motivated by the fact that such a subprogram may help to lift certain sequences of local states of p (that are almost walks in the transition graph p, but the shared states do not match properly) to actual walks in the transition graph of p:

Lemma IV.2.2.2. *Let $n{\geqslant}m{\geqslant}1$, and let h be an m-threaded universal helper of an n-threaded program p. Then $\mathsf{diam}(p) \leqslant (\min\{\mathcal{C},(G{-}1)L^m,\mathsf{diam}(h)\}{+}1)(L{-}1)(n{-}m) + \mathsf{diam}(h)$.*

Proof idea. Given a walk from one program state of p to another, construct a (potentially different) path from the former state to the latter state in which transitions induced by h are interleaved with the transitions that gradually reduce the number of indices in which the local parts of the two states differ. ■

Let us restate the contents of Lem. IV.2.2.2 in terms of our diameter bounds.

Theorem IV.2.2.3. *Let $m \geqslant 1$ and $(p_n)_{n\geqslant m}$ be a family of multithreaded programs such that*
- *the transition graph of 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.*

Let $d = \mathsf{diam}(p_m)$ and $c = (\min\{\mathcal{C},(G{-}1)L^m,d\}+1)(L{-}1)$. Then $\forall\, n \geqslant m\colon \mathsf{diam}(p_n) \leqslant cn - cm + d$.

Notice that this upper bound is linear in n, and the coefficient c is bounded above by $(\mathcal{C}{+}1)(L{-}1)$, which does not depend on the family of programs.

An example of such a family of Thm. IV.2.2.3 would be a family from the proof of Thm. IV.1.1, in which thread 0 would in addition obtain all the reverse thread transitions; the upper bound then would match the lower bound of $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. Other examples can be obtained by adding more threads to N2T6B, N2T7B, or N3T9 from § II: their transition graphs are strongly connected.

Models of real-world programs may naturally possess universal helpers. This may be the case when we produce an abstraction of a system. For example, the coarsest abstract thread [36] 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 have strongly connected graphs, 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.

IV.2.3. Probabilistic Analysis

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.

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 IV.2.3.1. *The diameter of a program with a 1-threaded universal helper does not exceed $(GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$. Formally, for every n-threaded program p we have $(\exists\, i{<}n\ \forall\, g,g' \in \mathsf{Glob},\, l,l' \in \mathsf{Loc}\colon (g,l) \to_i^* (g',l')) \Rightarrow \mathsf{diam}(p) \leqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L$.*

Proof idea. Follows from Lem. IV.2.2.2. ■

Lem. IV.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 IV.2.3.2. *Let $\pi\colon (\mathsf{Glob}\times\mathsf{Loc})^2 \to [0,1]$ be a probability distribution on thread transitions for which $\forall\, g,g' \in \mathsf{Glob},\, l,l' \in \mathsf{Loc}\ \exists\, k \in \mathbb{N}_{\geqslant 0},\, (\tilde{l}_i)_{i=0}^k \in (\mathsf{Loc}^n)^{k+1}, (\tilde{g}_i)_{i=0}^k \in \mathsf{Glob}^{k+1}\colon\ (g,l) = (\tilde{g}_0,\tilde{l}_0) \wedge (g',l') = (\tilde{g}_k,\tilde{l}_k) \wedge \forall\, i{<}k\colon \pi((\tilde{g}_i,\tilde{l}_i),(\tilde{g}_{i+1},\tilde{l}_{i+1})) > 0$. Then $\lim_{n\to\infty} \mathrm{Prob}\big($ $\mathsf{diam}($an n-threaded random program$) \leqslant (GL{-}L{+}1)(L{-}1)n + (2{-}L)(G{-}1)L\big) = 1$.*

Proof idea. The probability for a thread to satisfy the prerequisites for Lem. IV.2.3.1 is positive. ■

Note IV.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. □

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, say, uniformly at random would most likely fail for large n: we would likely get only at-most–linear values. In contrast, the search from [20], for example, is nonrandom, structured, and informed.

Thm. IV.2.3.2 coarsely 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 [37]. Assuming that the operating system with the scheduler are 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, uncorrectable bit-flips are likely to turn the threads into random pieces of code while still maintaining their parallel execution. If particular, Thm. IV.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. Of course, the usual caveats apply: the probability of a thread transition depending only on the thread transition is a simplification of the reality, and Thm. IV.2.3.2 applies to a nondeterministic programming model. 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.

V. 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 this:

- 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. We consider

$$\mathsf{Reach} \stackrel{\text{def}}{=} \left\{ (p,s,s') \middle| \begin{array}{l} p \text{ is a program } \wedge\ s,s' \in \mathsf{State}_p \\ \wedge\ \mathsf{d}_p(s,s') < \infty \end{array} \right\},$$

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

$$\mathsf{NonReach} \stackrel{\text{def}}{=} \left\{ (p,s,s') \middle| \begin{array}{l} p \text{ is a program } \wedge\ s,s' \in \mathsf{State}_p \\ \wedge\ \mathsf{d}_p(s,s') = \infty \end{array} \right\},$$

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

The asymptotic notation hidden in the complexity class definitions assumes constant G and L and variable n. First, we deal with program-state–to–program-state reachability:

Theorem V.1. $\mathsf{Reach}, \mathsf{NonReach} \in \mathsf{NSpace}(\log n)$.

Proof idea. Nondeterministically search in a Petri net created to simulate an input program. In this Petri net, a place corresponds to the number of threads that started in a particular local state, have a particular thread transition relation up to self-loops, and currently have yet another particular local state (cf. Lem. IV.2.1.6). We keep track of the shared state separately. ■

The nondeterministic logarithmic space complexity of reachability is not unusual; see, e.g., [38, 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 $\mathsf{Reach}^{\mathsf{loc}}$ and $\mathsf{NonReach}^{\mathsf{loc}}$ into bitstrings. A thread transition relation is encoded using a bitstring of length $\left|(\mathsf{Glob}\times\mathsf{Loc})^2\right| = G^2L^2$. 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 [39] will 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 $(p,s,i,\tau) \in \mathsf{Reach}^{\mathsf{loc}} \cup \mathsf{NonReach}^{\mathsf{loc}}$ 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 $\big(\lfloor \log G \rfloor + 1\big) + \big(\lfloor \log L \rfloor + 1\big)$ bits.

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

Theorem V.2. $\mathsf{Reach}^{\mathsf{loc}}, \mathsf{NonReach}^{\mathsf{loc}} \in \mathsf{NC}^1$.

Proof idea. The number of threads participating on a path can be bounded using Def. and Cor. III.9, allowing for storing a finite table of answers in the state space of a regular automaton. Syntax checking is done by an NC^1-circuit. ■

We don't 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.

VI. Related work

In the narrowest sense, little has been published on the complexity of reachability in finite-state shared-memory multithreaded programs equipped with interleaving semantics. The PSPACE-completeness of deciding reachability in multithreaded programs follows directly from Lem. 3.2.3 in [16]; it is as usual tedious to argue about logspace reductions, and the proof not been formally published to the best of our knowledge. As for singling out the contribution of the

number of threads n, already [40, p. 126] 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. In the special case that all the threads of a program execute the same code, it has been observed [41–43] that the symmetrized program has roughly n^L states, where L is the number of the local states.

In a wider sense, a crisp review of search in parametrized concurrent programs based on only one template is given in [44, § 3.2]. Relaxing the communication model leads us to [45–47] (for rendezvous communication) or [48] (for token ring) among replicated processes.

Local reachability in our setting can be viewed as a consequence of the finite-basis and Petri-net theories [49, 50]. The set of programs can be viewed as a single well-structured transition system [51], yielding the decidability of thread-state reachability via a constant bound on the distances to a target state. However, [51] fixes the target 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 to prove that the bound does not depend on the state we start 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 abstraction of various kinds. Symmetrization is possible, e.g., in the context of plain search [52] or in combination with a CEGAR loop and context inference [53]. Introducing counters is the main way symmetrization is implemented; sometimes the counter is abstracted to reduce the size of the state space [54]. For parametrized programs, cutoff techniques are applied. For example, [55] 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 which start in a fixed set of initial states is possible [56]: searching in the parallel composition of infinitely many copies can be reduced to searching in the parallel composition of finitely many copies. The authors of [56] do not generalize their technique to copies of more than one template in their paper but say that such generalizations are possible. A (non–counterexample-based) abstraction refinement for parametrized multithreaded programs consisting of copies of one thread is shown in [57]. Assuming a bounded number of shared and local states as well as thread-local error states, we conjecture that the number of refinements from [57, Thm. 1] and the cutoff from [56] can roughly be bounded from above by the maximum local diameter. Search in multithreaded programs can also be reduced to Petri nets [58, 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" [59] and "For fundamental reasons, we cannot avoid the exponential explosion in the number of threads" [60] spread throughout research-level texts. Such claims, taken literally, lead to questioning 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, formalized in one particular way as Open Problem 1.12.3 in [23], is resolved in this paper in the negative. We also show that several other straightforward mathematical formulations of the above claims are all embarrassingly wrong: it turns out that n is *never* in the exponent of the expressions obtained.

The work which is closest to this paper is [20], which handles most topics of the present paper for the special case of binary programs. For the binary programs (and only for them), [20] proves a linear lower bound on diamax, a polynomial upper bound on diamax, a linear upper bound on the diameter for a special class of programs, a linear upper bound for randomly chosen programs, a constant upper bound on the local diameter, and nondeterministic logarithmic space complexity for both reachability and local reachability. However, the practical applicability of [20] is somewhat limited: almost *no* real-world program is binary. The present paper eliminates this restriction, considering *all* finite-state multithreaded programs equipped with the interleaving semantics. Further, several results in this paper, when restricted to the binary case, are stronger than that of [20]. While providing the full list of differences would immediately break the limits of this paper, we would like to mention several particular challenges faced during generalization.

As usual with generalizations, given any high-level result of [20] and its high-level proof idea, it is difficult to say whether it is generalizable to the nonbinary programs or whether it is an artifact of the binary ones; we will see both cases, and one does have to scrutinize every single detail of the corresponding low-level proof (otherwise, as Gauss said, $\frac{1}{2}$ proof = 0). Luckily, it turned out that some results of [20] need only few new ideas to be generalizable. An example is the linear lower bound on diamax: the proof of Thm. IV.1.1 is a generalization of the proof of [20, Thm. 5.0.1] (and, a posteriori, no additional ideas are needed), but the amount of details in the general-case proof is daunting when compared to the binary-case proof. For this paper, we could additionally show (cf. Note D.3) that the computation in the lower-bound proof for the generalization is at its best: a better lower bound, should it exist, would require a completely different

proof. Another lucky example is the linear upper bound on the diameter of a randomly chosen program (cf. § IV.2.3): the proof depends on another result (which was also generalized), but is a straightforward more-or-less syntactic adaptation of [20, § 5.3] otherwise.

Apart from these two examples, other results of this paper did require (a varying amount of) novel ideas beyond [20]. For instance, expressing a general upper bound on diamax involved deriving a new expression containing a binomial coefficient (cf. Thm. IV.2.1.13). This expression is tighter than [20, Thm. 5.1.7]: we reduced the degree of the polynomial representing the upper bound, restricted to the binary case, by the factor of 16 (cf. Note IV.2.1.14). Before writing the proofs in full length, we considered certain parts technically challenging (and, as often, we consider them easy afterwards):

- Proving that an upper bound on the local diameter is explicitly constructible (cf. Def. and Cor. III.9), whereas [20, Thm. 6.2] showed only the existence of the bound.
- Improving the computational complexity of local reachability from NL in [20, § 7] to NC^1 (cf. Thm. V.2).

Most results of the present paper, when restricted to the binary case, are similarly strong or even stronger than the results from [20]. Still, there is no free lunch: our bounds in Lem. IV.2.2.2 and Thm. IV.2.2.3, when restricted to the binary case, match [20, Thm. 5.2.1, Cor. 5.2.2] only up to a multiplicative constant. The inequalities in Lem. IV.2.2.2 and Thm. IV.2.2.3 are weaker; a crucial part of the proof of [20, Thm. 5.2.1] is tied to binary programs. The bounds in Lem. IV.2.2.2 and Thm. IV.2.2.3 are new in general; they are a nontrivial extension of the binary-case result. We invested considerable effort into reducing the multiplicative constant and do not see how to tighten these inequalities further, so, we still consider this task a challenge for the future. We had to pay for an increase in generality also by skipping the direct computation of the actual values of diamax in the general case, as the required computational time got astronomic already in [20, § 8].

Being a generalization, our paper intentionally follows [20] concerning the structure, terminology, examples, and certain claims. The reader might find similar or even the same formulations; however, these formulations are now (unless otherwise stated) to be interpreted in the parametrized context rather than in the binary one.

Our solutions have been partially inspired and influenced by the insights from symmetry reduction [41], process replication [45], finite-base arguments [49], vector addition systems and Petri nets [61], counter abstraction [54], low-complexity arguments in case the shared memory cannot read-and-write as a single atomic operation [62], pattern-based verification [63], and linear interfaces [64].

Outside formal methods, the diameter is sometimes defined as the maximal distance, being infinity for not strongly-connected graphs [65] (whereas we consider the maximal *finite* distance, as the transition graphs need not be strongly connected). Defined more widely, the notion of graph diameter is important in, e.g., [66] (standard algorithms on graphs), [67] (an old survey on diameter-related problems mostly for undirected graphs), [68] (spectral graph theory, restricted to strongly connected graphs), [69] (strongly connected Eulerian directed graphs without 2-cycles), [70] (networks), [71, 72] (centrality computation), and [73] (the best-case performance of the simplex method in linear programming).

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 $t = L2^{GL(GL-1)}$, 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 [20]. 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. Besides the bounds, we have achieved the following results:

- a polynomial upper bound for the diameter for a subclass of multithreaded programs; this bound is linear in n and comes close to 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 [20]).

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 [74]. 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 a counterexample (to 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, e.g., 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. Moreover, if a good upper bound on diamax (or $\mathcal{C}$ 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 and thus to coarsely rank 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 trying to avoid 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 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, then unexpected mathematical artifacts may emerge. One such artifact is that the state-explosion problem turns 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) 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 next research step in theory, we plan to tighten the aforementioned inequalities. While we do not know whether diamax could be bounded from above, say, by a linear function, we do expect that the upper bound could be lowered in principle; one way could start with determining a possibly large class of threads that cannot occur in diamax-realizing programs and subtracting this class while defining E in Def. IV.2.1.5. Another goal is to estimate the maximal local diameter $\mathcal{C}$ (perhaps, using [75]). Locating the general and local reachability problems in the hierarchy of parametric complexity classes is a yet another, orthogonal line of research [24].

Acknowledgments

This work would have been impossible without prior research contributions of Steffen Borgwardt from the University of Colorado Denver, USA. Financial support of the Software and Systems Engineering Research Group, led by Manfred Broy at TUM, Germany, is acknowledged.

References

[0] H. Sutter, "The free lunch is over: a fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, no. 3, Mar. 2005.

[1] S. Owens, S. Sarkar, and P. Sewell, "A better x86 memory model: x86-TSO (extended version)," Univ. of Cambridge, Tech. Rep. UCAM-CL-TR-745, 2009.

[2] P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen, "x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors," *CACM*, vol. 53, no. 7, pp. 89–97, 2010.

[3] The MITRE corporation. "CWE category: concurrency issues." (May 2017), [Online]. Available: http://cwe.mitre.org/data/definitions/557.html.

[4] Insomnia Security. "Concurrency vulnerabilities." (2011), [Online]. Available: http://www.owasp.org/images/8/8e/OWASP_NZDay_2011_BrettMoore_ConcurrencyVulnerabilities.pdf.

[5] N. G. Leveson and C. S. Turner, "Investigation of the Therac-25 accidents," *IEEE Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[6] K. Poulsen. "Software bug contributed to blackout." (Feb. 2014), [Online]. Available: http://www.securityfocus.com/news/8016.

[7] ——, "Tracking the blackout bug." (Apr. 2014), [Online]. Available: http://www.securityfocus.com/news/8412.

[8] P. L. Anderson and I. K. Geckil, "Northeast blackout likely to reduce US earnings by $6.4 billion," Tech. Rep., Aug. 2003, http://www.andersoneconomicgroup.com/Portals/0/upload/Doc544.pdf.

[9] Z. Manna and A. Pnueli, *Temporal verification of reactive systems – safety*. Springer, 1995, ISBN: 978-0-387-94459-3.

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

[11] P. Ročkai, "Model checking software," Ph.D. dissertation, Masaryk university, Jan. 2015.

[12] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. MIT Press, Dec. 1999, ISBN: 978-0-262-03270-4.

[13] G. J. Holzmann, "The model checker SPIN," *IEEE TSE*, vol. 23, no. 5, pp. 279–295, May 1997.

[14] ——, *The SPIN model checker: primer and reference manual*. Addison-Wesley, 2003, ISBN: 0-321-22862-6.

[15] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008, ISBN: 978-0-262-02649-9.

[16] D. Kozen, "Lower bounds for natural proof systems," in *FOCS*, IEEE, 1977, pp. 254–266.

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

[18] Wikipedia. "List of PSpace-complete problems." (Jun. 2019), [Online]. Available: http://en.wikipedia.org/wiki/List_of_PSPACE-complete_problems.

[19] N. D. Jones, *Computability and Complexity*. 2006.

[20] A. Malkis and S. Borgwardt, "Reachability in binary multithreaded programs is polynomial," in *ICDCS*, IEEE, Jun. 2017.

[21] A. Malkis. "Sequence A290642 at OEIS." (Aug. 9, 2017), [Online]. Available: http://oeis.org/A290642.

[22] tombom. "Interrupting query sends wrong thread ID." (Sep. 21, 2017), [Online]. Available: http://dba.stackexchange.com/questions/186545/interrupting-query-sends-wrong-thread-id.

[23] A. Malkis, "Cartesian abstraction and verification of multithreaded programs," Ph.D. dissertation, Albert-Ludwigs-Universität Freiburg, 2010.

[24] ——, "Parametrized complexity of reachability in multithreaded programs," research rep., 2019, In preparation.

[25] B. K. Szymański, "A simple solution to Lamport's concurrent programming problem with linear wait," in *ICS*, ser. ICS '88, ACM, 1988.

[26] A. Malkis and A. Podelski, "Refinement with exceptions," University of Freiburg, research rep., 2008. [Online]. Available: http://wwwbroy.in.tum.de/~malkis/MalkisPodelski-refinementWithExceptions_techrep.pdf.

[27] A. Malkis, A. Podelski, and A. Rybalchenko, "Thread-modular verification and Cartesian abstraction," research rep., 2006, TV'06.

[28] Wikipedia. "Peterson's algorithm." (2019), [Online]. Available: http://en.wikipedia.org/wiki/Peterson's_algorithm.

[29] C. Flanagan, S. N. Freund, S. Qadeer, and S. A. Seshia, "Modular verification of multithreaded programs," *Theoretical Computer Science*, vol. 338, no. 1-3, pp. 153–183, 2005.

[30] H. P. Barendregt and E. Barendsen, "Introduction to λ-calculus," Department of Computer Science, Radboud University of Nijmegen, Tech. Rep., 2000.

[31] C. Flanagan and S. Qadeer, "Thread-modular model checking," in *SPIN*, ser. LNCS, vol. 2648, Springer, 2003, pp. 213–224.

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

[33] D. E. Knuth, "Big Omicron and Big Omega and Big Theta," *SIGACT News*, vol. 8, no. 2, Apr. 1976.

[34] S. Schmitz, "Complexity hierarchies beyond elementary," *TOCT*, vol. 8, no. 1, 3:1–3:36, 2016.

[35] D. Gorenstein, R. Lyons, and R. Solomon, *The classification of the finite simple groups*, 40.1–40.8 vols. AMS, 1994–2018.

[36] S. K. Lahiri, A. Malkis, and S. Qadeer, "Abstract threads," in *VMCAI*, 2010, pp. 231–246.

[37] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors," in *ISCA'14*, 2014, pp. 361–372.

[38] R. Alur, S. Kannan, and M. Yannakakis, "Communicating hierarchical state machines," in *ICALP'99*, 1999, pp. 169–178.

[39] M. Li and P. M. B. Vitányi, "Kolmogorov complexity and its applications," in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. van Leeuwen, Ed., 1990, pp. 187–254.

[40] B. D. Lubachevsky, "An approach to automating the verification of compact parallel coordination programs. I," *Acta Inf.*, pp. 125–169, 1984.

[41] E. A. Emerson and A. P. Sistla, "Symmetry and model checking," *FMSD*, vol. 9, no. 1/2, pp. 105–131, 1996.

[42] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening, "Symbolic counter abstraction for concurrent software," in *CAV*, 2009.

[43] ——, "Context-aware counter abstraction," *FMSD*, 2010.

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

[45] A. P. Sistla and S. M. German, "Reasoning with many processes," in *LICS*, 1987, pp. 138–152.

[46] ——, "Reasoning about systems with many processes," in *JACM*, 1992.

[47] E. M. Clarke and O. Grümberg, "Avoiding the state explosion problem in temporal logic model checking," in *PODC*, 1987.

[48] M. C. Browne, E. M. Clarke, and O. Grümberg, "Reasoning about networks with many identical finite-state processes," *Inf. Comput.*, 1989.

[49] G. Higman, "Ordering by divisibility in abstract algebras," *Proc. London Math. Soc. (3)*, vol. 2, pp. 326–336, 1952.

[50] R. M. Karp and R. E. Miller, "Parallel program schemata," *JCSS*, vol. 3, no. 2, pp. 147–195, 1969, ISSN: 0022-0000.

[51] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y. Tsay, "General decidability theorems for infinite-state systems," in *LICS*, 1996, pp. 313–321.

[52] A. F. Donaldson and A. Miller, "Automatic symmetry detection for Promela," *Journal of Automated Reasoning*, vol. 41, no. 3–4, pp. 251–293, 2008.

[53] T. A. Henzinger, R. Jhala, and R. Majumdar, "Race checking by context inference," in *PLDI*, 2004, pp. 1–13.

[54] A. Pnueli, J. Xu, and L. D. Zuck, "Liveness with $(0, 1, \infty)$-counter abstraction," in *CAV*, 2002, pp. 107–122.

[55] E. A. Emerson and V. Kahlon, "Reducing model checking of the many to the few," in *CADE*, 2000, pp. 236–254.

[56] A. Kaiser, D. Kroening, and T. Wahl, "Dynamic cutoff detection in parameterized concurrent programs," in *CAV*, 2010, pp. 645–659.

[57] P. A. Abdulla, F. Haziza, and L. Holík, "All for the price of few," in *VMCAI*, Springer, 2013, pp. 476–495.

[58] H. Zhao, "Using the Karp-Miller tree construction to analyse concurrent finite-state programs," M.S. thesis, Trinity College, Oxford, UK, 2009.

[59] S. F. Siegel and G. S. Avrunin, "Improving the precision of INCA by eliminating solutions with spurious cycles," *IEEE TSE*, vol. 28, no. 2, 2002.

[60] A. Farzan, Z. Kincaid, and A. Podelski, "Inductive data flow graphs," *SIGPLAN Not.*, vol. 48, no. 1, 2013.

[61] J. Leroux, "Presburger vector addition systems," in *LICS*, 2013, pp. 23–32.

[62] J. Esparza, P. Ganty, and R. Majumdar, "Parameterized verification of asynchronous shared-memory systems," *JACM*, vol. 63, no. 1, 10:1–10:48, 2016.

[63] J. Esparza, P. Ganty, and T. Poch, "Pattern-based verification for multithreaded programs," *ACM Trans. Program. Lang. Syst.*, vol. 36, no. 3, 9:1–9:29, 2014.

[64] S. La Torre, P. Madhusudan, and G. Parlato, "Model checking parameterized concurrent programs using linear interfaces," in *CAV*, 2010.

[65] D. B. West, *Introduction to graph theory*, 2nd ed. Prentice Hall, 2001, ISBN: 0-13-014400-2.

[66] N. Alon, Z. Galil, and O. Margalit, "On the exponent of the all pairs shortest path problem," *J. Comput. Syst. Sci.*, 1997.

[67] F. Chung, "Diameters of communication networks," in *Proc. Sympos. Appl. Math. 34 Amer. Math. Soc.*, 1986, pp. 1–18.

[68] ——, "The diameter and Laplacian eigenvalues of directed graphs," *Electr. J. Comb.*, vol. 13, no. 1, 2006.

[69] P. Dankelmann, "The diameter of directed graphs," *J. Comb. Theory, Ser. B*, vol. 94, no. 1, pp. 183–186, 2005.

[70] E. Lehman, F. T. Leighton, and A. R. Meyer, *Mathematics for computer science*. Jun. 6, 2018, ISBN: 978-988-8407-06-4.

[71] A. Abboud, F. Grandoni, and V. V. Williams, "Subcubic equivalences between graph centrality problems, APSP and diameter," in *SODA*, 2015, pp. 1681–1697.

[72] Q.-S. Hua, H. Fan, M. Ai, L. Qian, Y. Li, X. Shi, and H. Jin, "Nearly optimal distributed algorithm for computing betweenness centrality," in *ICDCS*, 2016, pp. 271–280.

[73] G. B. Dantzig, *Linear programming and extensions* (Rand Corporation Research Study). Princeton Univ. Press, 1963.

[74] E. W. Mayr and A. R. Meyer, "The complexity of the finite containment problem for Petri nets," *JACM*, vol. 28, no. 3, pp. 561–576, 1981.

[75] C. Rackoff, "The covering and boundedness problems for vector addition systems," *TCS*, 1978.

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

[77] CWoo. "Homomorphism between partial algebras, Isomorphism," Planetmath. (Mar. 22, 2013), [Online]. Available: http://planetmath.org/homomorphismbetweenpartialalgebras#S0.SS0.SSSx1.p5.

[78] M. Broy, *Logische und methodische Grundlagen der Programm- und Systementwicklung, Datenstrukturen, funktionale, sequenzielle und objektorientierte Programmierung*, in collab. with A. Malkis. Springer, 2019, ISBN: 978-3-658-26301-0.

[79] M. Alekseyev. "Overapproximating a particular binomial." (Apr. 17, 2019), [Online]. Available: http://mathoverflow.net/a/328183.

[80] N. Immerman, "Nondeterministic space is closed under complementation," *SIAM J. Comput.*, vol. 17, no. 5, pp. 935–938, 1988.

[81] R. Szelepcsényi, "The method of forced enumeration for nondeterministic automata," *Acta Inf.*, vol. 26, no. 3, pp. 279–284, 1988.

Appendix A.
Notation used in the proofs

The appendices are dedicated to the proofs of the indicated statements in the main body of the paper and more examples. Before we begin, let us introduce some notation.

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 example, $X \overset{!}{\subseteq} Y$ means: we claim that $X \subseteq Y$ holds and proceed 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. For example, "$\varphi(\_,1) \wedge \psi(\_,0)$" means "$(\exists\, x\colon \varphi(x,1)) \wedge (\exists\, x\colon \psi(x,0))$." Similarly, in general claims that are not formally typeset as formulas the 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 $(0,l_1,l_2)$."

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

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

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:

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

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

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

"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.

Given an equivalence relation $\sim$ on a set X and some $x \in X$, we write

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

for the equivalence class of x with respect to $\sim$.

If the index set of a sequence σ is a product, say, $I{\times}J$, we sometimes opt for double indexing, writing the (i,j)-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 *cardinality* of a sequence is the number of elements in it:

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

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.

If $h = (\rightsquigarrow_i)_{i<m}$ is a subprogram of a program $p = (\to_i)_{i<n}$ such that $\forall\, i{<}m\colon \rightsquigarrow_i \,=\, \to_{f(i)}$ for some injective map $f\colon m{\hookrightarrow}n$, such a map is called an *embedding* of h into p.

Appendix B.
Proofs of claims from § I.2

Lemma B.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 $(\to_i)_{i<n} \in \mathcal{P}$ we have $\forall\, i < n\colon \to_i \,=\, \to_{\mathsf{id}_n(i)}$.

Transitivity. Let $(\rightsquigarrow_0, \ldots, \rightsquigarrow_{m-1})$ be a subprogram of $(\to_0, \ldots, \to_{n-1})$ and $(\to_0, \ldots, \to_{n-1})$ 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 $\forall\, i{<}m\colon \rightsquigarrow_i \,=\, \to_{f(i)}$ and $\forall\, j{<}n\colon \to_j \,=\, \rightarrowtail_{g(j)}$. Then the map $g \circ f$ is injective, and for all $i{<}m$ we have $\rightsquigarrow_i \,=\, \to_{f(i)} \,=\, \rightarrowtail_{g(f(i))} = \rightarrowtail_{(g\circ f)(i)}$. ∎

Appendix C.
Proofs of claims from § III

C.1. Preliminaries from general order theory

If a bijective order-homomorphism between preordered sets $(X, \lesssim_X)$ and $(Y, \lesssim_Y)$ exists, we call these preordered sets *order-isomorphic* [76, p. 14, Ch. 0]. (In general, an isomorphism is defined as a bijective homomorphism such that its inverse is also a homomorphism [77]. In our case, both definitions lead to the same result [78, Lemma 2.2].)

Proof of Prop. III.1

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 $\{b \in \mathbb{N}^{m-1}_{\geqslant 0} \mid (b_1, \ldots, b_{i-1}, k, b_{i+1}, \ldots, b_{m-1}) \in A\}$ 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, c is greater than a in the componentwise partial order on $\mathbb{N}^m_{\geqslant 0}$. $\lightning$ ∎

C.2. Well-foundedness and antitone maps

Proposition C.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 $\preceq$ 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 $z \preceq y$. 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 $z = y$. ■

Proof of Lem. III.2

a) If f is the empty partial map, its image is empty and, therefore, finite.
Thus, let us assume from now on that f is nonempty. We write $\preceq$ for the componentwise partial order on $\mathbb{N}_{\geqslant 0}^m$.
Let $A \stackrel{\text{def}}{=} \{x \in \mathsf{dom}\, f \mid \forall\, y \in \mathsf{dom}\, f\colon y \preceq x \Rightarrow y = x\}$ be the set of minimal elements of the domain of f. Then A is an antichain, hence finite by Prop. III.1. The partial order $\preceq$ is well-founded by Prop. C.2.1, and $\mathsf{dom}\, f$ 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 $x \in \mathsf{dom}\, f$ be arbitrary. Then the set $B \stackrel{\text{def}}{=} \{y \in \mathsf{dom}\, f \mid y \preceq x\}$ is nonempty, so it has a minimal element z.
Now we show that z lies in A. From $z \in B$ we get $z \in \mathsf{dom}\, f$. Let $y \in \mathsf{dom}\, f$ be arbitrary such that $y \preceq z$. From $z \in B$ we get $z \preceq x$. By transitivity, $y \preceq x$. From $y \in \mathsf{dom}\, f$ 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 $z \in \mathsf{dom}\, f \wedge \forall\, y \in \mathsf{dom}\, f\colon y \preceq z \Rightarrow y = z$. 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 $\mathsf{img}\, f \subseteq (M+1)$, 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,
- deciding whether $\mathsf{dom}\, f$ is empty (by choosing $I = \emptyset$),
- deciding, given a vector $a \in \mathbb{N}_{\geqslant 0}^m$, the membership $a \in \mathsf{dom}\, f$ (by choosing $I = m$ and $s_i =$ '=' for all $i < m$), and
- computing any y from 1) if such a y exists (by enumerating all the tuples from $\mathbb{N}_{\geqslant 0}^m$ and, for each tuple, evaluating the constraint and, if the constraint is satisfied, checking for membership in $\mathsf{dom}\, f$).

Consider the procedure in Alg. 1.
The algorithm first checks whether $\mathsf{dom}\, f = \emptyset$; if so, $\mathsf{img}\, f$ is empty. In this case, the procedure returns "empty".
Otherwise, a maximal (with respect to subset inclusion) antichain in $\mathsf{dom}\, f$ is constructed, beginning with an arbitrary element $x \in \mathsf{dom}\, f$. 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: 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 $y_i \geqslant \max\{c, c'\}$, 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 $c > c'$), and 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 $i < m$); we use the laws of absorption and annihilation (for the Boolean constant false) and associativity and commutativity to achieve maximal simplification. After this simplification, in each disjunct of F, each variable y_i occurs at most once. The resulting disjuncts of F, which are 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 $a_{i_1}, \ldots, a_{i_k} \in \mathbb{N}_{\geqslant 0}$), are examined one by one. Among these disjuncts, we search, using 1), 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 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 Prop. III.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 $A \subseteq \{a \in \mathsf{dom}\, f \mid \exists\, b \in \mathrm{D}\colon a \preceq b\}$, and the proof of a) implies that evaluating $f|_{\{a \in \mathsf{dom}\, f \mid \exists\, b \in D\colon\, a \preceq b\}}$ suffices to find the maximum of f. The evaluation of $f|_{\{a \in \mathsf{dom}\, f \mid \exists\, b \in D\colon\, a \preceq b\}}$ can be done, e.g., by looping through all $b \in D$ and, for each such b, looping through all $a \preceq b$. Since we thereby particularly evaluate $f|_A$, we necessarily encounter the maximal value of f in the process. ■

C.3. Application of order theory to programs

Proof of Lem. III.3

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

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

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 $t_0, \ldots, t_{k-1}$ counts all the thread transition relations, the image of φ does indeed not include the zero vector. We write $\preceq$ for the componentwise partial order on $\mathbb{N}_{\geqslant 0}^k \setminus \{k \times \{0\}\}$. Now we prove the claims of the lemma.

"φ is an order-homomorphism." By definition, φ is total. Now let $p = (\rightsquigarrow_0, \ldots, \rightsquigarrow_{m-1})$ and $q = (\to_0, \ldots, \to_{n-1})$ 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 $f|_{\{i<m \mid \rightsquigarrow_i = t_r\}}$ 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 $\varphi(p) \preceq \varphi(q)$.

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

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

  choose any such x;

  $\mathrm{D} := \{\mathrm{x}\}$; // D always stores an antichain in $\mathsf{dom}\, f$

  repeat

    grow := false;

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

    while $\mathrm{F} \neq \mathrm{false} \wedge \mathrm{F}$ *is a nonempty disjunction* $\wedge \neg \mathrm{grow}$ **do**

      d := any disjunct of F;

      Remove d from F;

      if $\exists\, y \in \mathsf{dom}\, f \colon \mathrm{d}$ **then** // using the algorithm from 1)

        Compute any such y;

        grow := true;

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

  until $\neg$grow; // after the loop, D is a maximal antichain in $\mathsf{dom}\, f$

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

else return "empty" // in this case $\mathsf{dom}\, f = \emptyset$

"If $\varphi(p) \preceq \varphi(q)$, then p is a subprogram of q." So let $\varphi(p) \preceq \varphi(q)$. For each $r{<}k$ let $S_r = \{i{<}m \mid \rightsquigarrow_i = t_r\}$ and $S'_r = \{i{<}n \mid \rightarrow_i = t_r\}$. 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 $f = \bigcup_{r<k} f_r$. Notice that the sets S_r for $r{<}k$ are pairwise disjoint and that $\bigcup_{r<k} S_r = m$. Thus, f is a mapping $m \to n$. Notice that the sets S'_r for $r{<}k$ are also pairwise disjoint. Thus, $f \colon m \hookrightarrow n$ is injective.

Now we prove that $\forall\, i{<}m \colon \rightsquigarrow_i = \rightarrow_{f(i)}$. 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 $\rightarrow_{f(i)} = t_r$. Combining, we obtain $\rightsquigarrow_i = \rightarrow_{f(i)}$.

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

Proof of Lem. III.4

In the following, we write $\preceq$ for the componentwise partial order on $\mathbb{N}_{\geqslant 0}^k \setminus \{k{\times}\{0\}\}$ 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 $n \in \mathbb{N}_+$, $p, q \in \mathcal{P}$, 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 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 $n \hookrightarrow n{-}1$, ↯. 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 § I.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:

$$\forall\, n \in \mathbb{N}_+,\ \ n\text{-threaded } p \in \mathcal{P} \colon \quad \|\varphi(p)\|_1 = n\,. \qquad (1)$$

Since there are exactly $k' \stackrel{\text{def}}{=} \left|\mathfrak{P}\big((\mathsf{Glob} \times \mathsf{Loc})^2\big)\right| = 2^{G^2 L^2}$ 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 $\mathbb{N}_{\geqslant 0}^k \setminus \{k{\times}\{0\}\}$. Therefore, $k{\geqslant}k'$. If k were strictly greater than k', then $\mathbb{N}_{\geqslant 0}^k \setminus \{k{\times}\{0\}\}$ would have a unit vector outside the image of

single-threaded programs under φ, which, given the fact that φ is onto, would contradict (1). So $k{=}k'$, and the restriction of φ to single-threaded programs is a bijection to the set of unit vectors of $\mathbb{N}_{\geqslant 0}^k \setminus \{k{\times}\{0\}\}$.

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 $\varphi\big((\to_i)_{i<n}\big) \stackrel{!}{=} \Big(\big|\{i{<}n \mid \to_i = t_r\}\big|\Big)_{r<k}$ for all programs $(\to_i)_{i<n}$ 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 $m{<}n$. 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

$$\varphi(q) \;=\; \Big(\big|\{i < n{-}1 \mid \to_i = t_r\}\big|\Big)_{r<k}. \qquad (2)$$

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 $v \stackrel{\text{def}}{=} \varphi(p) - e^{s'}$. Note that $\|v\|_1 = \|\varphi(p)\|_1 - 1 \stackrel{(1)}{=} n{-}1$. Since φ is onto, $\varphi(q') = v$ for some $q' \in \mathcal{P}$. Due to (1), 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 (2), $(\varphi(q))_s \geqslant v_s$. The definition of s implies $(\varphi(p))_s \geqslant v_s{+}1$. The definition of v implies $v_s = (\varphi(p){-}e^{s'})_s = (\varphi(p))_s - e^{s'}_s \geqslant (v_s + 1) - 0 = v_s + 1$, ↯.

Thus, our assumption is wrong, and $s{=}s'$. Therefore, $\to_{n-1} = t_s$. Then $(\varphi(p))_s = (\varphi(q))_s{+}1 \stackrel{(2)}{=} \big|\{i < n{-}1 \mid \to_i = t_s\}\big| + 1 = \big|\{i{<}n \mid \to_i = t_s\}\big|$. For $r \in k \setminus \{s\}$, we have $(\varphi(p))_r = (\varphi(q))_r + 0 \stackrel{(2)}{=} \big|\{i < n{-}1 \mid \to_i = t_r\}\big| =$ [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 $a \in \mathbb{N}_{\geqslant 0}^k \setminus \{k \times \{0\}\}$. Each preimage $p \in \varphi^{-1}(a)$ is $\|a\|_1$-threaded due to (1). 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 $a \in \mathbb{N}_{\geqslant 0}^k \setminus \{k \times \{0\}\}$. 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 $\varphi^{-1}(a)$. ■

Proof of Thm. III.5

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

a) Lem. III.3 implies the existence of some $k \in \mathbb{N}_+$ and some order-epimorphism $\varphi\colon \mathcal{P} \twoheadrightarrow \mathbb{N}_{\geqslant 0}^k \setminus \{k{\times}\{0\}\}$, where the codomain is equipped with the componentwise partial order, which we write as $\preceq$.
Let $A = \{\varphi(p) \mid p \in \operatorname{dom} f\}$.
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 $(\operatorname{dom} f) \cap \varphi^{-1}(a)$. For that, let $p, q \in \operatorname{dom} f$ 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 $f|_{(\operatorname{dom} f)\cap\varphi^{-1}(a)}$ is constant.
Thus, the map
$g\colon A \to \mathbb{N}_{\geqslant 0}, \quad a \mapsto f(p)$ for any $p \in (\operatorname{dom} f) \cap \varphi^{-1}(a)$
is well defined. It is also antitone, so, by Lem. III.2a), $\operatorname{img} g$ is finite. Since $\forall\, p \in \operatorname{dom} f\colon f(p) = g(\varphi(p))$, we obtain

$$\operatorname{img} f \subseteq \operatorname{img} g\,. \qquad (3)$$

Hence, $\operatorname{img} f$ is also finite.

b) Consider k, φ, g, and its domain $A = \operatorname{img}(\varphi|_{\operatorname{dom} f})$ from above. The very definition of g implies $\operatorname{img} g \subseteq \operatorname{img} f$, so, using (3), $\operatorname{img} g = \operatorname{img} f$. We will apply the algorithm from Lem. III.2b) to g. To this end, it suffices to supply the algorithms solving the following problems:

- Decide, given an arbitrary set $I \subseteq k$ and a sequence of pairs $(s_i, a_i)_{i\in I} \in (\{\text{`}{=}\text{'}, \text{`}{\geqslant}\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$, whether some $y \in \operatorname{dom} g$ exists satisfying $\bigwedge_{i\in I}\big((s_i = \text{`}{=}\text{'} \wedge y_i = a_i) \vee (s_i = \text{`}{\geqslant}\text{'} \wedge y_i \geqslant a_i)\big)$. According to Lem. III.4, 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 some $I \subseteq k$ and a sequence of pairs $(s_i, a_i)_{i\in I} \in (\{\text{`}{=}\text{'}, \text{`}{\geqslant}\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$ are given. Let $I' = \{t_i \mid i \in I\}$ and $(s'_{t_i}, a'_{t_i}) = (s_i, a_i)$ for all $i \in I$. Notice that there is some $y \in (\operatorname{dom} g) = \operatorname{img}(\varphi|_{\operatorname{dom} f})$ satisfying $\bigwedge_{i\in I}\big((s_i = \text{`}{=}\text{'} \wedge y_i = a_i) \vee (s_i = \text{`}{\geqslant}\text{'} \wedge y_i \geqslant a_i)\big)$ iff there is some $(\to_i)_{i<n} \in \operatorname{dom} f$ satisfying $\bigwedge_{\rightsquigarrow\in I'}\Big(\big(s'_{\rightsquigarrow} = \text{`}{=}\text{'} \wedge \big|\{i{<}n \mid \to_i = \rightsquigarrow\}\big| = a'_{\rightsquigarrow}\big) \vee \big(s'_{\rightsquigarrow} = \text{`}{\geqslant}\text{'} \wedge \big|\{i{<}n \mid \to_i = \rightsquigarrow\}\big| \geqslant a'_{\rightsquigarrow}\big)\Big)$:
  - "⇒": Given a y as stated, let $(\to_i)_{i<n}$ be an arbitrary member of $\varphi^{-1}(y) \cap (\operatorname{dom} f)$.
  - "⇐": Given a program $(\to_i)_{i<n}$ as stated, let $y = \varphi\big((\to_i)_{i<n}\big)$.

  An algorithm answering whether, given a set I' of thread transition relations and an I'-indexed sequence of pairs $(s'_{\rightsquigarrow}, a'_{\rightsquigarrow})_{\rightsquigarrow\in I'} \in (\{\text{`}{=}\text{'}, \text{`}{\geqslant}\text{'}\} \times \mathbb{N}_{\geqslant 0})^{I'}$, there is any

Algorithm 2: Evaluate g
Input: $a \in A$
Output: $g(a)$
foreach $\mathrm{p} \in \varphi^{-1}(a)$ **do**
  if $\mathrm{p} \in \mathsf{dom}\, f$ **then return** $f(\mathrm{p})$

$(\to_i)_{i<n} \in \mathsf{dom}\, f$ satisfying $\bigwedge_{\rightsquigarrow \in I'} \Big(\big(s'_{\rightsquigarrow} = \text{`='} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| = a'_{\rightsquigarrow}\big) \vee \big(s'_{\rightsquigarrow} = \text{`}\geqslant\text{'} \wedge \big|\{i<n \mid \to_i = \rightsquigarrow\}\big| \geqslant a'_{\rightsquigarrow}\big)\Big)$ 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 Lem. III.4, the problem is solvable by Alg. 2. ■

Proof of Lem. III.6

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 $\mathsf{dom}\, f = \Big\{(\to_i)_{i<n} \in \mathcal{P} \;\Big|\; \exists i<n\colon \to_i = \rightsquigarrow \wedge\, \mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n\times\{a\}), i, (g', a')\big) < \infty\Big\}$, $(\mathsf{dom}\, f) \ni (\to_i)_{i<n} \mapsto \min\Big\{\mathsf{d}^{\mathsf{loc}}_{(\to_i)_{i<n}}\big((g, n\times\{a\}), i, (g', a')\big) \;\Big|\; i<n \wedge \rightsquigarrow = \to_i\Big\}$.

Now we will show that f is antitone. Let $p, q \in \mathsf{dom}\, f$ 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 $p \in \mathsf{dom}\, f$, 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 $h^{-1}\colon (\mathsf{img}\, h) \hookrightarrow n$ be the inverse of h on its image. We define a sequence $\hat{\sigma} = (\hat{\sigma}_k)_{k\leqslant f(p)} \in \big(\mathsf{Glob}\times\mathsf{Loc}^m\big)^{f(p)+1}$ by $\hat{\sigma}_k \stackrel{\text{def}}{=} \left(\check{g}^{[k]}, \left(\begin{cases} \check{l}^{[k]}_{h^{-1}(j)}, & \text{if } j \in \mathsf{img}\, h, \\ a, & \text{otherwise} \end{cases}\right)_{j<m}\right)$, where $(\check{g}^{[k]}, \check{l}^{[k]}) \stackrel{\text{def}}{=} \sigma_k$, 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 $q \in \mathsf{dom}\, f$ and $f(q) \leqslant f(p)$. Since p and q were arbitrary, we have shown that f is antitone.

Thm. III.5a) 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 $p = (\to_i)_{i<n}$, an arbitrary $i<n$ such that $\to_i = \rightsquigarrow$ and $\mathsf{d}^{\mathsf{loc}}_p\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ be given. Then $p \in \mathsf{dom}\, f$. The definition of f implies the existence of some $\hat{\imath} < n$ such that $\to_{\hat{\imath}} = \rightsquigarrow$ 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 $\hat{\sigma}_k \stackrel{\text{def}}{=} \left(\check{g}^{[k]}, \left(\begin{cases} \check{l}^{[k]}_j, & \text{if } j \notin \{i, \hat{\imath}\}, \\ \check{l}^{[k]}_i, & \text{if } j = \hat{\imath}\,, \\ \check{l}^{[k]}_{\hat{\imath}}, & \text{if } j = i \end{cases}\right)_{j<n}\right)$, where $(\check{g}^{[k]}, \check{l}^{[k]}) \stackrel{\text{def}}{=} \sigma_k$, 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 $\mathsf{d}^{\mathsf{loc}}_p\big((g, n\times\{a\}), i, (g', a')\big) \leqslant c$.

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 Thm. III.5b). To this end, it suffices to supply the algorithms solving the following problems:

- Membership of a given program in $\mathsf{dom}\, f$.
  Let $p = (\to_i)_{i<n} \in \mathcal{P}$ be arbitrary. To test whether p belongs to $\mathsf{dom}\, f$, we first search for an $i<n$ such that $\to_i = \rightsquigarrow$ and return "false" if we find none. If one is found, we check for this i whether $\mathsf{d}^{\mathsf{loc}}_p\big((g, n\times\{a\}), i, (g', a')\big) < \infty$ 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 an arbitrary set $I \subseteq \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big)$ and an I-indexed sequence of pairs $(s_{\rightarrowtail}, b_{\rightarrowtail})_{\rightarrowtail\in I} \in (\{\text{`='}, \text{`}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$, whether some $(\to_i)_{i<n} \in \mathsf{dom}\, f$ satisfying $\bigwedge_{\rightarrowtail\in I} \Big(\big(s_{\rightarrowtail} = \text{`='} \wedge \big|\{i<n \mid \to_i = \rightarrowtail\}\big| = b_{\rightarrowtail}\big) \vee \big(s_{\rightarrowtail} = \text{`}\geqslant\text{'} \wedge \big|\{i<n \mid \to_i = \rightarrowtail\}\big| \geqslant b_{\rightarrowtail}\big)\Big)$ exists.
  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 $I \subseteq \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big)$ and an I-indexed sequence of pairs $(s_{\rightarrowtail}, b_{\rightarrowtail})_{\rightarrowtail\in I} \in (\{\text{`='}, \text{`}\geqslant\text{'}\} \times \mathbb{N}_{\geqslant 0})^I$ 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 $\mathrm{start} \notin (k\times\mathsf{Loc}) \cup \mathsf{Glob}$. We construct the set of the Petri net's places as the disjoint union $(k\times\mathsf{Loc}) \,\dot\cup\, \mathsf{Glob} \,\dot\cup\, \{\mathrm{start}\}$.
  Initially, the Petri net has one token in start and b_{t_j} tokens in (j, a) for each $j<k$ such that $t_j \in I$; there are no other tokens in the initial marking.
  To define the transitions, let $\hat{I} = \mathfrak{P}\big((\mathsf{Glob}\times\mathsf{Loc})^2\big) \setminus \{\rightarrowtail \in I \mid s_{\rightarrowtail} = \text{`='}\}$. First, the Petri net has $|\hat{I}|$ transitions indexed by $\hat{I}$; for each $j<k$ such that $t_j \in \hat{I}$, 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). Second, the Petri net has a transition that completely leaves start: this transition consumes a token from start and adds a token to the place g. Third and finally, the Petri net has 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., [50]) 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 $\rightarrowtail \in I$ with $s_{\rightarrowtail} =$ '$\geqslant$', exactly $b_{\rightarrowtail}$ copies of each $\rightarrowtail \in I$ with $s_{\rightarrowtail} =$ '$=$', and an arbitrary number of other thread transition relations.

- 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. ∎

Proof of Cor. III.7

According to Lem. III.6, 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 $p=(\rightarrow_i)_{i<n}$, and all $i<n$, if $\rightarrow_i = \rightsquigarrow$ 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 $p=(\rightarrow_i)_{i<n}$. 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 $\mathsf{d}_p^{\mathsf{loc}}\big((g,n\times\{a\}),i,(g',a')\big) \leqslant \zeta(g,g',a,a',\rightarrow_i) \leqslant c$.

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

Before we remove the restriction to uniform program states, we argue that local distances stay invariant under renaming of local states. More formally:

Lemma C.3.1. *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*

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

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

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

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.

"$\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) \overset{!}{\leqslant} \mathsf{d}_q^{\mathsf{loc}}\Big(\big(g,(h_j(l_j))_{j<n}\big),i,\big(g',h_i(b)\big)\Big)$":
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 $\sigma = \Big(\big(\hat{g}^{[r]},\hat{l}^{[r]}\big)\Big)_{r\leqslant k}$ in the transition graph of q 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 $t\colon k\to n$ that tells us which thread takes a step at each time point, i.e., such that for each $r<k$ we have $\Big(\hat{g}^{[r]},\hat{l}^{[r]}_{t(r)}\Big) \rightsquigarrow_{t(r)} \Big(\hat{g}^{[r+1]},\hat{l}^{[r+1]}_{t(r)}\Big)$ and $\forall\, j\in n\setminus\{t(r)\}\colon \hat{l}_j^{[r]} = \hat{l}_j^{[r+1]}$. Then for each $r<k$ we have $\Big(\hat{g}^{[r]},h_{t(r)}\Big(h_{t(r)}^{-1}\Big(\hat{l}^{[r]}_{t(r)}\Big)\Big)\Big) \rightsquigarrow_{t(r)} \Big(\hat{g}^{[r+1]},h_{t(r)}\Big(h_{t(r)}^{-1}\Big(\hat{l}^{[r+1]}_{t(r)}\Big)\Big)\Big)$, which, using the assumption, implies $\Big(\hat{g}^{[r]},h_{t(r)}^{-1}\Big(\hat{l}^{[r]}_{t(r)}\Big)\Big) \rightarrow_{t(r)} \Big(\hat{g}^{[r+1]},h_{t(r)}^{-1}\Big(\hat{l}^{[r+1]}_{t(r)}\Big)\Big)$. Let $\check{l}^{[r]} \overset{\text{def}}{=} \Big(h_j^{-1}\big(\hat{l}_j^{[r]}\big)\Big)_{j<n}$ for all $r\leqslant k$. Then, for each $r<k$ we have $\big(\hat{g}^{[r]},\check{l}^{[r]}_{t(r)}\big) \rightarrow_{t(r)} \big(\hat{g}^{[r+1]},\check{l}^{[r+1]}_{t(r)}\big)$ and $\forall\, j\in n\setminus\{t(r)\}\colon \check{l}_j^{[r]} = h_j^{-1}\big(\hat{l}_j^{[r]}\big) = h_j^{-1}\big(\hat{l}_j^{[r+1]}\big) = \check{l}_j^{[r+1]}$. Therefore, the sequence $\check{\sigma} \overset{\text{def}}{=} \Big(\big(\hat{g}^{[r]},\check{l}^{[r]}\big)\Big)_{r\leqslant k}$ is a walk in the transition graph of p. The walk starts in $\Big(\hat{g}^{[0]},\Big(h_j^{-1}\big(\hat{l}_j^{[0]}\big)\Big)_{j<n}\Big) = \big(\hat{g}^{[0]},(l_j)_{j<n}\big) = (g,l)$ and ends in the program state $\Big(\hat{g}^{[k]},\Big(h_j^{-1}\big(\hat{l}_j^{[k]}\big)\Big)_{j<n}\Big) = \Big(g',\Big(h_j^{-1}\Big(\hat{l}_j^{[k]}\Big)\Big)_{j<n}\Big)$, whose lo-

cal part of thread i is $h_i^{-1}\Big(\hat{l}_i^{[k]}\Big) = b$. Note that $\mathsf{length}(\check{\sigma})$ $= k$. Therefore, $\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) \leqslant k$.

"$\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) \overset{!}{\geqslant} \mathsf{d}_q^{\mathsf{loc}}\Big(\big(g,(h_j(l_j))_{j<n}\big),i,\big(g',h_i(b)\big)\Big)$": 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 $\sigma = \Big(\big(\hat{g}^{[r]},\hat{l}^{[r]}\big)\Big)_{r\leqslant k}$ in the transition graph of p such that $\hat{g}^{[0]} = g$, $\hat{l}^{[0]} = l$, $\hat{g}^{[k]} = g'$, and $\hat{l}_i^{[k]} = b$. There is a map $t\colon k\to n$ that tells us which thread takes a step at each time point, i.e., such that for each $r<k$ we have $\big(\hat{g}^{[r]},\hat{l}^{[r]}_{t(r)}\big) \to_{t(r)} \big(\hat{g}^{[r+1]},\hat{l}^{[r+1]}_{t(r)}\big)$ and $\forall\, j \in n\backslash\{t(r)\}\colon \hat{l}_j^{[r]} = \hat{l}_j^{[r+1]}$. The assumption implies that for each $r<k$ we have $\Big(\hat{g}^{[r]},h_{t(r)}\big(\hat{l}^{[r]}_{t(r)}\big)\Big) \rightsquigarrow_{t(r)} \Big(\hat{g}^{[r+1]},h_{t(r)}\big(\hat{l}^{[r+1]}_{t(r)}\big)\Big)$. Let $\check{l}^{[r]} \overset{\text{def}}{=} \Big(h_j\big(\hat{l}_j^{[r]}\big)\Big)_{j<n}$ for each $r\leqslant k$. Then, for each $r<k$ we have $\big(\hat{g}^{[r]},\check{l}^{[r]}_{t(r)}\big) \rightsquigarrow_{t(r)} \big(\hat{g}^{[r+1]},\check{l}^{[r+1]}_{t(r)}\big)$ and $\forall\, j\in n\backslash\{t(r)\}\colon \check{l}_j^{[r]} = h_j\big(\hat{l}_j^{[r]}\big)$ $= h_j\big(\hat{l}_j^{[r+1]}\big) = \check{l}_j^{[r+1]}$. Therefore, the sequence $\check{\sigma} \overset{\text{def}}{=} \Big(\big(\hat{g}^{[r]},\check{l}^{[r]}\big)\Big)_{r\leqslant k}$ is a walk in the transition graph of q. The walk starts in

$$\Big(\hat{g}^{[0]},\Big(h_j\big(\hat{l}_j^{[0]}\big)\Big)_{j<n}\Big) = \Big(g,\big(h_j(l_j)\big)_{j<n}\Big)$$

and ends in the program state

$$\Big(\hat{g}^{[k]},\Big(h_j\big(\hat{l}_j^{[k]}\big)\Big)_{j<n}\Big) = \Big(g',\Big(h_j\big(\hat{l}_j^{[k]}\big)\Big)_{j<n}\Big),$$

whose local part of thread i is $h_i\Big(\hat{l}_i^{[k]}\Big) = h_i(b)$. Note that $\mathsf{length}(\check{\sigma}) = k$. Therefore, $\mathsf{d}_q^{\mathsf{loc}}\big((g,(h_j(l_j))_{j<n}),i,$ $\big(g',h_i(b)\big)\big) \leqslant k$. ■

Proof of Thm. III.8

We fix c to be the smallest constant from Cor. III.7. Let $n\in\mathbb{N}_+$. Let $p = (\to_i)_{i<n}$ 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 $\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) < \infty$. It suffices to show that $\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) \overset{!}{\leqslant} c$. For each $j<n$ we let

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

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

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

for all $\tilde{g},\tilde{g}' \in \mathsf{Glob}$, $\tilde{l},\tilde{l}' \in \mathsf{Loc}$, and $j<n$. We set $q \overset{\text{def}}{=} (\rightsquigarrow_j)_{j<n}$. Lem. C.3.1 implies $\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big)$ $= \mathsf{d}_q^{\mathsf{loc}}\big((g,n\times\{l_i\}),i,(g',h_i(b))\big)$ $=$ [since $h_i = \mathsf{id}_{\mathsf{Loc}}$] $\mathsf{d}_q^{\mathsf{loc}}\big((g,n\times\{l_i\}),i,(g',b)\big)$. Therefore, $\mathsf{d}_q^{\mathsf{loc}}\big((g,n\times\{l_i\}),i,$ $(g',b)\big) < \infty$. Cor. III.7 implies $\mathsf{d}_q^{\mathsf{loc}}\big((g,n\times\{l_i\}),i,(g',b)\big)$ $\leqslant c$. Thus, $\mathsf{d}_p^{\mathsf{loc}}\big((g,l),i,(g',b)\big) \leqslant c$.

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-threaded program p, $i<n\colon \mathsf{d}_p^{\mathsf{loc}}\big((g,n\times\{a\}),i,(g',a')\big) <$ $\infty \Rightarrow \mathsf{d}_p^{\mathsf{loc}}\big((g,n\times\{a\}),i,(g',a')\big) \leqslant \tilde{c}\Big\}$. 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 $\mathsf{d}_p^{\mathsf{loc}}((g,n\times\{a\}),i,(g',a')) < \infty$ but $\mathsf{d}_p^{\mathsf{loc}}((g,n\times\{a\}),i,(g',a')) \not\leqslant c-1$. Then, $\mathsf{d}_p^{\mathsf{loc}}((g,n\times\{a\}),$ $i,(g',a')) \geqslant c$, contradicting our assumption. Thus, our assumption was wrong and $\max(\mathsf{img}\,\mathsf{diamax}^{\mathsf{loc}}) = c$. ■

The top level of computing $\mathcal{C}$ is Alg. 3, 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 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 $\mathsf{d}^{\mathsf{loc}}\big((g,n\times\{a\}),i,(g',a')\big) \leqslant \zeta(g,g',a,a',\rightsquigarrow)$. Following the involved lemmas, this algorithm can be expanded to any depth. For example, one informal but convenient way to look at the next deeper level (i.e., computing $\zeta(g,g',a,a',\rightsquigarrow)$) would be to apply the higher-order function defined by Thm. III.5 to the three closures listed towards the end of the proof of Lem. III.6 for the map f provided at the beginning of Lem. III.6; the imperative counterpart to this combined higher-order program can be obtained by inlining the three closures.

Algorithm 3: Computing the maximal local diameter $\mathcal{C}$
Input: $\mathsf{Glob},\mathsf{Loc}$
Program variables: $\mathrm{g},\mathrm{g}',\ \mathrm{a},\mathrm{a}',\ \rightsquigarrow$
Output: $\mathcal{C}$
$\mathcal{C} := 0$;
foreach $\mathrm{g},\mathrm{g}'\in\mathsf{Glob}$, $\mathrm{a},\mathrm{a}'\in\mathsf{Loc}$, $\rightsquigarrow\,\subseteq(\mathsf{Glob}\times\mathsf{Loc})^2$ **do**
  if $\zeta(\mathrm{g},\mathrm{g}',\mathrm{a},\mathrm{a}',\rightsquigarrow) > \mathcal{C}$ **then** $\mathcal{C} := \zeta(\mathrm{g},\mathrm{g}',\mathrm{a},\mathrm{a}',\rightsquigarrow)$;
return $\mathcal{C}$

Appendix D. Proofs of claims from § IV.1

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 D.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 $\mathsf{diamax}(n)$ $\overset{!}{=} (L-1)n$ 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

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

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 D.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 parts; any walk using more than G shared parts would have a self-intersection and could be shortened. Thus, each shared part is used at most once in σ, implying $\mathsf{length}(\sigma) < G$.

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

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

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

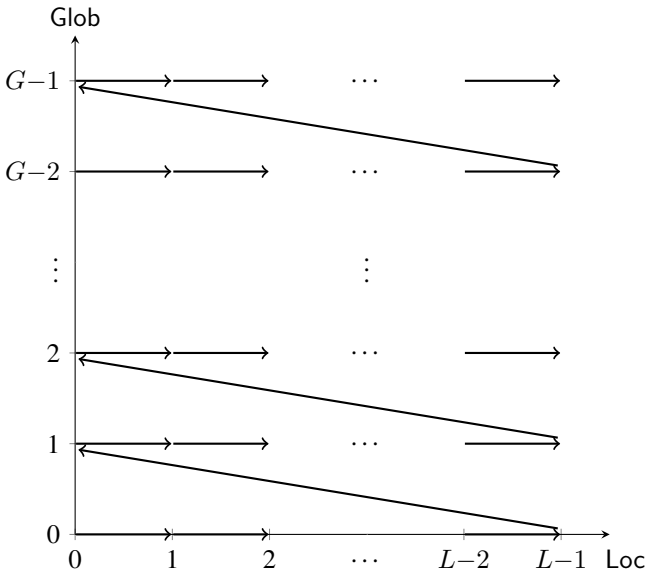
Proof of Thm. IV.1.1

The cases $L=1$ or $G=1$ have been treated in Lems. D.1 and D.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

$$\{\, ((g,l),(g,l+1)) \mid g < G \wedge l+1 < L \,\} \quad (4)$$
$$\cup\, \{\, ((g,L-1),(g+1,0)) \mid g+1 < G \,\}. \quad (5)$$

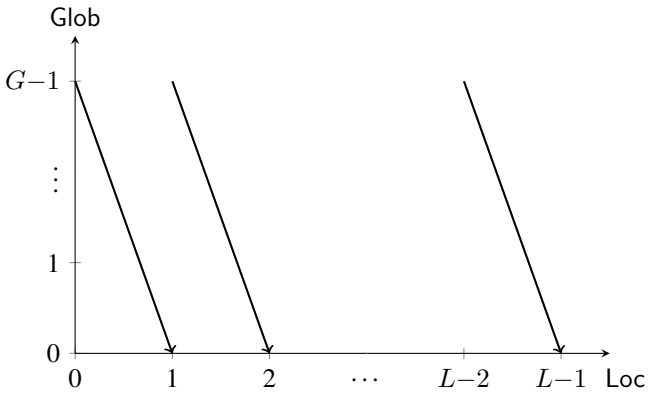
For example, for $G\geqslant 5$ and $L\geqslant 5$ these thread transitions can be visualized as follows:



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

$$\{((G-1,l),(0,l+1)) \mid l+1 < L\}. \quad (6)$$

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 $(\mathbb{N}_{\geqslant 0} \cap [0,(G-1)L]) = \big(\mathbb{N}_{\geqslant 0} \cap [0,L]\big) \,\dot\cup\, \big(\mathbb{N}_{\geqslant 0} \cap [L,(G-1)L]\big)$. 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

$$D_{k((G-1)L+1)+m} \stackrel{\text{def}}{=}$$
$$\left\{(0,l) \in \mathsf{State} \;\middle|\; l_0 = m \wedge \sum_{t=1}^{n-1} l_t = k\right\} \quad (7)$$
$$\dot\cup \left\{(G-1,l) \in \mathsf{State} \;\middle|\; l_0 = m+1 \wedge \sum_{t=1}^{n-1} l_t = k-1\right\}, \quad (8)$$

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

$$D_{k((G-1)L+1)+m} \overset{\text{def}}{=\!=} \left\{ \left(\left\lfloor \frac{m}{L} \right\rfloor, l \right) \in \mathsf{State} \,\middle|\, l_0 = (m \bmod L) \wedge \sum_{t=1}^{n-1} l_t = k \right\}. \quad (9)$$

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 (7) or (8).

- 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 (9), $g = \lfloor \frac{m'}{L} \rfloor$ and $l_0 = (m' \bmod L)$. From $m' \geqslant L$ we obtain $g{>}0$. By (8), $g = G{-}1$ 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 (9), $g = \lfloor \frac{m}{L} \rfloor$ and $l_0 = (m \bmod L)$. From $m \geqslant L$ we obtain $g{>}0$. By (8), $g = G{-}1$ 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 (9), $\lfloor \frac{m}{L} \rfloor = g = \lfloor \frac{m'}{L} \rfloor$, $(m \bmod L) = l_0 = (m' \bmod L)$, and $k = \sum_{t=1}^{n-1} l_t = k'$. Then, $m = L\lfloor \frac{m}{L} \rfloor + (m \bmod L) = L\lfloor \frac{m'}{L} \rfloor + (m' \bmod L) = m'$ 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:

$$\forall\, i, j \in \mathbb{N}_{\geqslant 0}\colon \quad D_i \cap D_j \;\neq\; \emptyset \;\;\Rightarrow\;\; i = j\,. \quad (10)$$

Now, consider the map

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

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 (10) we obtain $(g, l) \notin D_j$ for all $j{<}i$. The induction hypothesis implies

$$\forall\, j < i\colon \quad (g, l) \notin \mathsf{depth}^{-1}(\{j\})\,. \quad (11)$$

As the next step, we are going to show that 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 (7), 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 $G{-}1 = \lfloor \frac{(G-1)L}{L} \rfloor$, $\hat l_0 = 0 = (((G{-}1)L) \bmod L)$, and $\sum_{t=1}^{n-1} \hat l_t = k{-}1$. By (9), $(G{-}1, \hat l) \in D_{(k-1)((G-1)L+1)+(G-1)L} = D_{i-1}$. By the induction hypothesis, $\mathsf{depth}(G{-}1, \hat l) = i{-}1$. By (6), $(G{-}1, \hat l) \longrightarrow (0, l)$.

Case $m{\geqslant}1$. Let $\hat l = l[0 \mapsto m{-}1]$. From (7) 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 (4), $(0, \hat l) \longrightarrow (0, l)$.

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

Case $(g, l) \in D_i$ due to (8), 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 $G{-}1 = \lfloor \frac{(G-1)L}{L} \rfloor$, $\hat l_0 = 0 = \Big(\big((G{-}1)L\big) \bmod L\Big)$, and $\sum_{t=1}^{n-1} \hat l_t = k{-}1$. According to (9), $(G{-}1, \hat l) \in D_{(k-1)((G-1)L+1)+(G-1)L} = D_{i-1}$.

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 (8), $(G{-}1, \hat l) \in D_{k((G-1)L+1)+m-1} = D_{i-1}$.

The induction hypothesis implies $\mathsf{depth}(G{-}1, \hat l) = i{-}1$. By (4), $(G{-}1, \hat l) \longrightarrow (g, l)$. Therefore, $\mathsf{depth}(g, l) \leqslant i$.

Case $(g, l) \in D_i$ due to (9), i.e., $L{\leqslant}m \wedge g = \lfloor \frac{m}{L} \rfloor \wedge l_0 = (m \bmod L) \wedge \sum_{t=1}^{n-1} l_t = k$. 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 $(\hat g, \hat l) = (0, \hat l) \overset{\text{by (7)}}{\in} D_{k((G-1)L+1)+L-1} = D_{k((G-1)L+1)+gL-1}$.

Case $\hat g{\geqslant}1$. Note that $\hat g = g{+}\lfloor -\frac{1}{L} \rfloor = \lfloor g{-}\frac{1}{L} \rfloor = \lfloor \frac{gL-1}{L} \rfloor$, $\hat l_0 = L{-}1 = ((g{-}1)L + L - 1) \bmod L) = ((gL - 1) \bmod L)$, $\sum_{t=1}^{n-1} \hat l_t = k$, and $gL - 1 = (\hat g{+}1)L - 1 \geqslant 2L - 1 \geqslant L$. By (9), $(\hat g, \hat l) \in D_{k((G-1)L+1)+gL-1}$.

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 (5), $(\hat g, \hat l) \longrightarrow (g, l)$.

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 (9), $(g,\hat{l}) \in D_{k((G-1)L+1)+m-1} = D_{i-1}$. The induction hypothesis implies $\mathsf{depth}(g,\hat{l}) = i-1$. By (4), $(g,\hat{l}) \longrightarrow (g,l)$.

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

We have shown $\mathsf{depth}(g,l) \leqslant i$. By (11), $\mathsf{depth}(g,l) = i$.

"$\supseteq$". 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 (9), $\hat{g} = G{-}1$, $\hat{l}_0 = 0$, 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 (4), 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 (8), $(g,l) \in D_{k((G-1)L+1)}$.
- Case (5), 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 (6), 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 (7), $(g,l) \in D_{k((G-1)L+1)}$.

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 (7) or (8):

- Case (7), 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 (4), 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 (7), $(g,l) \in D_{k((G-1)L+1)+m}$.
  - Case (5), 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 (6), 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 (8), 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 (4), 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 (8), $(g,l) \in D_{k((G-1)L+1)+m}$.
  - Case (5), 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 (6), 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 (7), $(g,l) \in D_{k((G-1)L+1)+m}$.

Case $m{=}L$. We obtain $(\hat{g},\hat{l}) \in D_{k((G-1)L+1)+m-1}$ where, due to $L{\geqslant}2$, $0 \leqslant m{-}1 < L$. Moreover, $(m{-}1)+1 = L > \hat{l}_0$. By (8), $(G{-}1,\hat{l}) \notin D_{k((G-1)L+1)+m-1}$, and so $\hat{g} \neq G{-}1$. By (7), $\hat{g} = 0$, $\hat{l}_0 = L{-}1$, 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 (4), i.e., $\hat{t}{=}0 \wedge \hat{g} = g \wedge \hat{l}_0{+}1 = l_0$. Then $l_0 = L$. $\lightning$
- Case (5), i.e., $\hat{t}{=}0 \wedge \hat{g}{+}1 = g \wedge \hat{l}_0 = L{-}1 \wedge l_0 = 0$. Then, $g = 1 = \lfloor \frac{m}{L} \rfloor$, $l_0 = 0 = (m \bmod L)$, and $\sum_{t=1}^{n-1} l_t = k$. By (9), $(g,l) \in D_{k((G-1)L+1)+m}$.
- Case (6), 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, $\hat{g} = \lfloor \frac{m-1}{L} \rfloor$, $\hat{l}_0 = ((m{-}1) \bmod L)$, 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 (4), i.e., $\hat{t}{=}0 \wedge \hat{g} = g \wedge \hat{l}_0{+}1 = l_0$. Note that $m = (m{-}1){+}1 = \Big(L\lfloor \frac{m-1}{L} \rfloor + \big((m{-}1) \bmod L\big)\Big) + 1 = L\hat{g} + \hat{l}_0 + 1 = Lg + l_0$. From $l_0{<}L$ and the uniqueness of the quotient with the remainder we obtain $g = \lfloor \frac{m}{L} \rfloor$ and $l_0 = (m \bmod L)$. Finally, $\sum_{t=1}^{n-1} l_t = k$. By (9), $(g,l) \in D_{k((G+1)L+1)+m}$.
- Case (5), i.e., $\hat{t}{=}0 \wedge \hat{g}{+}1 = g \wedge \hat{l}_0 = L{-}1 \wedge l_0 = 0$. Then, $m = (m{-}1) + 1 = \Big(L\lfloor \frac{m-1}{L} \rfloor + \big((m{-}1) \bmod L\big)\Big) + 1 = (L\hat{g} + \hat{l}_0) + 1 = L\hat{g} + L - 1 + 1 = L(\hat{g}{+}1) = Lg$. The uniqueness of the quotient with the remainder implies $\lfloor \frac{m}{L} \rfloor = g$ and $(m \bmod L) = 0 = l_0$. Finally, $\sum_{t=1}^{n-1} l_t = k$. By (9), $(g,l) \in D_{k((G+1)L+1)+m}$.
- Case (6), i.e., $\hat{t}{\geqslant}1 \wedge \hat{g} = G{-}1 \wedge \hat{l}_{\hat{t}}{+}1 = l_{\hat{t}} \wedge g{=}0$. Then $G{-}1 = \lfloor \frac{m-1}{L} \rfloor$. Since $m{-}1 < (G{-}1)L$, we must have $\frac{m-1}{L} < G{-}1$, and therefore $\lfloor \frac{m-1}{L} \rfloor < G{-}1$. $\lightning$

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

$$\hat{d} \stackrel{\text{def}}{=\!=} \big((n{-}1)(L{-}1)+1\big)\big((G{-}1)L+1\big) + L - 2 \quad (12)$$

we have $(G{-}1, n \times \{L{-}1\}) \in D_{\hat{d}}$ by (8). Therefore, $\hat{d} = \mathsf{depth}\big(G{-}1, n \times \{L{-}1\}\big) = \mathsf{d}\Big(\big(0, n \times \{0\}\big)\Big)$,

$\big(G{-}1, n \times \{L{-}1\}\big)\big) \leqslant \mathsf{diamax}(n)$. Note that $\hat{d} = (n{-}1)(L{-}1)((G{-}1)L+1) + (G{-}1)L + 1 + L - 2 = (n{-}1)(L{-}1)(GL{-}L{+}1){+}GL{-}L{+}L{-}1 = n(L{-}1)(GL{-}L+1) - (L{-}1)(GL-L+1) + GL - 1 = n(L{-}1)(GL-L+1) - (GL^2 - L^2 + L - GL + L - 1) + GL - 1 = n(L{-}1)(GL-L+1) - GL^2 + L^2 + GL - 2L + 1 + GL - 1 = n(L{-}1)(GL - L + 1) - GL^2 + L^2 + 2GL - 2L = n(L{-}1)(GL - L + 1) + (-L(G{-}1) + 2(G{-}1))L = n(L{-}1)(GL - L + 1) + (2{-}L)(G{-}1)L$. ■

Note D.3. 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 distance in the transition graph of this program.

To this end, we first claim that, in the context of the above proof, each successors of a 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 $\forall\, \hat{\jmath} \in n\backslash\{j\}\colon l_{\hat{\jmath}} = l'_{\hat{\jmath}}$. We consider three cases concerning how $(g,l) \in D_i$ originated:

Case (7), 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 (4), i.e., $g' = g \wedge l'_0 = l_0{+}1$. Then, $g' = 0$ and $m{+}1 = l'_0 < L \leqslant (G{-}1)L$. By (7), $(g',l') \in D_{k((G-1)L+1)+m+1}$.
- Case (5), 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 (9), $(g',l') \in D_{k((G-1)L+1)+m+1}$.

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

Case (8), i.e., $m{<}L \wedge g = G{-}1 \wedge l_0 = m{+}1 \wedge \sum_{t=1}^{n-1} l_t = k{-}1$. Since $g{+}1 = G$, the transition (5) 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 (4), 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 (8).
- Case (6), 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 (7).

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

Case (9), 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 (4), 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 (9), $(g',l') \in D_{k((G-1)L+1)+m+1}$.
  - 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 (8), $(g',l') \in D_{k'((G-1)L+1)+m'}$. 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 (5), 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 $m' = m{+}1 = \Big(L\lfloor \frac{m}{L} \rfloor + (m \bmod L)\Big) + 1 = (Lg + l_0) + 1 = Lg + L - 1 + 1 = Lg + L = L(g{+}1) = Lg'$. Therefore, $g' = \lfloor \frac{m'}{L} \rfloor$ 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 (9), $(g',l') \in D_{k((G-1)P+1)+m'} = D_{i+1}$.
- Case (6), 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 $\frac{m}{L} < G{-}1 = g = \lfloor \frac{m}{L} \rfloor \leqslant \frac{m}{L}$, ↯. 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 (7), $(g',l') \in D_{k'((G-1)L+1)+m'} =$ [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:

$$\forall\, i \in \mathbb{N}_{\geqslant 0},\ s \in D_i,\ s' \in \mathsf{State}\colon\ s \longrightarrow s' \ \Rightarrow\ s' \in D_{i+1}\,. \tag{13}$$

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 (7).

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 (9).

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 (9).

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 (8).

We have shown

$$\mathsf{State} \subseteq \bigcup_{i\in\mathbb{N}_{\geqslant 0}} D_i\,. \tag{14}$$

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 (7) and (8),
- $k = (L-1)(n-1)+1$ and $m \geqslant L$ according to (9),
- $k \geqslant (L-1)(n-1)+2$ according to (7), (8), and (9).

Therefore:

$$D_i{=}\emptyset \text{ for all } i \geqslant ((L{-}1)(n{-}1){+}1)((G{-}1)L{+}1){+}L{-}1\,. \tag{15}$$

Now, let $s, s' \in \mathsf{State}$ be arbitrary such that $s \longrightarrow^* s'$. By (14), there are some $i,j \in \mathbb{N}_{\geqslant 0}$ such that $s \in D_i$ and $s' \in D_j$. By (12) and (15), $i,j \leqslant \hat{d}$. By (13), $\mathsf{d}(s,s') = j{-}i \leqslant j \leqslant \hat{d}$. Since s, s' were arbitrary, the diameter of the program is at most $\hat{d}$. Since the proof of Thm. IV.1.1 provided us with two states at this distance, $\hat{d}$ is the exact value of the diameter of the program. □

Appendix E.
Proofs of claims from § IV.2.1

Proof of Lem. IV.2.1.3

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

Let $i{<}n$ be arbitrary. Let $j \stackrel{\text{def}}{=} \varphi^{-1}(i)$. 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 $i{<}n$. ■

Proof of Lem. IV.2.1.4

We show:

"$\approx$ is reflexive on State": Due to the $\sim$-invariance of the identity $\mathsf{id}_n\colon n \hookrightarrow\!\!\!\!\rightarrow n$.

"$\approx$ is symmetric": Let $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$. Then $\hat{g}{=}\check{g}$. Choose some $\sim$-invariant $\varphi\colon n{\hookrightarrow\!\!\!\!\rightarrow}n$ such that $\forall\, i{<}n\colon \hat{l}_i = \check{l}_{\varphi(i)}$. Lem. IV.2.1.3 implies that φ^{-1} is $\sim$-invariant. Also, $\forall\, i{<}n\colon \hat{l}_{\varphi^{-1}(\varphi(i))} = \check{l}_{\varphi(i)}$. The surjectivity of φ implies that $\forall\, j{<}n\colon \hat{l}_{\varphi^{-1}(j)} = \check{l}_j$. Summarizing, $(\check{g},\check{l}) \approx (\hat{g},\hat{l})$.

"$\approx$ is transitive": Let $(\hat{g},\hat{l}){\approx}(\check{g},\check{l}){\approx}(\overline{g},\overline{l})$. Then $\hat{g}{=}\check{g}{=}\overline{g}$ and some $\sim$-invariant $\varphi,\psi\colon n{\hookrightarrow\!\!\!\!\rightarrow}n$ exist such that $\forall\, i{<}n\colon \hat{l}_i{=}\check{l}_{\varphi(i)}$ and $\forall\, j{<}n\colon \check{l}_j{=}\overline{l}_{\psi(j)}$. Then $\psi{\circ}\varphi$ is $\sim$-invariant and $\forall\, i{<}n\colon \hat{l}_i \sim \overline{l}_{\psi(\varphi(i))}$. Therefore, $(\hat{g},\hat{l}) \approx (\overline{g},\overline{l})$. ■

Proof of Lem. IV.2.1.6

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

"$\Rightarrow$": We assume the left-hand side $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$. Then $\hat{g}{=}\check{g}$, and some $\sim$-invariant $\zeta\colon n \hookrightarrow\!\!\!\!\rightarrow n$ exists such that

$$\forall\, i < n\colon\quad \hat{l}_i = \check{l}_{\zeta(i)}\,. \tag{16}$$

The $\sim$-invariance of ζ means

$$\forall\, i < n\colon\quad i \sim \zeta(i)\,. \tag{17}$$

Let $a,b \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. We claim that the map $\bar{\zeta}\ :\ \{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \hat{l}_t{=}b\} \to \{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \check{l}_t{=}b\},\ t \mapsto \zeta(t)$ is well defined and a bijection. We prove this claim now:

"$\bar{\zeta}$ is well defined": Let $t{\in}n$ be such that $l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \hat{l}_t{=}b$. From (16) we obtain $\check{l}_{\zeta(t)}{=}b$. From (17) we obtain $l_{\zeta(t)}{=}a$ and ${\to_{\zeta(t)}}\backslash D = {\rightsquigarrow}$. Therefore, $\zeta(t) \in \{t{<}n \mid l_t{=}a{\wedge}{\to_t}\backslash D = {\rightsquigarrow}{\wedge}\check{l}_t{=}b\}$.

"$\bar{\zeta}$ is injective": Follows from the injectivity of ζ.

"$\bar{\zeta}$ is surjective": Let $t{\in}n$ be given such that $l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \check{l}_t = b$. Then $\zeta^{-1}(t) \in n$, and $l_{\zeta^{-1}(t)}$ = [due to (17)] $l_{\zeta(\zeta^{-1}(t))} = l_t = a$, and ${\to_{\zeta^{-1}(t)}}\backslash D$ = [due to (17)] ${\to_{\zeta(\zeta^{-1}(t))}}\backslash D = {\to_t}\backslash D = {\rightsquigarrow}$, and $\hat{l}_{\zeta^{-1}(t)}$ = [due to (16)] $\check{l}_{\zeta(\zeta^{-1}(t))} = \check{l}_t = b$. So $\zeta^{-1}(t) \in \mathsf{dom}\,\bar{\zeta}$. Also, $\bar{\zeta}(\zeta^{-1}(t)) = t$.

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

"$\Leftarrow$": We assume the right-hand side, i.e., $\hat{g}{=}\check{g}$ and $\forall\, a,b \in \mathsf{Loc},\ {\rightsquigarrow}\in E\colon \big|\{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \hat{l}_t{=}b\}\big| = \big|\{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \check{l}_t{=}b\}\big|$. Then, for each triple $(a,{\rightsquigarrow},b) \in \mathsf{Loc}{\times}E{\times}\mathsf{Loc}$ there is a bijection $\gamma_{a,\rightsquigarrow,b}\colon \{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \hat{l}_t{=}b\} \hookrightarrow \{t{<}n \mid l_t{=}a \wedge {\to_t}\backslash D = {\rightsquigarrow} \wedge \check{l}_t{=}b\}$. 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

$$\zeta \stackrel{\text{def}}{=} \bigcup_{a,b\in\mathsf{Loc},\ {\rightsquigarrow}\in E} \gamma_{a,\rightsquigarrow,b}\,.$$

Then ζ is a permutation of n. If $i{<}n$ is arbitrary, then $\zeta(i) = \gamma_{l_i,{\to_i}\backslash D,\hat{l}_i}(i)$, and so $l_i = l_{\gamma_{l_i,{\to_i}\backslash D,\hat{l}_i}(i)} = l_{\zeta(i)}$, and ${\to_i}\backslash D = {\to_{\gamma_{l_i,{\to_i}\backslash D,\hat{l}_i}(i)}}\backslash D = {\to_{\zeta(i)}}\backslash D$, and $\hat{l}_i = \check{l}_{\gamma_{l_i,{\to_i}\backslash D,\hat{l}_i}(i)} = \check{l}_{\zeta(i)}$. So ζ is $\sim$-invariant, and $\forall\, i{<}n\colon \hat{l}_i = \check{l}_{\zeta(i)}$. Thus, $(\hat{g},\hat{l}) \approx (\check{g},\check{l})$. ■

Proof of Lem. IV.2.1.7

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

So let an arbitrary $m \in \mathbb{N}_{\geqslant 0} \cup \{\infty\}$ 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 $\mathsf{d}\big((g,l),s\big) = 0$. 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 $\forall\, i{<}n\colon l_i{=}\check{l}_{\varphi(i)}$. The definition of $\sim$ implies $\forall\, i{<}n\colon l_i{=}l_{\varphi(i)}$. Therefore, $\forall\, i{<}n\colon l_{\varphi(i)}{=}\check{l}_{\varphi(i)}$. 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 $\mathsf{d}\big((g,l),s'\big) = 0$. 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 $\forall\, i{<}n\colon \check{l}_i{=}l_{\varphi(i)}$. The definition of $\sim$ implies $\forall\, i{<}n\colon l_i{=}l_{\varphi(i)}$. Therefore, $\forall\, i{<}n\colon \check{l}_i{=}l_i$. 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 $\varphi\colon n\hookrightarrow n$ such that $\forall\, i<n\colon \check{l}_i=\hat{l}_{\varphi(i)}$.

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

$$\mathsf{d}\big((g,l),(\check{g}',\check{l}')\big) \;=\; m-1\,. \tag{18}$$

Let $\hat{l}' \;=\; \lambda\, i{\in}n.\, \check{l}'_{\varphi^{-1}(i)}$. Then $\forall\, j<n\colon \hat{l}'_{\varphi(j)} = \check{l}'_{\varphi^{-1}(\varphi(j))} = \check{l}'_j$. So $(\check{g}',\check{l}') \approx (\check{g}',\hat{l}')$. The induction hypothesis implies $\mathsf{d}\Big((g,l),(\check{g}',\hat{l}')\Big) = m{-}1$. Choose some $t{\in}n$ such that $(\check{g}',\check{l}'_t) \to_t (\check{g},\check{l}_t) \wedge \forall\, \bar{t} \in n\backslash\{t\}\colon \check{l}'_{\bar{t}} = \check{l}_{\bar{t}}$. Due to (18) and the distance condition in this branch of the case split, $(\check{g}',\check{l}'_t) \neq (\check{g},\check{l}_t)$. Since $t \sim \varphi(t)$, we have $\to_t\backslash D \;=\; \to_{\varphi(t)}\backslash D$. Thus, $(\check{g}',\check{l}'_t) \to_{\varphi(t)} (\check{g},\check{l}_t)$. Knowing in addition that $\hat{l}'_{\varphi(t)}{=}\check{l}'_t$ and $\check{l}_t{=}\hat{l}_{\varphi(t)}$, we get $(\check{g}',\hat{l}'_{\varphi(t)}) \to_{\varphi(t)} (\check{g},\hat{l}_{\varphi(t)})$. Moreover, $\forall\, \bar{t} \in n\backslash\{\varphi(t)\}\colon\; \hat{l}'_{\bar{t}} = \check{l}'_{\varphi^{-1}(\bar{t})} =$ [from $\bar{t} \neq \varphi(t)$ we get $\varphi^{-1}(\bar{t}) \neq t$] $\check{l}_{\varphi^{-1}(\bar{t})} = \hat{l}_{\bar{t}}$. Combining, $(\check{g}',\hat{l}') \longrightarrow (\check{g},\hat{l}) = (\hat{g},\hat{l})$. Thus, $\mathsf{d}\big((g,l),(\hat{g},\hat{l})\big) \leqslant m = \min\Big\{\mathsf{d}\big((g,l),(\check{g},\check{l})\big), \mathsf{d}\big((g,l),(\hat{g},\hat{l})\big)\Big\} \leqslant \mathsf{d}\big((g,l),(\hat{g},\hat{l})\big)$. Therefore, $\mathsf{d}\big((g,l),(\hat{g},\hat{l})\big) = m = \mathsf{d}\big((g,l),(\check{g},\check{l})\big)$.

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

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

Let $\check{l}' = \lambda\, i{\in}n.\, \hat{l}'_{\varphi(i)}$. Due to Lem. IV.2.1.3, φ^{-1} is $\sim$-invariant. Moreover, $\forall\, j<n\colon \check{l}'_{\varphi^{-1}(j)} = \hat{l}'_{\varphi(\varphi^{-1}(j))} = \hat{l}'_j$. So $(\hat{g}',\hat{l}') \approx (\hat{g}',\check{l}')$. The induction hypothesis implies $\mathsf{d}\big((g,l),(\hat{g}',\check{l}')\big) = m{-}1$. Choose some $t{\in}n$ such that $(\hat{g}',\hat{l}'_t) \to_t (\hat{g},\hat{l}_t)$ and $\forall\, \bar{t} \in n\backslash\{t\}\colon \hat{l}'_{\bar{t}} = \hat{l}_{\bar{t}}$. Due to (19) and the distance condition in this branch of the case split, $(\hat{g}',\hat{l}'_t) \neq (\hat{g},\hat{l}_t)$. Note that $t \sim \varphi^{-1}(t)$; thus, $\to_t\backslash D = \to_{\varphi^{-1}(t)}\backslash D$. Thus, $(\hat{g}',\hat{l}'_t) \to_{\varphi^{-1}(t)} (\hat{g},\hat{l}_t)$. Knowing in addition that $\check{l}'_{\varphi^{-1}(t)} = \hat{l}'_t$ and $\hat{l}_t = \hat{l}_{\varphi(\varphi^{-1}(t))} = \check{l}_{\varphi^{-1}(t)}$, we get $(\hat{g}',\check{l}'_{\varphi^{-1}(t)}) \to_{\varphi^{-1}(t)} (\hat{g},\check{l}_{\varphi^{-1}(t)})$. Moreover, $\forall\;\; \bar{t} \in n \setminus \{\varphi^{-1}(t)\}\colon\; \check{l}'_{\bar{t}} = \hat{l}'_{\varphi(\bar{t})} =$ [since $\bar{t} \neq \varphi^{-1}(t)$, we have $\varphi(\bar{t}) \neq t$] $\hat{l}_{\varphi(\bar{t})} = \check{l}_{\bar{t}}$. Combining, $(\hat{g}',\check{l}') \longrightarrow (\hat{g},\check{l}) = (\check{g},\check{l})$. Thus, $\mathsf{d}\big((g,l),(\check{g},\check{l})\big) \leqslant m = \min\Big\{\mathsf{d}\big((g,l),(\check{g},\check{l})\big), \mathsf{d}\big((g,l),(\hat{g},\hat{l})\big)\Big\} \leqslant \mathsf{d}\big((g,l),(\check{g},\check{l})\big)$. Therefore, $\mathsf{d}\big((g,l),(\check{g},\check{l})\big) = m = \mathsf{d}\big((g,l),(\hat{g},\hat{l})\big)$.

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

Proof of Lem. IV.2.1.8

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 $k = \mathsf{d}\big((g,l),s\big)$. Note that $\forall\, i{\leqslant}k\colon \mathsf{d}\big((g,l),\sigma^{[i]}\big) = i$. Lem. IV.2.1.7 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, $\big|\mathsf{State}/\!\approx\big| \geqslant \mathsf{d}\big((g,l),s\big) + 1 > \mathsf{d}\big((g,l),s\big)$. ■

Proof of Lem. IV.2.1.9

Recall that V is the set of all maps

$$f\colon (\mathsf{Loc}{\times}E) \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}$$

such that for all $a{\in}\mathsf{Loc}$ and all $\rightsquigarrow\, \in E$ we have

$$\|f(a,\rightsquigarrow)\|_1 = \big|\{t{<}n \mid l_t{=}a \wedge \to_t\backslash D = \rightsquigarrow\}\big|\,. \tag{20}$$

Let $\varphi\colon \mathsf{Loc}^n \to (\mathsf{Loc}{\times}E) \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}$, $\hat{l} \mapsto \lambda\,(a,\rightsquigarrow) \in \mathsf{Loc}{\times}E.\,\lambda\, b{\in}\mathsf{Loc}.\,\big|\{t{<}n \mid l_t{=}a \wedge \to_t\backslash D = \rightsquigarrow \wedge\, \hat{l}_t{=}b\}\big|$.

First, we show that $\mathsf{img}\,\varphi \overset{!}{\subseteq} V$. For that, let $\hat{l}{\in}\mathsf{Loc}^n$ be arbitrary; it suffices to show that $\varphi(\hat{l}) \overset{!}{\in} V$. Note that $\varphi(\hat{l}) \in \big((\mathsf{Loc}{\times}E) \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}\big)$. Now, let $a \in \mathsf{Loc}$ and $\rightsquigarrow\, \in E$. Then, $\|\varphi(\hat{l})(a,\rightsquigarrow)\|_1 = \sum_{b\in\mathsf{Loc}} \big|\{t{<}n \mid l_t{=}a \wedge \to_t\backslash D = \rightsquigarrow \wedge\, \hat{l}_t{=}b\}\big| = \big|\{t{<}n \mid l_t{=}a \wedge \to_t\backslash D = \rightsquigarrow\}\big|$. So $\varphi(\hat{l}) \in V$. Since $\hat{l}$ was arbitrary,

$$\mathsf{img}\,\varphi \;\subseteq\; V. \tag{21}$$

Next, we show that $\mathsf{img}\,\varphi \overset{!}{\supseteq} V$. As a preparation step, we are going to enumerate the members of Loc by a bijection $enlo\colon L \hookrightarrow\!\!\!\!\rightarrow \mathsf{Loc}$. (The term $enlo$ means "*en*umerate *lo*cals.") The members of each equivalence class $c \in {}^{n}\!/\!_{\sim}$ will be enumerated by a bijection $encl(c)\colon c \hookrightarrow\!\!\!\!\rightarrow |c|$. (The term $encl$ means "*en*umerate *cl*ass.")

Now, let $f{\in}V$ be arbitrary. For each $t{\in}n$, the set $\big\{\bar{m}{<}L \;\big|\; \sum_{m<\bar{m}} f(l_t,\to_t\backslash D)(enlo(m)) \leqslant encl([t]_\sim)(t)\big\}$ is finite and nonempty (as it contains zero). Thus, the map

$$\hat{l} \;=\; \lambda\, t \in n.\, enlo\Big(\max\Big\{\bar{m} < L \;\Big|\; \sum_{m<\bar{m}} f(l_t,\to_t\backslash D)(enlo(m)) \leqslant encl([t]_\sim)(t)\Big\}\Big) \tag{22}$$

is well defined. We are going to prove that $\hat{l}$ is the preimage of f under φ.

First of all, notice that $\hat{l} \in \mathsf{dom}\,\varphi$.

To show $\varphi(\hat{l}) \overset{!}{=} f$, let $a \in \mathsf{Loc}$ and $\rightsquigarrow\, \in E$ be arbitrary. We are going to prove that $\varphi(\hat{l})(a,\rightsquigarrow) \overset{!}{=} f(a,\rightsquigarrow)$. Let $g = f(a,\rightsquigarrow) \circ enlo$ and

$$c \;=\; \{t < n \;\mid\; l_t = a \wedge \to_t\backslash D = \rightsquigarrow\} \tag{23}$$

(which is an equivalence class with respect to thread confusion or empty). Note that $\sum_{m<L} g(m) = \|g\|_1 = \|f(a,\rightsquigarrow) \circ enlo\|_1 =$ [since $enlo$ is a bijection and by the definition of the 1-norm] $\|f(a,\rightsquigarrow)\|_1 \overset{\text{by (20)}}{=} \big|\{t{<}n \mid l_t{=}a \wedge \to_t\backslash D = \rightsquigarrow\}\big| \overset{\text{by (23)}}{=} |c|$. In summary,

$$\sum_{m<L} g(m) = |c|\,. \tag{24}$$

Let $h\colon c \hookrightarrow\!\!\!\!\rightarrow |c|$ be the map

$$h \;=\; \begin{cases} encl(c), & \text{if } c \neq \emptyset, \\ \text{the empty map}, & \text{if } c = \emptyset. \end{cases}$$

Let $b \in \mathsf{Loc}$ be arbitrary and $\bar{b} = enlo^{-1}(b)$. Before proving $\varphi(\hat{l})(a,\rightsquigarrow)(b) \overset{!}{=} f(a,\rightsquigarrow)(b)$, we will show two auxiliary

statements. First, from $\sum_{m\leqslant\bar b} g(m) \leqslant \sum_{m<L} g(m) \overset{\text{by (24)}}{=} |c|$ we obtain

$$\Big[\sum_{m<\bar b} g(m), \sum_{m\leqslant\bar b} g(m)\Big[\;\subseteq\; |c|\,. \tag{25}$$

Second, we show

$$\bar b = \max\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\} \;\overset{!}{\Leftrightarrow}$$
$$\sum_{m<\bar b} g(m) \leqslant h(t) < \sum_{m\leqslant\bar b} g(m) \qquad (t\in c)\,.$$

We prove the two directions of this bi-implication for an arbitrary $t{\in}c$ separately:

"$\Rightarrow$": We assume $\bar b = \max\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\}$. Then, this set contains $\bar b$. So $\sum_{m<\bar b} g(m) \leqslant h(t)$. Now we show $h(t) \overset{!}{<} \sum_{m\leqslant\bar b} g(m)$:

Case $\bar b{+}1 < L$: Since $\bar b$ is the maximum of $\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\}$, this set doesn't contain $\bar b{+}1$. Since $\bar b{+}1 < L$, we must have $h(t) < \sum_{m<\bar b+1} g(m) = \sum_{m\leqslant\bar b} g(m)$.

Case $\bar b{+}1 = L$: Then $h(t) < |c| =$ [using (24)] $\sum_{m<L} g(m)$ $=$ [since $\bar b{+}1 = L$] $\sum_{m\leqslant\bar b} g(m)$.

"$\Leftarrow$": We assume $\sum_{m<\bar b} g(m) \leqslant h(t) < \sum_{m\leqslant\bar b} g(m)$. Then, $\bar b$ belongs to the set over which the maximum is taken, so, $\bar b \leqslant \max\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\}$.

Case $\bar b{+}1 < L$: Let $\bar m{<}L$ be arbitrary such that $\sum_{m<\bar m} g(m) \leqslant h(t)$. Since $h(t) < \sum_{m\leqslant\bar b} g(m) = \sum_{m<\bar b+1} g(m)$, we have $\sum_{m<\bar m} g(m) < \sum_{m<\bar b+1} g(m)$ by transitivity. Since all members of these two sums are nonnegative, we must have $\bar m < \bar b{+}1$, i.e., $\bar m \leqslant \bar b$. Since $\bar m$ was arbitrary, $\max\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\} \leqslant \bar b$.

Case $\bar b{+}1 = L$: Then, $\bar b$ is already the largest number below L, in particular, greater than or equal to any number $\bar m{<}L$ satisfying $\sum_{m<\bar m} g(m) \leqslant h(t)$.

We have shown

$$\bar b \;=\; \max\big\{\bar m{<}L \;\big|\; \sum_{m<\bar m} g(m) \leqslant h(t)\big\} \;\Leftrightarrow \tag{26}$$
$$\sum_{m<\bar b} g(m) \leqslant h(t) < \sum_{m\leqslant\bar b} g(m) \qquad (t\in c)\,.$$

After these preparations, we obtain the following chain of equations:

$$\varphi(\hat l)(a,\rightsquigarrow)(b) \;=\; \big|\{t{<}n \;\mid\; l_t{=}a \wedge {\to_t}\backslash D \;=\; \rightsquigarrow \wedge\; \hat l_t{=}b\}\big|$$
$$\overset{\text{by (23)}}{=} \big|\{t{\in}c \;\mid\; \hat l_t{=}b\}\big| \overset{\text{by (22)}}{=} \Big|\Big\{t{\in}c \;\Big|\; \bar b \;=\; \max\Big\{\bar m{<}L \;\Big|\; \sum_{m<\bar m} f\big(l_t, {\to_t}\backslash D\big)\big(enlo(m)\big) \;\leqslant\; encl\big([t]_\sim\big)\big(t\big)\Big\}\Big\}\Big| \overset{\text{by (23)}}{=} \Big|\Big\{t{\in}c$$
$$\Big|\; \bar b \;=\; \max\Big\{\bar m{<}L \;\Big|\; \sum_{m<\bar m} f\big(a,\rightsquigarrow\big)\big(enlo(m)\big) \;\leqslant\; encl(c)(t)\Big\}\Big\}\Big|$$
$$= \Big|\Big\{t{\in}c \;\Big|\; \bar b \;=\; \max\big\{\bar m{<}L \;\mid\; \sum_{m<\bar m} g(m) \;\leqslant\; h(t)\big\}\Big\}\Big| \overset{\text{by (26)}}{=}$$
$$\Big|\big\{t{\in}c \;\mid\; \sum_{m<\bar b} g(m) \;\leqslant\; h(t) \;<\; \sum_{m\leqslant\bar b} g(m)\big\}\Big| \;=\; \Big|h^{-1}\Big(\big[\sum_{m<\bar b} g(m), \sum_{m\leqslant\bar b} g(m)\big[\Big)\Big| \;=$$

[using (25) and the fact that $h\colon c \hookrightarrow\!\!\!\!\rightarrow |c|$ is a bijection] $\Big|\big[\sum_{m<\bar b} g(m), \sum_{m\leqslant\bar b} g(m)\big[\Big| = \sum_{m\leqslant\bar b} g(m) - \sum_{m<\bar b} g(m) = g(\bar b) = f(a,\rightsquigarrow)(enlo(\bar b)) = f(a,\rightsquigarrow)(b)$.

Since b was arbitrary, $\varphi(\hat l)(a,\rightsquigarrow) = f(a,\rightsquigarrow)$.

Since a and $\rightsquigarrow$ were arbitrary, $\varphi(\hat l) = f$.

So $f \in \operatorname{img}\varphi$.

Since $f{\in}V$ was arbitrary, we get $V \subseteq \operatorname{img}\varphi$.

Together with (21), we obtain

$$\operatorname{img}\varphi \;=\; V\,. \tag{27}$$

Lem. IV.2.1.6 implies

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

Due to this fact and (27), the map

$$\psi\colon \mathsf{State}/_{\approx} \to \mathsf{Glob}{\times}V,\; \big[(\bar g, \bar l)\big]_\approx \mapsto \big(\bar g, \varphi(\bar l)\big)$$

is well defined and a bijection. ∎

Proposition E.1. *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: this is a standard combinatorial lemma. 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 balls into m baskets can be written as a string over $\odot$ and $|$, e.g., $\odot\odot|\odot||\odot\odot\odot|||\odot|$, where $\odot$ denotes a ball, $|$ separates the baskets, and there are k symbols $\odot$ 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 $\odot$ 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 $\binom{k+m-1}{k}$. ∎

Proof of Lem. IV.2.1.10

Let $k_{a,\rightsquigarrow} \overset{\text{def}}{=} \big|\{t{\in}n \mid l_t{=}a \wedge {\to_t}\backslash D = \rightsquigarrow\}\big|$ for each $a \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. Prop. E.1 implies that $\Big|\big\{g \in (\mathsf{Loc} \to \mathbb{N}_{\geqslant 0}) \;\big|\; \sum_{c\in\mathsf{Loc}} g(c) = k_{a,\rightsquigarrow}\big\}\Big| = \binom{k_{a,\rightsquigarrow}+L-1}{k_{a,\rightsquigarrow}} = \binom{k_{a,\rightsquigarrow}+L-1}{L-1}$ for

each $a \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. Thus,

$$\prod_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} \Big|\big\{g \in (\mathsf{Loc}\to\mathbb{N}_{\geqslant 0}) \mid \sum_{c\in\mathsf{Loc}} g(c) = k_{a,\rightsquigarrow}\big\}\Big| = \prod_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} \binom{k_{a,\rightsquigarrow}+L-1}{L-1}. \quad (28)$$

The left-hand side of (28) is equal to $\Big|\prod_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} \{g \in (\mathsf{Loc}\to\mathbb{N}_{\geqslant 0}) \mid \sum_{c\in\mathsf{Loc}} g(c) = k_{a,\rightsquigarrow}\}\Big| = \Big|\big\{f \in \big((\mathsf{Loc}\times E) \to \mathsf{Loc} \to \mathbb{N}_{\geqslant 0}\big) \;\Big|\; \forall a\in\mathsf{Loc}, \rightsquigarrow\in E\colon \sum_{c\in\mathsf{Loc}} f(a,\rightsquigarrow)(c) = k_{a,\rightsquigarrow}\big\}\Big| = |V|$. Since $\sum_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} k_{a,\rightsquigarrow} = n$, the right-hand side of (28) is bounded above by $\max\big\{\prod_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} \binom{\hat{k}_{a,\rightsquigarrow}+L-1}{L-1} \mid (\hat{k}_{a,\rightsquigarrow})_{a\in\mathsf{Loc},\rightsquigarrow\in E} \in (\mathbb{N}_{\geqslant 0})^{\mathsf{Loc}\times E} \wedge \sum_{\substack{a\in\mathsf{Loc}\\ \rightsquigarrow\in E}} \hat{k}_{a,\rightsquigarrow} = n\big\} = \Big[$knowing that $|\mathsf{Loc}\times E| = L\cdot 2^{G^2L^2-GL} = L\cdot 2^{GL(GL-1)}\Big]$ $\max\big\{\prod_{i<L\cdot 2^{GL(GL-1)}} \binom{\hat{k}_i+L-1}{L-1} \mid (\hat{k}_i)_{i<L\cdot 2^{GL(GL-1)}} \in (\mathbb{N}_{\geqslant 0})^{L\cdot 2^{GL(GL-1)}} \wedge \sum_{i<L\cdot 2^{GL(GL-1)}} \hat{k}_i = n\big\}$. Therefore,

$$|\mathsf{Glob}\times V| \leqslant G\cdot\max\left\{\prod_{i<L\cdot 2^{GL(GL-1)}} \binom{\hat{k}_i+L-1}{L-1} \;\middle|\; (\hat{k}_i)_{i<L\cdot 2^{GL(GL-1)}} \in (\mathbb{N}_{\geqslant 0})^{L\cdot 2^{GL(GL-1)}} \wedge \sum_{i<L\cdot 2^{GL(GL-1)}} \hat{k}_i = n\right\}.$$

■

Proof of Lem. IV.2.1.11

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

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

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

$$\delta \overset{\text{def}}{=} \min\left\{\frac{n}{t}-k_i,\, k_j-\frac{n}{t}\right\} \quad\text{and}$$

$$\tilde{k}_h \overset{\text{def}}{=} \begin{cases} k_h\,, & \text{if } i\neq h\neq j\,,\\ k_h+\delta\,, & \text{if } h=i\,,\\ k_h-\delta\,, & \text{if } h=j \end{cases} \qquad (h<t).$$

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

$$\prod_{h<t} \binom{\tilde{k}_h+m}{m} \leqslant \binom{n/t+m}{m}^t. \quad (29)$$

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 $\frac{\left(\prod_{h<m} \frac{\tilde{k}_i+m-h}{h+1}\right)\left(\prod_{h<m} \frac{\tilde{k}_j+m-h}{h+1}\right)}{\left(\prod_{h<m} \frac{k_i+m-h}{h+1}\right)\left(\prod_{h<m} \frac{k_j+m-h}{h+1}\right)} = \prod_{h<m} \frac{(\tilde{k}_i+m-h)(\tilde{k}_j+m-h)}{(k_i+m-h)(k_j+m-h)} = \prod_{r=1}^{m} \frac{(\tilde{k}_i+r)(\tilde{k}_j+r)}{(k_i+r)(k_j+r)} = \prod_{r=1}^{m} \frac{(k_i+r+\delta)(k_j+r-\delta)}{(k_i+r)(k_j+r)} = \prod_{r=1}^{m} \frac{(k_i+r)(k_j+r)+\delta(k_j+r)-\delta(k_i+r)-\delta^2}{(k_i+r)(k_j+r)} = \prod_{r=1}^{m} \Big(1+\frac{\delta(k_j-k_i)-\delta^2}{(k_i+r)(k_j+r)}\Big) = \prod_{r=1}^{m} \Big(1+\delta\frac{k_j-k_i-\delta}{(k_i+r)(k_j+r)}\Big) \geqslant$

[since $k_j-k_i-\delta = k_j-\frac{n}{t}+\frac{n}{t}-k_i-\delta \geqslant \delta+\delta-\delta = \delta > 0$]

$\prod_{r=1}^{m} \Big(1+\frac{\delta^2}{(k_i+r)(k_j+r)}\Big) \geqslant 1.$

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 (29)] $\binom{n/t+m}{m}^t$. ■

(An aside is worth to be made. A mathematically inclined reader could notice that $\binom{\frac{n}{t}+m}{m}^t <$ [using [79]] e^{nH_m} for all $m,n,t\in\mathbb{N}_+$, where $H_m = \sum_{i=1}^{m} \frac{1}{i}$ is the m^{th} Harmonic number. Omitting intermediate computations, this inequality would lead to $\mathsf{diamax}(n) < Ge^{nH_{L-1}}$. Such an upper bound would be exponential in n, and it would be possible to asymptotically reduce it to $\mathsf{O}\big((1+\varepsilon)^n\big)$ for an arbitrarily small $\varepsilon\in\mathbb{R}_+$ at the cost of arbitrarily large constants hidden in the asymptotic notation.)

Proof of Cor. IV.2.1.12

$\Big|\mathsf{State}/_{\approx}\Big|$ = [according to Lem. IV.2.1.9] $|\mathsf{Glob}\times V| \leqslant$ [using Lem. IV.2.1.10] $G\cdot\max\big\{\prod_{i<L\cdot 2^{GL(GL-1)}} \binom{k_i+L-1}{L-1} \mid k_0,\ldots,k_{L\cdot 2^{GL(GL-1)}-1} \in \mathbb{N}_{\geqslant 0} \wedge \sum_{i<L\cdot 2^{GL(GL-1)}} k_i = n\big\} \leqslant$ [applying Lem. IV.2.1.11 to $m = L-1$ and $t = L\cdot 2^{GL(GL-1)}$] $G\binom{\frac{n}{L\cdot 2^{GL(GL-1)}}+L-1}{L-1}^{L\cdot 2^{GL(GL-1)}}$. ■

Proposition E.2. $\forall x\in\mathbb{Q}, m\in\mathbb{N}_{\geqslant 0}\colon \binom{x+m}{m} = \prod_{j=1}^{m} \frac{x+j}{j}$.

(Remark: This is a standard, simple combinatorial lemma.)

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] $\frac{\prod_{r=1}^{m}(x+r)}{\prod_{h=1}^{m} h} = \prod_{j=1}^{m} \frac{x+j}{j}$. ■

Proof of Thm. IV.2.1.13

In this proof, we use the fact that the n-threaded program

as well as the initial state assumed for the prior claims in § IV.2.1 were arbitrary.

We have $\mathsf{diamax}(n) = \max\{\mathsf{diam}(p) \mid p \text{ is an } n\text{-threaded program}\} = \max\{(\mathsf{img}\,\mathsf{d}_p) \setminus \{\infty\} \mid p \text{ is an } n\text{-threaded program}\} <$ [using Lem. IV.2.1.8] $\max\left\{ \left|{}^{\mathsf{State}}\!/\!_{\approx}\right| \;\middle|\; \exists \text{ an } n\text{-threaded program } p\colon \mathsf{State} \text{ is the set of states of } p \text{ and } \approx \text{ is defined as in Def. IV.2.1.1 for } p \right\} \leqslant$ [using Cor. IV.2.1.12] $G\left(\frac{n}{L\cdot 2^{GL(GL-1)}} + L - 1 \atop L-1\right)^{L\cdot 2^{GL(GL-1)}} =$ [using Prop. E.2] $G\left(\prod_{r=1}^{L-1} \frac{\frac{n}{L\cdot 2^{GL(GL-1)}} + r}{r}\right)^{L\cdot 2^{GL(GL-1)}} \leqslant$ [for $L{=}1$, the product in the last line is empty and has value 1, and for $L{\geqslant}2$, $1 \leqslant r \leqslant L{-}1$, and $n{\geqslant}1$, we have $\frac{\frac{n}{L\cdot 2^{GL(GL-1)}}+r}{r} = \frac{n}{L\cdot 2^{GL(GL-1)}\cdot r} + 1 \leqslant \frac{n}{2\cdot 2^{1\cdot 2(1\cdot 2-1)\cdot 1}} + 1 = \frac{n}{2\cdot 2^2} + 1 = \frac{n}{8} + 1 \leqslant 2n$] $G(2n)^{(L-1)L\cdot 2^{GL(GL-1)}}$. ■

Appendix F.
Proofs of claims from § IV.2.2

Proof of Lem. IV.2.2.1

Let $(g,l) \in \mathsf{State}$ and $g' \in \mathsf{Glob}$.

If $g{=}g'$, take $l'{=}l$ and obtain $(g,l) \longrightarrow^{\leqslant \min\{\mathcal{C},(G-1)L^m,\mathsf{diam}(h)\}} (g',l')$ in zero steps.

Otherwise, $g'{\neq}g$. Since the transition graph is strongly connected, there is a walk from (g,l) to some state $(g',\_)$. Among all such walks, we choose a shortest one, say, $\sigma = (\sigma_j)_{j\leqslant k}$; then σ is a path and $k \geqslant 1$. Let l' be such that $(g',l') = \sigma_k$. By the definition of $\mathcal{C}$, we have $\mathsf{d}^{\mathsf{loc}}((g,l),0,(g',l'_0)) \leqslant \mathcal{C}$. According to the definition of local distances, there is a walk $\hat\sigma = (\hat\sigma_j)_{j\leqslant \hat k}$ and $\hat l \in \mathsf{Loc}^m$ such that $\hat\sigma_0 = (g,l) \wedge \hat\sigma_{\hat k} = (g',\hat l) \wedge \hat l_0 = l'_0 \wedge \hat k = \mathsf{d}^{\mathsf{loc}}((g,l),0,(g',l'_0))$. The choice of σ implies $\mathsf{length}(\sigma) \leqslant \mathsf{length}(\hat\sigma)$, which, in turn, implies $k \leqslant \hat k \leqslant \mathcal{C}$.

Since σ is shortest, it contains no repetitions of states. Thus, each state from $(\mathsf{Glob} \setminus \{g'\}) \times \mathsf{Loc}^m$ occurs in σ at most once. Assume for the purpose of contradiction that $k > (G-1)L^m$. Then $(\sigma_j)_{j\leqslant (G-1)L^m}$ is a strict prefix of σ and contains at least $(G-1)L^m + 1$ different states, so it contains at least one state outside $(\mathsf{Glob} \setminus \{g'\}) \times \mathsf{Loc}^m$, i.e., a state of the form $(g',\_)$. This is a contradiction to the minimality of k. Thus our assumption was false and $k \leqslant (G-1)L^m$. Since σ is a shortest path from (g,l) to (g',l'), we have $k \leqslant \mathsf{diam}(h)$. Notice that $(g,l) \longrightarrow^{\leqslant k} (g',l')$. Hence, $(g,l) \longrightarrow^{\leqslant \min\{\mathcal{C},(G-1)L^m,\mathsf{diam}(h)\}} (g',l')$. ■

Lemma F.1. *Let h be an m-threaded subprogram of an n-threaded program $p = (\to_0,\dots,\to_{n-1})$ via an embedding f, and the transition graph of h be strongly connected. Let $\longrightarrow$ be the transition relation of p, $i \in n \setminus (\mathsf{img}\, f)$, (g,l) a state of p, $k \in \mathbb{N}_{\geqslant 0}$, $\sigma = \left(\sigma^{[j]}\right)_{j\leqslant k}$ a sequence of local states that starts with l_i and satisfies $\forall\, j < k\colon \left(\_,\sigma^{[j]}\right) \to_i \left(\_,\sigma^{[j+1]}\right) \vee \sigma^{[j]} = \sigma^{[j+1]}$. Then there is a state $(\hat g,\hat l)$ of p such that $\hat l|_{n\setminus((\mathsf{img}\, f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\, f)\cup\{i\})}$, $\sigma^{[k]} = \hat l_i$, and $(g,l) \longrightarrow^{\leqslant (\min\{\mathcal{C},(G-1)L^m,\mathsf{diam}(h)\}+1)(L-1)} (\hat g,\hat l)$.*

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 are now going to shorten σ, producing a sequence θ containing no repetitions. Formally, the contraction is performed by Alg. 4.

Algorithm 4: Contracting σ into θ.
Input: σ, k
Program variables: nonnegative integers s, j, sequence θ over Loc
Output: s, θ

```
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:

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

Thus, we obtain the following postcondition of the program:

- $\mathrm{j} = \mathrm{s} = \mathsf{length}(\theta)$,
- $\theta^{[0]} = \sigma^{[0]} \;\wedge\; \theta^{[\mathrm{s}]} = \sigma^{[k]}$,
- for all $\bar\jmath < \mathrm{s}$, the local state $\theta^{[\bar\jmath]}$ occurs in θ exactly once, and
- $\forall\, \bar\jmath < \mathrm{s}\colon (\_,\theta^{[\bar\jmath]}) \to_i (\_,\theta^{[\bar\jmath+1]})$.

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 θ 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 and $\delta = \min\{\mathcal{C},(G{-}1)L^m,\mathsf{diam}(h)\}$. Since the transition graph of h is strongly connected, Lem. IV.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 even $j \leqslant 2s$ with $j \geqslant 2$ 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 $\big(\bar{g}^{[0]}, (\varphi_{f(r)}^{[0]})_{r<m}\big) = \big(g, (l_{f(r)})_{r<m}\big) \leadsto^{\leqslant\delta} (\check{e}^{[0]}, \tau^{[0]}) = \big(\bar{g}^{[1]}, (\varphi_{f(r)}^{[1]})_{r<m}\big)$, $\varphi_i^{[0]} = l_i = \sigma^{[0]} = \theta^{[0]} = \varphi_i^{[1]}$, $\varphi^{[0]}|_{n\setminus((\mathsf{img}\, f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\, f)\cup\{i\})} = \varphi^{[1]}|_{n\setminus((\mathsf{img}\, f)\cup\{i\})}$, and so $(\bar{g}^{[0]}, \varphi^{[0]}) \longrightarrow^{\leqslant\delta} (\bar{g}^{[1]}, \varphi^{[1]})$.
- For odd $j < 2s$ we have $\big(\bar{g}^{[j]}, \varphi_i^{[j]}\big) = \big(\check{e}^{[(j-1)/2]}, \theta^{[(j-1)/2]}\big) \to_i \big(\hat{e}^{[(j-1)/2]}, \theta^{[(j+1)/2]}\big) = \big(\bar{g}^{[j+1]}, \varphi_i^{[j+1]}\big)$, $\varphi_u^{[j]} = \tau_{f^{-1}(u)}^{[(j-1)/2]} = \varphi_u^{[j+1]}$ (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 $\big(\bar{g}^{[j]}, \varphi^{[j]}\big) \longrightarrow \big(\bar{g}^{[j+1]}, \varphi^{[j+1]}\big)$.
- For even $j < 2s$ such that $j \neq 0$ we have $\Big(\bar{g}^{[j]}, \big(\varphi_{f(r)}^{[j]}\big)_{r<m}\Big) = \big(\hat{e}^{[j/2-1]}, \tau^{[j/2-1]}\big)$ [since $j \leqslant 2s-2$, we get $\frac{j}{2} \leqslant s{-}1$, and so $\frac{j}{2} - 1 < s-1$] $\leadsto^{\leqslant\delta} \big(\check{e}^{[j/2]}, \tau^{[j/2]}\big) = \Big(\bar{g}^{[j+1]}, \big(\varphi_{f(r)}^{[j+1]}\big)_{r<m}\Big)$, $\varphi_i^{[j]} = \theta^{[j/2]} = \varphi_i^{[j+1]}$, $\varphi^{[j]}|_{n\setminus((\mathsf{img}\, f)\cup\{i\})} = l|_{n\setminus((\mathsf{img}\, f)\cup\{i\})} = \varphi^{[j+1]}|_{n\setminus((\mathsf{img}\, f)\cup\{i\})}$, and so $\big(\bar{g}^{[j]}, \varphi^{[j]}\big) \longrightarrow^{\leqslant\delta} \big(\bar{g}^{[j+1]}, \varphi^{[j+1]}\big)$.
- 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, particularly, $\big(\bar{g}^{[0]}, \varphi^{[0]}\big) \longrightarrow^* \big(\bar{g}^{[2s]}, \varphi^{[2s]}\big)$. The number of odd positive integers below $2s$ is $\big|\{i \mid i \text{ odd } \wedge\ 1 \leqslant i \leqslant 2s-1\}\big| = \frac{(2s-1)-1}{2} + 1 = s$. The number of even positive integers below $2s$ is $\big|\{i \mid i \text{ even } \wedge\ 2 \leqslant i \leqslant 2s-2\}\big| = \frac{(2s-2)-2}{2} + 1 = s{-}1$. Thus, the length of the just mentioned walk is at most $\delta + s + \delta(s{-}1) = (\delta{+}1)s \leqslant (\delta{+}1)(L{-}1)$. We set $(\hat{g}, \hat{l}) \stackrel{\text{def}}{=} \big(\bar{g}^{[2s]}, \varphi^{[2s]}\big)$. ■

Proof of Lemma IV.2.2.2

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 $\zeta = (\min\{\mathcal{C}, (G{-}1)L^m, \mathsf{diam}(h)\}+1)(L{-}1)$.

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 the transition graph of 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, $\mathsf{length}(\hat{\tau}) \leqslant \mathsf{diam}(h) \leqslant \zeta(n-m) + \mathsf{diam}(h)$, and the lemma is proven.

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 $\sigma_{i_0}^{[0]} = l_{i_0}$ and $\forall\, \bar{\jmath} < k\colon\ (\_, \sigma_{i_0}^{[\bar{\jmath}]}) \to_{i_0} (\_, \sigma_{i_0}^{[\bar{\jmath}+1]}) \vee \sigma_{i_0}^{[\bar{\jmath}]} = \sigma_{i_0}^{[\bar{\jmath}+1]}$. By Lem. F.1, there is a program state $(\hat{g}, \hat{l})$ of p such that $(g,l) \longrightarrow^{\leqslant\zeta} (\hat{g}, \hat{l})$, $\sigma_{i_0}^{[k]} = \hat{l}_{i_0}$, 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 $\hat{l}_r = \hat{l}_{i_0} = \sigma_{i_0}^{[k]} = l'_{i_0} = l'_r$.

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 $r \in n \setminus (\mathsf{img}\, f) \wedge r > i_{j-1}$, $\sigma_{i_j}^{[0]} = l_{i_j} = \check{l}_{i_j}$, and $\forall\, \bar{\jmath} < k\colon\ (\_, \sigma_{i_j}^{[\bar{\jmath}]}) \to_{i_j} (\_, \sigma_{i_j}^{[\bar{\jmath}+1]}) \vee \sigma_{i_j}^{[\bar{\jmath}]} = \sigma_{i_j}^{[\bar{\jmath}+1]}$. Lem. F.1 implies the existence of a program state $(\hat{g}, \hat{l})$ of p such that $(\check{g}, \check{l}) \longrightarrow^{\leqslant\zeta} (\hat{g}, \hat{l})$, $\sigma_{i_j}^{[k]} = \hat{l}_{i_j}$, 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 ex-

ceeding 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 the transition graph of h is strongly connected, it exhibits a path τ from the h-components of the last state of φ_{s-1} to $\left(g', \left(l'_{f(r)}\right)_{r<m}\right)$ 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 $\varphi_0 \dots \varphi_{s-1}\psi$ has length at most $\zeta s + \mathsf{diam}(h) \leqslant \zeta(n-m) + \mathsf{diam}(h)$. ■

Appendix G. Proofs of claims from § IV.2.3

Proof of Lem. IV.2.3.1

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 $\mathsf{diam}(h) \leqslant GL-1$. Lem. IV.2.2.2 implies $\mathsf{diam}(p) \leqslant (\min\{(G{-}1)L, GL-1\}+1)(L{-}1)(n{-}1) + GL - 1 = (\min\{GL-L, GL-1\}+1)(L{-}1)(n{-}1)+GL-1 \leqslant ((GL-L)+1)(L{-}1)(n{-}1)+GL-1 = (GL-L+1)(L{-}1)n-(GL-L+1)(L{-}1)+GL-1 = (GL-L+1)(L{-}1)n-(GL^2-L^2+L-GL+L-1)+GL-1 = (GL{-}L{+}1)(L{-}1)n-(GL^2-L^2-GL+2L-1)+GL-1 = (GL{-}L{+}1)(L{-}1)n-GL^2+L^2+GL-2L+1+GL-1 = (GL-L+1)(L{-}1)n-GL^2+L^2+2GL-2L = (GL-L+1)(L{-}1)n+(-GL+L+2G-2)L = (GL-L+1)(L{-}1)n+(-L(G-1)+2(G-1))L = (GL-L+1)(L{-}1)n+(2{-}L)(G{-}1)L$. ■

Proof of Thm. IV.2.3.2

The probability for a thread to satisfy the prerequisites for Lem. IV.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. ■

Proof of Note IV.2.3.3

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 $e < (GL)Ln + 0 = GL^2n$. ■

Appendix H. Proof of claims from § V

Proof of Thm. V.1

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 Lem. IV.2.1.6. 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 $(\to_i)_{i<n} = p$ be the program, $(g,l) = s$ the source program state, and $(g',l') = s'$ the target one. Next, we allocate $L^2 2^{GL(GL-1)}$ counters of $\lceil \log(n+1) \rceil$ bits each on the working tape of M. We will refer to these counters as $c_{a,\rightsquigarrow,b}$ for $a,b \in \mathsf{Loc}$ and $\rightsquigarrow \in E$, where E has been defined in Def. IV.2.1.5. In the following state exploration, a newly reached program state $(\hat{g},\hat{l})$ will be tracked in these counters: $c_{a,\rightsquigarrow,b}$ will store $\left|\{t{<}n \mid l_t{=}a \wedge {\to_t}\setminus D = {\rightsquigarrow} \wedge \hat{l}_t{=}b\}\right|$ for each $a,b \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. 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 $c_{a,\rightsquigarrow,b}$ will be initialized to $\left|\{t{<}n \mid l_t{=}a{=}b \wedge {\to_t}\setminus D{=}{\rightsquigarrow}\}\right|$ for $a,b \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. 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 $G(2n)^{L(L-1)2^{GL(GL-1)}}$. If so, M halts, since it has visited some equivalence class twice (according to the proof of Thm. IV.2.1.13 in the appendix). Otherwise, M continues and checks whether s' has been reached. To do this, M checks whether g' is equal to the current shared part and l' has exactly as many threads of different classes as described by the counters $c_{a,\rightsquigarrow,b}$ for $a,b \in \mathsf{Loc}$ and $\rightsquigarrow \in E$. If it is the case, M accepts: then an execution from s to s' exists. Otherwise, M nondeterministically chooses a thread index $i{<}n$ and a non-loop thread transition from $\to_i$, say, $\big((\hat{g},a),(\hat{g}',a')\big) \in {\to_i}\setminus D$. 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 $c_{l_i,\to_i\setminus D,a} > 0$. 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 $c_{l_i,\to_i\setminus D,a}$, and increments $c_{l_i,\to_i\setminus D,a'}$. 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 Lem. IV.2.1.7. 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 $\sum_{a,b\in\mathsf{Loc},\,\rightsquigarrow\in E} c_{a,\rightsquigarrow,b}$ remains constant; therefore, $\mathsf{O}(\log n)$ space suffices for each of the $L^2 2^{GL(GL-1)}$ counters. Taking into account also the shared-state variable of size $\lceil\log(G{+}1)\rceil$ on the working tape, 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 $\mathsf{Reach}\in\mathsf{NSpace}(\log n)$.

Since $\mathsf{NSpace}(\log n)$ is closed under complementation [80, 81], the non-reachability problem also belongs to $\mathsf{NSpace}(\log n)$. ■

Proof of Thm. V.2

We will describe of a logspace-uniform family of circuits that will recognize $\mathsf{Reach}^{\mathsf{loc}}$ similarly to how the machine M from the proof of Thm. V.1 does it. Given $m\in\mathbb{N}_{\geqslant 0}$, we now describe a circuit with m inputs, polynomial size in m, and $\mathsf{O}(\log m)$ depth that recognizes all words in $\mathsf{Reach}^{\mathsf{loc}}$ 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 $\lfloor\log_2 G\rfloor+1+(\lfloor\log_2 L\rfloor+1)n$ 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 $(G^2L^2+1)n+\lfloor\log_2 G\rfloor+1+(\lfloor\log_2 L\rfloor+1)n=\big(G^2L^2+\lfloor\log_2 L\rfloor+2\big)n+\lfloor\log_2 G\rfloor+1$ till $m-\Big(\big(\lfloor\log_2 G\rfloor+1\big)+\big(\lfloor\log_2 L\rfloor+1\big)\Big)-1=m-\lfloor\log_2 G\rfloor-\lfloor\log_2 L\rfloor-3$ as the binary encoding of i. Interpret the remaining positions $m-\lfloor\log_2 G\rfloor-\lfloor\log_2 L\rfloor-2$ 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, $(\to_i)_{i<n}=p$ will be the program, $(g,l)=s$ the source program state, and $(g',l')=\tau$ the target thread state. The machine M^{loc} allocates $L2^{GL(GL-1)}$ binary counters on the working tape; each binary counter has a width of $\lfloor\log_2\mathcal{C}\rfloor+1$ bits, where $\mathcal{C}$ is the maximal local diameter (cf. Def. and Cor. III.9). We will refer to these counters as $\tilde{c}_{a,\rightsquigarrow}$ for $a\in\mathsf{Loc}$ and $\rightsquigarrow\in E$. The machine also copies the shared state g, the thread transition relation $\to_i\backslash D$, and the target thread state τ 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 $\tilde{c}_{a,\rightsquigarrow}\leqslant C$ threads starting in the local state a and having the transition relation $\rightsquigarrow$ (for $a\in\mathsf{Loc}$ and $\rightsquigarrow\in E$); the goal is to decide on the reachability of τ in any thread with the transition relation $\to_i\backslash D$. 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 $G\cdot(\mathcal{C}+1)^{L\cdot 2^{GL(GL-1)}}\cdot 2^{GL(GL-1)}\cdot GL$), 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 $\mathcal{C}$ 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 $\mathcal{C}$ from s to the thread state τ of the thread i. Therefore, at most $\mathcal{C}$ 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 $\mathcal{C}$ 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 $\mathsf{Reach}^{\mathsf{loc}}$.

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 $\mathsf{NonReach}^{\mathsf{loc}}$, 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 $\mathsf{d}_p^{\mathsf{loc}}(s,i,\tau)=\infty$. ■