Multithreaded-Cartesian Abstract Interpretation of Multithreaded Recursive Programs Is Polynomial

Alexander Malkis*
Technische Universität München, Germany

Accepted on 2015-07-13, last updated on 2024-10-07.

Abstract Undecidability is the scourge of verification for many program classes. We consider the class of shared-memory multithreaded programs in the interleaving semantics such that the number of threads is finite and constant throughout all executions, each thread has an unbounded stack, and the shared memory and the stack-frame memory are finite. Verifying that a given program state does not occur in executions of such a program is undecidable. We show that the complexity of verification drops to polynomial time under multithreaded-Cartesian abstraction. Furthermore, we demonstrate that multithreaded-Cartesian abstract interpretation generates an inductive invariant which is a regular language. Under logarithmic cost measure, both proving non-reachability and creating a finite automaton can be attained in $\mathcal{O}(n \log_2 n)$ time in the number of threads n and in polynomial time in all other quantities.

1 Introduction

Verification of multithreaded programs is hard. In the presence of recursive procedures, the problem of membership in the strongest inductive invariant is undecidable: given a two-threaded program with a stack per thread, one can simulate a Turing tape. However, if the stack depth is the only unbounded quantity, there might be interesting inductive invariants of special forms such that membership in such invariants is decidable. In other words, one might circumvent undecidability by considering specially-formed overapproximations of the set of program states that are reachable from the initial ones.

We now briefly sketch one such interesting form. Let a program state be an $(n+1)$-tuple in which one component contains a valuation of the shared variables and each of the remaining n components contains a valuation of the local variables (including the control flow) of a distinct thread. Let us say that two program states are equivalent if they have the same shared-variables entry. We define a set of states to be of the *multithreaded-Cartesian* form if each equivalence class is an $(n+1)$-dimensional Cartesian product. (A rigorous definition will appear in § 4.) The multithreaded-Cartesian inductive invariants of a program constitute a Moore family.

It is known that in the finite-state case the membership problem for the strongest multithreaded-Cartesian inductive invariant is in PTIME [17]. We extend this result to programs in which each thread has a potentially unbounded stack. Moreover,

* I greatly acknowledge useful comments and suggestions from Neil Deaton Jones.

we show that the strongest multithreaded-Cartesian inductive invariant is a regular language when viewed as a formal language of strings. Computing a corresponding finite automaton as well as solving the membership problem for the strongest multithreaded-Cartesian inductive invariant can be accomplished in time $\mathcal{O}(n \log_2 n)$, where n is the number of threads, and in polynomial time in all the other quantities.

The presentation will proceed as follows.

- After an overview of related work, we rigorously define our program class in § 3 and formulate the problem of determining the strongest multithreaded-Cartesian inductive invariant in the abstract interpretation framework in § 4.
- Next, in § 5, we present a new inference system, which we call TMR, which constructs n automata such that the i^{th} automaton describes an overapproximation of the set of pairs (shared state, stack word of the i^{th} thread) that occur in the computations of the multithreaded program.
- Based on the computation result of TMR, we show how to create an automaton that describes the strongest multithreaded-Cartesian inductive invariant.
- Then, we determine the asymptotic worst-case running times of TMR and the automaton construction under logarithmic cost measure [18].
- In §§ 6–7, we conclude with the proof of correctness of our construction.

We make sure that if some or all of the input quantities (the number of threads, the number of shared states, and the number of different stack frames) are infinite, TMR still leads to a logically valid (but not necessarily executable) description of the multithreaded-Cartesian abstract interpretation. This opens way to using constraint solvers in the infinite case. We will impose finiteness restrictions only when presenting low-level algorithms and computing the running times.

Due to restricted space, most computations and proofs are found in the appendix.

2 Related Work

There is a large body of work on the analysis of concurrent programs with recursion; we discuss next only the literature which is, subjectively, most related to our work.

The roots of multithreaded-Cartesian abstraction date back to the Owicki-Gries proof method [21], followed by thread-modular reasoning of C. B. Jones [14], and the Flanagan-Qadeer model-checking method for nonrecursive programs [12]. The basic versions of these methods (without auxiliary variables) exhibit the same strength. This strength is precisely characterized by multithreaded-Cartesian abstract interpretation, which was first discovered by R. Cousot [11] and later rediscovered in [16, 17].

Flanagan and Qadeer [12] introduce also a method for recursive multithreaded programs, for which they claim an $\mathcal{O}(n^2)$ upper bound on the worst-case running time. Their analysis, which predates TMR, simultaneously computes procedure summaries and distributes the changes of the shared states between the threads. Where their algorithm is only summarization-based, our TMR is an explicit automaton construction. Our program model is slightly different compared to [12]. First, to simplify our presentation, we remove the concept of a local store: while in practice there may be different kinds of local stores (static-storage function-scope variables in the sense of the C programming language, the registers of a processor, . . .), every program can be modified to perform all thread-local computations on the stack. Second, we allow changing the shared state when a stack grows or shrinks to permit richer program models; whenever possible, we also allow infinite-size sets.

An alternative approach to prove polynomial time of multithreaded-Cartesian abstract interpretation could be to apply Horn clauses as done in [20] for some other problems. That way would not reveal regularity or connections to the algorithms of Flanagan and Qadeer; it will also just give a running-time bound for the unit cost measure, whereas we count more precisely in the logarithmic cost measure. We will not discuss it here.

Outside multithreaded-Cartesian abstract interpretation there are many other methods for analyzing concurrent recursive programs.

If the interplay between communication and recursion is restricted, decidable fragments can be identified; we will mention just a few. If only one thread has a stack and the other threads are finite-state, then one can construct the product of the threads and model-check a large class of properties [8, 25]. Alternatively, one can allow certain forks and joins but restrict communication only to threads that do not have ongoing procedure calls [6]. In the synchronous execution model, one may restrict the threads to perform all the calls synchronously and also perform all returns synchronously [1]. Alternatively, one may restrict pop operations to be performed only on the first nonempty stack [3].

Without restrictions on the interplay between communication and recursion, one may allow non-termination of the analysis [22] or be satisfied with an approximate analysis, which can be sound [5] or complete [15, 23] (for every choice of parameters), but never both. If shared-memory communication is replaced by rendezvous, verification is still undecidable [24].

The summarization idea behind TMR dates back to the works of Büchi [7]. Since then, it has been developed further in numerous variants for computing the exact semantics, e.g., for two-way deterministic pushdown automata with a write-once-read-many store [19], for imperative stack-manipulating programs with a rich set of operations on the stack [2], and in implementations of partial evaluators [13].

3 Programs

Now we introduce notation and our model of recursive multithreaded programs.

Let $\mathbb{N}_0$ (resp. $\mathbb{N}_+$) be the sets of natural numbers with (resp. without) zero. We write X^* (resp. X^+) for the set of finite (resp. finite nonempty) words over an alphabet X, ε for the empty word, and $|w|$ for the length of a word $w \in X^*$.

An *n-threaded recursive program* (from now on, simply a *program*) is a tuple

$$(\mathrm{Glob}, \mathrm{Frame}, \mathrm{init}, (\overset{\downarrow}{\sqcup}_t, \boxminus_t, \overset{\uparrow}{\sqcup}_t)_{t<n})$$

such that the following conditions hold:

- Glob and Frame are arbitrary sets such that, without loss of generality, $(\mathrm{Glob} \times \mathrm{Frame}) \cap \mathrm{Glob} = \emptyset$. (We think of Glob as the set of shared states, e.g., the set of valuations of shared variables. We think of Frame as the set of stack frames, where a stack frame is, e.g., a valuation of procedure-local variables and the control-flow counter. The necessity of the disjointness condition will get clear later on.)
- n is an arbitrary ordinal. (For our convenience, we think of n as both the number of threads and the set of thread identifiers. For example, we view $(\mathrm{Frame}^+)^n$ as the set of maps $n \to \mathrm{Frame}^+$, and, in the finite case, n as $\{0, 1, \ldots, n-1\}$. Real programs are usually modeled by finite n or $n = \omega$, and we allow arbitrary n.)

- init $\subseteq$ Glob $\times$ (Frame$^+$)^n is such that $\forall\,(g,l) \in$ init, $t{\in}n$: $|l_t| = 1$. (By $l_t = l(t)$ we indicate the t^{th} component of $l \in (\text{Frame}^+)^n$. We think of init as of the set of initial states. The depth of the stacks of the threads is 1 in every initial state.)
- For each $t{\in}n$, the transition relation of thread t is given by sets $\overset{\downarrow}{\sqcup}_t \subseteq (\text{Glob}\times\text{Frame})\times(\text{Glob}{\times}\text{Frame}{\times}\text{Frame})$, $\overset{\rightarrow}{\sqcup}_t \subseteq (\text{Glob}{\times}\text{Frame})^2$, and $\overset{\uparrow}{\sqcup}_t \subseteq (\text{Glob}{\times}\text{Frame}{\times}\text{Frame})\times(\text{Glob}{\times}\text{Frame})$. (These are sets of push, internal, and pop transitions of thread t, respectively.)

We denote by Loc $=$ Frame$^+$ the set of *local states* of each thread; the elements of Glob$\times$Loc are called *thread states*. The operational semantics of each thread $t{<}n$ is given by the relation $\rightsquigarrow_t \subseteq (\text{Glob}{\times}\text{Loc})^2$, which is defined by

$$
\begin{gathered}
(g,w) \rightsquigarrow_t (g',w') \quad \overset{\text{def}}{\Longleftrightarrow} \\
\big((\exists\, a,b,c \in \text{Frame},\ u \in \text{Frame}^*\colon\ \ w = au \,\wedge\, w' = bcu \,\wedge\, ((g,a),(g',b,c)) \in \overset{\downarrow}{\sqcup}_t) \\
\vee\ (\exists\, a,b \in \text{Frame},\ u \in \text{Frame}^*\colon\ \ w = au \,\wedge\, w' = bu \,\wedge\, ((g,a),(g',b)) \in \overset{\rightarrow}{\sqcup}_t) \\
\vee\ (\exists\, a,b,c \in \text{Frame},\ u \in \text{Frame}^*\colon\ \ w = abu \,\wedge\, w' = cu \,\wedge\, ((g,a,b),(g',c)) \in \overset{\uparrow}{\sqcup}_t)\big)
\end{gathered}
$$

for $g, g' \in$ Glob and $w, w' \in$ Loc. Notice that the stacks are always kept nonempty. Let the set of *program states* be

$$\text{State} \;=\; \text{Glob} \times \text{Loc}^n\,.$$

The operational semantics of the whole program is given by the *concrete domain*

$$D \;=\; \mathfrak{P}(\text{State})\,,$$

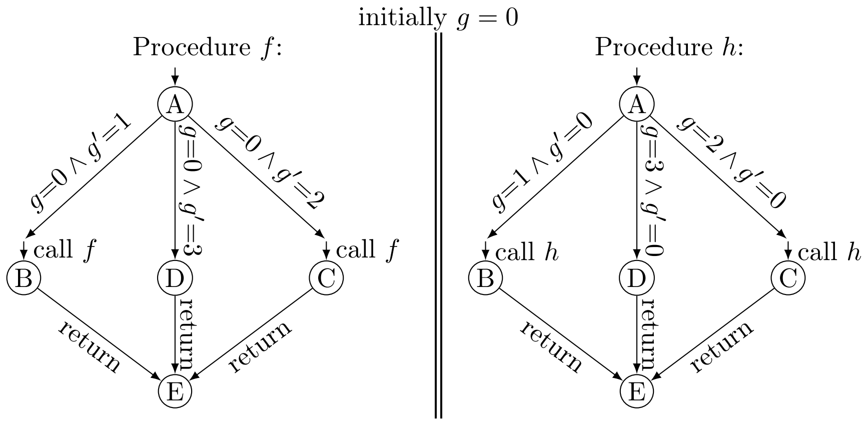
which is the power set of the set of program states, and the successor map

$$
\begin{aligned}
&\text{post} : D \to D\,,\\
&\qquad Q \mapsto \{(g',l') \mid \exists\, t{\in}n,\ (g,l) \in Q\colon (g,l_t) \rightsquigarrow_t (g',l'_t) \,\wedge\, \forall\, s \in n\backslash\{t\}\colon l_s = l'_s\}\,.
\end{aligned}
$$

Broadly speaking, program analyses compute or approximate the so-called *collecting semantics*, which is the strongest inductive invariant (lfp $=$ least fixpoint)

$$\text{lfp}(\lambda\, S \in D.\ \text{init} \cup \text{post}(S))\,.$$

This set can become rather complex, loosely speaking, due to subtle interplay between concurrency and recursion. A nontrivial example is presented by the following control-flow graph of a two-threaded program over a shared variable g:



Procedures f and h execute in parallel. Roughly speaking, the left thread announces how it builds its stack by changing g from 0 to 1 or 2, and the right thread follows the stack operations of the left thread, confirming that it proceeds by resetting g to 0. Setting g to 3 initiates reduction of the stacks. For simplification, we assume that the thread transitions between each pair of named consecutive control flow locations (A to B, A to C, A to D, B to E, C to E, D to E) are atomic.

We model this program by Glob $=$ {0,1,2,3}, Frame $=$ {A,B,C,D} (E does not occur in computations), n=2, init $= \{(0,(\mathrm{A},\mathrm{A}))\}$, $\overset{\downarrow}{\sqcup}_0 = \{((0,\mathrm{A}),(1,\mathrm{A},\mathrm{B})),((0,\mathrm{A}),(2,\mathrm{A},\mathrm{C}))\}$, $\overset{-}{\sqcup}_0 = \{((0,\mathrm{A}),(3,\mathrm{D}))\}$, $\overset{\uparrow}{\sqcup}_0 = \{((g,y,z),(g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B},\mathrm{C},\mathrm{D}\} \wedge z{\in}\mathrm{Frame}\}$, $\overset{\downarrow}{\sqcup}_1 = \{((1,\mathrm{A}),(0,\mathrm{A},\mathrm{B})),((2,\mathrm{A}),(0,\mathrm{A},\mathrm{C}))\}$, $\overset{-}{\sqcup}_1 = \{((3,\mathrm{A}),(0,\mathrm{D}))\}$, and $\overset{\uparrow}{\sqcup}_1 = \{((g,y,z),(g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B},\mathrm{C},\mathrm{D}\} \wedge z{\in}\mathrm{Frame}\}$.

One can show (Example 14) that the strongest inductive invariant is

$$
\begin{array}{l}
\{0\} \times \left(\begin{array}{l}
\quad \{(\mathrm{A}y,\mathrm{A}y),(\mathrm{D}y,\mathrm{D}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup \{(\mathrm{D}y,z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge \ z \text{ is a suffix of } y\} \\
\cup \{(y,\mathrm{D}z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge \ y \text{ is a suffix of } z\} \\
\cup \{(y,z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z \ \vee \ z \text{ is a suffix of } y)\}
\end{array}\right) \\
\cup \{1\} \times \{(\mathrm{AB}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup \{2\} \times \{(\mathrm{AC}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup \{3\} \times \left(\begin{array}{l}
\quad \{(\mathrm{D}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\
\cup \{(y,\mathrm{A}z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge y \text{ is a suffix of } z\}
\end{array}\right).
\end{array}
$$

This set, viewed as a formal language over Glob, Frame, and some special symbol separating the stacks, is not context-free.

Notice that $g \in \{0,3\}$ is a valid postcondition of the considered program. In the next section we will see what multithreaded-Cartesian abstract interpretation is and how it helps proving this postcondition.

4 Multithreaded-Cartesian Abstract Interpretation

Now we are going to describe an approximation operator on the concrete domain of states of a program, essentially recapitulating the key points of [16]. Loosely speaking, the definition of the approximation will not depend on the internal structure of Loc and post.

The *multithreaded-Cartesian approximation* is the map

$$\rho_{\mathrm{mc}} : D \to D, \quad S \mapsto \{(g,l) \in \mathrm{State} \mid \forall\, t{\in}n\, \exists\, \hat{l}{\in}\mathrm{Loc}^n \colon (g,\hat{l}) \in S \ \wedge\ l_t = \hat{l}_t\}\,,$$

which, intuitively, given a set of states, partitions it into blocks according to the shared state, and approximates each block by its Cartesian hull.

One can show that ρ_{mc} is an upper closure operator on $(D,\subseteq)$ (Prop. 17).

We define the *multithreaded-Cartesian (collecting) semantics* as the least fixpoint

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S)))\,.$$

For our running example, for any $S \subseteq \mathrm{State}$ we have

$$\rho_{\mathrm{mc}}(S) \ = \ \{(g,(l_0,l_1)) \mid (\exists\, \bar{l}_1{\in}\mathrm{Loc}\colon (g,(l_0,\bar{l}_1)) \in S) \ \wedge\ (\exists\, \bar{l}_0{\in}\mathrm{Loc}\colon (g,(\bar{l}_0,l_1)) \in S)\}.$$

The multithreaded-Cartesian semantics of our running example is (see Example 18)

$$\left(\begin{array}{l}\quad\{0\}\times(\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\}\cup\{\mathrm{B},\mathrm{C}\}^+)\\ \cup\,\{1\}\times\{\mathrm{AB}x \mid x\in\{\mathrm{B},\mathrm{C}\}^*\}\\ \cup\,\{2\}\times\{\mathrm{AC}x \mid x\in\{\mathrm{B},\mathrm{C}\}^*\}\\ \cup\,\{3\}\times(\{\mathrm{D}x \mid x\in\{\mathrm{B},\mathrm{C}\}^*\}\cup\{\mathrm{B},\mathrm{C}\}^+)\end{array}\right)\times(\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B},\mathrm{C}\}^*\}\cup\{\mathrm{B},\mathrm{C}\}^+)\,.$$

This set, viewed as a formal language, is regular; a corresponding regular expression is $\big(0(\mathrm{A|B|C|D})(\mathrm{B|C})^* \mid 1\mathrm{AB}(\mathrm{B|C})^* \mid 2\mathrm{AC}(\mathrm{B|C})^* \mid 3(\mathrm{B|C|D})(\mathrm{B|C})^*\big)\ \dagger\ (\mathrm{A|B|C|D})(\mathrm{B|C})^*$, where $\dagger{\notin}\mathrm{Frame}$ is a fresh symbol separating the local parts. Notice that the postcondition $g\in\{0,3\}$ holds also in this abstract semantics.

5 Model-Checking Recursive Multithreaded Programs

Now we develop an efficient algorithm to compute the input program's multithreaded-Cartesian semantics. First we show the inference system TMR, then we show how its output is interpreted as multithreaded-Cartesian semantics, and finally we turn to computational issues.

5.1 Inference system TMR

Given an n-threaded program as described in § 3, our algorithm generates n automata that describe an overapproximation of the set of stack words of the threads that occur in computations.

Fix some "fresh" element $\mathfrak{f}\notin\mathrm{Glob}\,\dot\cup\,(\mathrm{Glob}{\times}\mathrm{Frame})$. Let

$$V \;=\; \mathrm{Glob}\,\dot\cup\,\{(g',b)\mid \exists\, g{\in}\mathrm{Glob}, a{,}c\in\mathrm{Frame}, t{\in}n\colon\ ((g,a),(g',b,c))\in\overset{\downarrow}{\sqcup}_t\}\,\dot\cup\,\{\mathfrak{f}\}\,.$$

We now define binary relations $G_t\subseteq\mathrm{Glob}^2$ and ternary relations $\underset{t}{\rightarrow}\subseteq V{\times}\mathrm{Frame}{\times}V$ for all $t{\in}n$ by the following inference system.

$$(\textsc{tmr init})\ \frac{(g,l)\in\mathrm{init}}{g\xrightarrow[t]{l_t}\mathfrak{f}}\ t{\in}n \qquad (\textsc{tmr step})\ \frac{((g,a),(g',b))\in\overset{-}{\sqcup}_t\qquad g\xrightarrow[t]{a}v}{g'\xrightarrow[t]{b}v\qquad (g,g')\in G_t}\ t{\in}n$$

$$(\textsc{tmr push})\ \frac{g\xrightarrow[t]{a}v\qquad ((g,a),(g',b,c))\in\overset{\downarrow}{\sqcup}_t}{g'\xrightarrow[t]{b}(g',b)\xrightarrow[t]{c}v\qquad (g,g')\in G_t}$$

$$(\textsc{tmr pop})\ \frac{g\xrightarrow[t]{a}v\xrightarrow[t]{b}\bar v\qquad ((g,a,b),(g',c))\in\overset{\uparrow}{\sqcup}_t}{g'\xrightarrow[t]{c}\bar v\qquad (g,g')\in G_t}$$

$$(\textsc{tmr env})\ \frac{(g,g')\in G_t\qquad g\xrightarrow[s]{a}v}{g'\xrightarrow[s]{a}v}\ t{\neq}s\text{ are in }n$$

TMR INIT gathers stack contents of the initial states. TMR STEP, TMR PUSH, and TMR POP create an automaton describing thread states that occur in computations of the threads in isolation; the stacks are obtained from the upper labels of certain walks. Moreover, the three rules collect information about how the shared state is altered. The rule TMR ENV transfers shared-state changes between the threads.

For our program from page 4, the automata constructed by TMR are in Fig. 1.

5.2 Interpretation of the output of TMR

Now we define the set of states that the inference system represents.

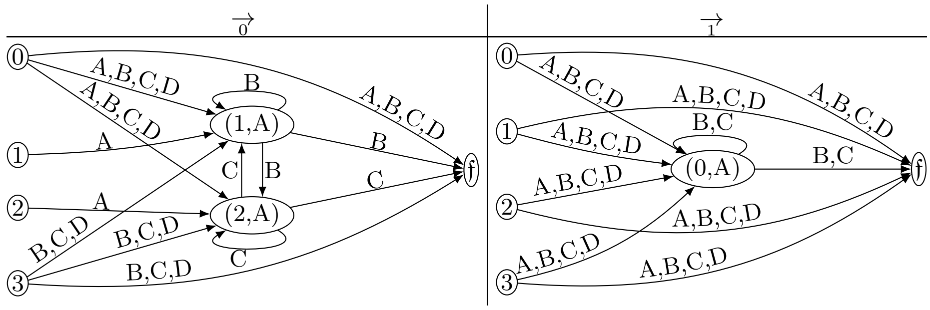


Figure 1. Automata constructed by TMR for our example. Each arrow on the left carries the lower index 0; each arrow on the right carries the lower index 1.

For that, we extend $\to$ to words of stack frames in a standard way. For each $t\in n$, consider the quaternary relation $\underset{t}{\to} \subseteq V\times\mathrm{Frame}^*\times\mathbb{N}_0\times V$ (slightly abusing notation, we employ the same symbol as for the ternary relation from § 5.1) defined by the following inference system:

$$\frac{}{v \xrightarrow[t]{\varepsilon}{}^{0}\; v} \qquad \frac{i\in\mathbb{N}_0 \qquad a\in\mathrm{Frame} \qquad y\in\mathrm{Frame}^* \qquad v \xrightarrow[t]{a} \hat{v} \xrightarrow[t]{y}{}^{i} \bar{v}}{v \xrightarrow[t]{ay}{}^{i+1} \bar{v}}$$

For each $t\in n$, we define $\underset{t}{\to}{}^{*} \subseteq V\times\mathrm{Frame}^*\times V$ by $\underset{t}{\to}{}^{*} = \bigcup_{i\in\mathbb{N}_0} \underset{t}{\to}{}^{i}$ and $L_{g,t} = \{w \mid g \xrightarrow[t]{w}{}^{*} \mathfrak{f}\}$ ($g\in\mathrm{Glob}$).

Informally, a walk $g \xrightarrow[t]{w}{}^{*} \mathfrak{f}$ means that the state (g, w) of thread t occurrs in the approximate semantics, and $g \xrightarrow[t]{w}{}^{*} (g', b)$ means that a procedure call starting with thread state (g', b) can reach (g, w) in thread t in the approximate semantics.

The inference system TMR represents the set

$$\bigcup_{g\in\mathrm{Glob}} \{g\} \times \prod_{t\in n} L_{g,t}\,. \tag{1}$$

5.3 Computing multithreaded-Cartesian semantics

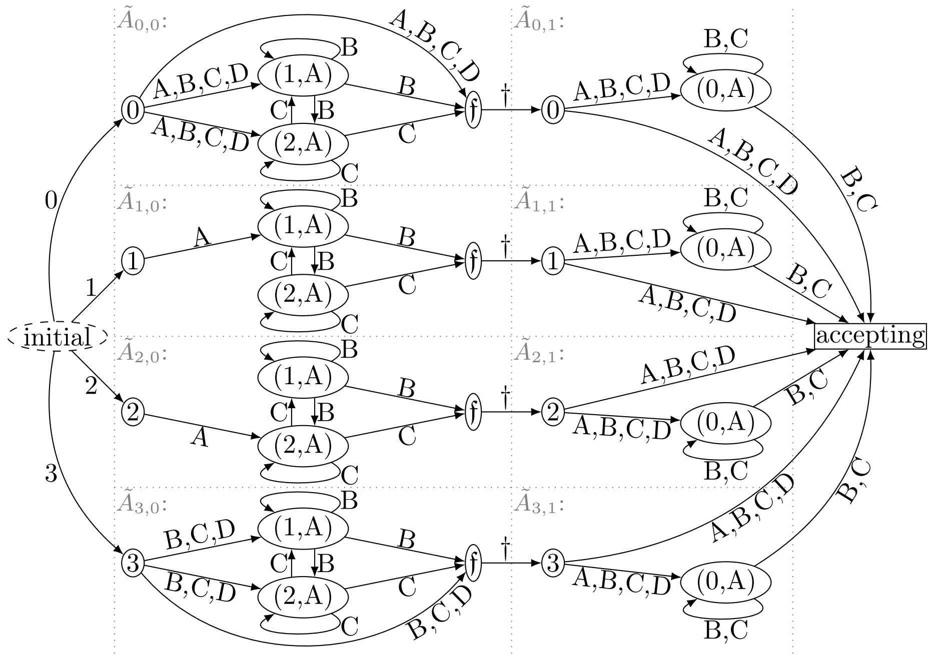
For actual computations, which we discuss now, let us assume finite n, Glob, and Frame till the end of § 5.3.

If we are just interested in checking non-reachability of thread states, executing TMR suffices: if a local state l is not in $L_{g,t}$, then the thread state (g, l) of the t^{th} thread does not occur in computations of the program ($g\in\mathrm{Glob}$, $l\in\mathrm{Loc}$, $t\in n$). If we are interested in checking non-reachability of single program states, executing TMR also suffices: for $(g, \bar{l}) \in \mathrm{State}$, if $(g, \bar{l}_t) \notin L_{g,t}$ for some $t\in n$, then the state (g, l) is unreachable from the initial ones. Executing TMR on a RAM with logarithmic-cost measure can be achieved in $\mathcal{O}\big(n(|\mathrm{init}| + |\mathrm{Glob}|^4|\mathrm{Frame}|^5)(L(|\mathrm{init}|) + L(n) + L(|\mathrm{Glob}|) + L(|\mathrm{Frame}|))\big)$ time, where $L(x)$ is the length of the binary representation of $x\in\mathbb{N}_0$ (see § H.4). With rigorous definitions of the input the running time is $\mathcal{O}((\text{input length})^2 L(\text{input length}))$.

If we wish to prove more general invariants, we construct a finite automaton for (1) as follows. First, we make the state spaces of the automata accepting $L_{g,t}$ disjoint

$((g,t) \in \mathrm{Glob}{\times}\mathrm{Loc})$, obtaining automata $\tilde{A}_{g,t}$ $((g,t) \in \mathrm{Glob}{\times}\mathrm{Loc})$. If we wish to obtain a deterministic automaton at the end, we additionally determinize all $\tilde{A}_{g,t}$ $((g,t) \in \mathrm{Glob}{\times}\mathrm{Loc})$. Second, for each $g{\in}\mathrm{Glob}$, chain $\tilde{A}_{g,t}$ for $t{<}n$ to accept exactly the words of the form $w_0\dagger \ldots \dagger w_{n-1}$ over $\mathrm{Frame}\dot{\cup}\{\dagger\}$ (where $\dagger{\notin}\mathrm{Frame}$ is a fresh symbol separating the local parts) such that $(w_t)_{t<n} \in \prod_{t<n} L_{g,t}$. Third, introduce a single initial state that dispatches different g to $\tilde{A}_{g,0}$ ($g{\in}\mathrm{Glob}$). Thus, (1) can be viewed as a regular language. The nondeterministic, ε-free automaton can be constructed (including executing TMR) in the same $\mathcal{O}\big(n(|\mathrm{init}| + |\mathrm{Glob}|^4|\mathrm{Frame}|^5)(L(|\mathrm{init}|) + L(n) + L(|\mathrm{Glob}|) + L(|\mathrm{Frame}|))\big)$ asymptotic time.

For our running example, we transform the left automaton from Fig. 1 into four automata $\tilde{A}_{0,0}$–$\tilde{A}_{3,0}$ accepting $L_{0,0}$–$L_{3,0}$ and the right automaton into four automata $\tilde{A}_{0,1}$–$\tilde{A}_{3,1}$ accepting $L_{0,1}$–$L_{3,1}$. We combine them into a nondeterministic finite automaton for (1) as follows (only the reachable part is shown):



In this graphical representation, the disjoint copies carry the same node labels, and the final states of $\tilde{A}_{0,1}$–$\tilde{A}_{3,1}$ have been merged to a unique accepting state. (Certainly, much more minimization is possible, mimicking sharing in BDDs—which is an interesting topic by itself but not our goal here.)

Theorem 1. *The inference system TMR is equivalent to multithreaded Cartesian abstract interpretation. Formally:*

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) \;=\; \bigcup_{g\in\mathrm{Glob}} \{g\} \times \prod_{t\in n} L_{g,t}\,.$$

Indeed, in our running example, the multithreaded-Cartesian collecting semantics corresponds to the language accepted by the above automaton.

The following §§ 6–7 will be devoted to proving Theorem 1.

6 Model-Checking General Multithreaded Programs

As an intermediate step in proving equivalence of multithreaded Cartesian abstract interpretation and TMR we are going to show another, simpler inference-system for proving properties of multithreaded programs. This inference system (up to names of variables and sets) is due to Flanagan and Qadeer [12]. Its definition does not depend on the internal structure of Loc and $\leadsto_t$ $(t{\in}n)$.

Let us define sets $\widetilde{R}_t \subseteq \mathrm{Glob}{\times}\mathrm{Loc}$ and $\widetilde{G}_t \subseteq \mathrm{Glob}^2$ for all $t{\in}n$ by the following inference system FQ:

$$\text{(FQ INIT)}\ \frac{(g,l)\in\mathrm{init}}{(g,l_t)\in\widetilde{R}_t}\ t{\in}n \qquad \text{(FQ STEP)}\ \frac{(g,l)\in\widetilde{R}_t \qquad (g,l)\leadsto_t (g',l')}{(g',l')\in\widetilde{R}_t \qquad (g,g')\in\widetilde{G}_t}\ t{\in}n$$

$$\text{(FQ ENV)}\ \frac{(g,g')\in\widetilde{G}_t \qquad (g,l)\in\widetilde{R}_s}{(g',l)\in\widetilde{R}_s}\ s{\neq}t \text{ are in } n$$

For finite-state programs, the families $\widetilde{R}$ and $\widetilde{G}$ can be generated in polynomial time. The algorithm is sound independently of finiteness, e.g., also for recursive programs.

One can show (Cor. 21), roughly speaking, that multithreaded-Cartesian abstract interpreration is equivalent to FQ. We will use FQ intermediately, showing FQ $\approx$ TMR.

7 Proof of Theorem 1

We show a semi-formal, high-level proof outline; rigorous details are found in § G.

We start by defining

$$R_t \;=\; \{(g,w)\in\mathrm{Glob}{\times}\mathrm{Loc} \mid g \xrightarrow[t]{w}{}^{*} \mathfrak{f}\} \qquad (t\in n).$$

Informally, the set R_t contains exactly the thread states of the thread t in the invariant denoted by TMR $(t{<}n)$.

Now let $G=(G_t)_{t\in n}\in(\mathfrak{P}(\mathrm{Glob}^2))^n$ and $R=(R_t)_{t\in n}\in(\mathfrak{P}(\mathrm{Glob}{\times}\mathrm{Loc}))^n$.

For our running example,

$$\begin{aligned}
R_0 &= \begin{pmatrix} \{0\}\times(\{\mathrm{A}x,\mathrm{D}x|x{\in}\{\mathrm{B,C}\}^*\}\cup\{\mathrm{B,C}\}^+)\cup\{1\}\times\{\mathrm{AB}x|x{\in}\{\mathrm{B,C}\}^*\} \\ \cup\,\{3\}\times(\{\mathrm{D}x \mid x{\in}\{\mathrm{B,C}\}^*\}\cup\{\mathrm{B,C}\}^+)\ \ \cup\{2\}\times\{\mathrm{AC}x|x{\in}\{\mathrm{B,C}\}^*\}\end{pmatrix} = \widetilde{R}_0\\
G_0 &= \{(0,1),(0,2),(0,3),(0,0),(3,3)\} \;=\; \widetilde{G}_0\\
R_1 &= \{0,1,2,3\}\times(\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B,C}\}^*\}\cup\{\mathrm{B,C}\}^+) \;=\; \widetilde{R}_1\\
G_1 &= \{(1,0),(2,0),(3,0),(0,0),(1,1),(2,2),(3,3)\} \;=\; \widetilde{G}_1
\end{aligned}$$

The equality between the sets generated by TMR and the sets generated by FQ is striking. We will show that it is not by coincidence, essentially proving

$$(\text{result of FQ } =)\ (\widetilde{R},\widetilde{G}) \;=\; (R,G)\ (= \text{ result of TMR}). \tag{2}$$

This equality will directly imply Thm. 1.

So let $\preceq$ be the componentwise partial order on $(\mathfrak{P}(\mathrm{Glob}{\times}\mathrm{Loc}))^n\times(\mathfrak{P}(\mathrm{Glob}^2))^n$:

$$(\hat{R},\hat{G})\preceq(\hat{R}',\hat{G}') \overset{\text{def}}{\iff} \forall\, t{<}n\colon \hat{R}_t{\subseteq}\hat{R}'_t \wedge \hat{G}_t{\subseteq}\hat{G}'_t\,.$$

Intuitively, we prove (2) by separating the equality into two componentwise inclusions: soundness (if a safety property holds according to TMR, then the strongest multithreaded-Cartesian invariant implies this property) and completeness (every safety property implied by the strongest multithreaded-Cartesian invariant can be proven by TMR).

The soundness proof will be conceptually short and the completeness proof a bit more intricate, building on ideas from post-saturation of pushdown systems.

7.1 Soundness: left componentwise inclusion in (2)

The crucial step is showing that the result of TMR is closed under FQ:

$$\text{(result of FQ } =) \ (\widetilde{R}, \widetilde{G}) \ \preceq \ (R, G) \ (= \text{ result of TMR}) \qquad \text{(proven in Lemma 22)}\,.$$

More precisely, the proof goes by applying FQ once to (R, G), thereby obtaining $(\check{R}, \check{G})$, and showing $(\check{R}, \check{G}) \preceq (R, G)$ componentwise. Internally, it amounts to checking that elements in $(\check{R}, \check{G})$ produced by FQ can also be produced by TMR.

7.2 Completeness: right componentswise inclusion in (2)

For each thread t we define its *operational semantics with FQ-context* as the transition relation of thread t in which the thread can additionally change the shared state according to the guarantees defined by FQ:

$$\overset{\widetilde{G}}{\leadsto}_t \ := \ \leadsto_t \ \cup \ \{((g, w), (g', w)) \mid w \in \mathrm{Loc} \wedge \exists\, s \in n \backslash \{t\} \colon (g, g') \in \widetilde{G}_s\} \qquad (t < n)\,.$$

Let $\overset{\widetilde{G}}{\leadsto}{}^*_t$, the *bigstep operational semantics with FQ-context*, be the reflexive-transitive closure of $\overset{\widetilde{G}}{\leadsto}_t$ on the set of thread states $(t < n)$.

Now we examine the system TMR. We view the relation (edge set) $\to$ defined by TMR as an element of $(\mathfrak{P}(V \times \mathrm{Frame} \times V))^n$ (where $v \xrightarrow[t]{a} v'$ means $(v, a, v') \in {\to}(t)$).

One can obtain G and $\to$ inductively by generating iterates $((\underset{t}{\to}_i)_{t<n}, (G_{t,i})_{t<n})$ of the derivation operator of TMR for $i \in \mathbb{N}_0$ (the right index i meaning the iterate number). More precisely, we start with empty sets $G_{t,0}$ and $\underset{t}{\to}_0$ for all $t<n$ and obtain $G_{t,i+1}$ and $\underset{t}{\to}_{i+1}$ for all $t<n$ by applying the rules of TMR exactly once to $G_{t,i}$ and $\underset{t}{\to}_i$ for all $t<n$. The described sequence of iterates is ascending, and each element derived by TMR has a derivation tree of some finite depth i:

$$\underset{t}{\to} \ = \ \bigcup_{i \in \mathbb{N}_0} \underset{t}{\to}_i \qquad \text{and} \qquad G_t \ = \ \bigcup_{i \in \mathbb{N}_0} G_{t,i} \qquad\qquad (t < n).$$

Sloppily speaking, the derivation operator of TMR produces graphs on V, and larger iterates contain larger graphs. Given a walk in an edge set $\underset{t}{\to}_i$, it in general has some "new" edges not present in the prior iterate $i-1$. Different walks connecting the same pair of nodes and carrying the same word label may have a different number of new edges. We let $\mathrm{tr}(t, i, v, \bar{v}, w)$ be the minimal number of new edges in iterate i in walks labeled by w from v to $\bar{v}$ in the edge set $\underset{t}{\to}_i$. (See Def. 27.)

After these preparations, we create a connection between FQ and TMR. Informally, we show: (i) stack words accepted by the automata created by TMR, together with the corresponding shared state, lie in the sets defined by FQ; (ii) prefixes of such words correspond to ongoing procedure calls as specified by the bigstep operational semantics with FQ-context; (iii) the shared state changes defined by TMR are also defined by FQ.

These claims are proven together by nested induction on the iterate number (outer induction) and the number of new edges tr(...) (inner induction). Formally, we show:

Lemma 2. *For all $i \in \mathbb{N}_0$ and all $j \in \mathbb{N}_0$ we have:*
(i) $\forall\, g{\in}\mathrm{Glob}, t{\in}n, w{\in}\mathrm{Frame}^*:\ \ \mathrm{tr}(t,i,g,\mathfrak{f},w) = j \ \Rightarrow\ (g,w) \in \widetilde{R}_t,$
(ii) $\forall g,\bar{g}{\in}\mathrm{Glob}, t{\in}n, b{\in}\mathrm{Frame}, w{\in}\mathrm{Frame}^*: \mathrm{tr}(t,i,g,(\bar{g},b),w){=}j \Rightarrow (\bar{g},b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g,w),$
(iii) $\forall\, t \in n:\ \ G_{t,i} \subseteq \widetilde{G}_t.$

The formal proof proceeds by double induction on (i,j).

Parts (i) and (iii) directly imply that the result of FQ is closed under TMR:

$$(\text{result of FQ } =)\ (\widetilde{R},\widetilde{G})\ \succeq\ (R,G)\ (= \text{ result of TMR}) \qquad (\text{proven in Lemma 28})\,.$$

7.3 Combining the left and right inclusions

FQ describes the abstract semantics exactly, whence we obtain:

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) \ = \bigcup_{g\in\mathrm{Glob}} \{g\} \times \prod_{t\in n} L_{g,t}\,,$$

where "$\subseteq$" follows from § 7.1 (cf. Prop. 23), and "$\supseteq$" follows from § 7.2 (cf. Prop. 29).

8 Conclusion

We considered the multithreaded-Cartesian approximation, which is a succinct description of the accuracy of the thread-modular approaches of Owicki and Gries, C. B. Jones, and Flanagan and Qadeer (without auxiliary variables). We applied it to multithreaded programs with recursion, presenting an algorithm for discovering a representation of the multithreaded-Cartesian collecting semantics. The algorithm creates a finite automaton whose language coincides with the multithreaded-Cartesian collecting semantics. In particular, the involved inductive invariant is shown to be a regular language. The algorithm uses ideas from a seminal algorithm of Flanagan and Qadeer and works in time $\mathcal{O}(n \log_2 n)$ in the number of threads n and polynomial in other quantities. We remark that, in contrast, the model-checking problem (without abstraction) is known to be undecidable.

While multithreaded programs with recursion occur rarely in practice, the models may contain both concurrency and recursion [9]. For example, in certain cases it is possible to model integer variables as stacks. But even for multithreaded programs whose procedures are nonrecursive, our algorithm TMR offers compact representation of stack contents, which depends only on the number of threads as well as on the sizes of shared states and frames, but not on the stack depth. A useful consequence of equivalence between FQ and TMR is that one may choose inlining procedures or creating an automaton depending on the costs of constructing and running an analysis, well knowing that its precision will not change. This opens way to potential time and space savings without changing the strength of an analysis.

Acknowledgment. Besides Neil Deaton Jones, I acknowledge comments and suggestions from Laurent Mauborgne and Xiuna Zhu. This work was executed partially at IMDEA Software, Spain and partially at Software & Systems Enrineering Research Group, led by Prof. Broy. I acknowledge the financial support of the projects of the German Federal Ministry for Education and Research, IDs 01IS11035 and 01IS13020.

References

1. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Symposium on Theory of Computing, 2004*, pages 202–211. ACM, 2004.
2. Nils Andersen and Neil Deaton Jones. Generalizing Cook's transformation to imperative stack programs. In Juhani Karhumäki, Hermann A. Maurer, and Grzegorz Rozenberg, editors, *Results and Trends in Theoretical Computer Science, Colloquium in Honor of Arto Salomaa, Graz, Austria, June 10-11, 1994, Proceedings*, volume 812 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1994.
3. Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In Masami Ito and Masafumi Toyama, editors, *Developments in Language Theory, 12th International Conference, Kyoto, Japan, September 16–19, 2008. Proceedings*, volume 5257 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2008.
4. Bruno Blanchet. Introduction to abstract interpretation. Lecture script, `http://bblanche.gitlabpages.inria.fr/absint.pdf`, 2002.
5. Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science*, 14(4):551–582, 2003.
6. Laura Bozzelli, Salvatore La Torre, and Adriano Peron. Verification of well-formed communicating recursive state machines. *Theoretical Computer Science*, 203(2–3):382–405, August 2008.
7. Julius Richard Büchi. Regular canonical systems. *Archiv für mathematische Logik und Grundlagenforschung*, 6:91–111, 1962/63.
8. Olaf Burkart and Bernhard Steffen. Pushdown processes: Parallel composition and model checking. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR '94, Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 22-25, 1994, Proceedings*, volume 836 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 1994.
9. Sagar Chaki, Edmund Melson Clarke, Nicholas Kidd, Thomas William Reps, and Tayssir Touili. Verifying concurrent message-passing C programs with recursive calls. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 334–349, 2006.
10. Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.
11. Radhia Cousot. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles.* PhD thesis, Institut national polytechnique de Lorraine, 1985. §4.3.2.4.3, pages 4-118, 4-119, 4-120.
12. Cormac Flanagan and Shaz Qadeer. Thread-modular model checking. In Thomas Ball and Sriram K. Rajamani, editors, *Model Checking Software, 10th International SPIN Workshop*, volume 2648 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 2003.
13. Torben Amtoft Hansen, Thomas Nikolajsen, Jesper Larsson Träff, and Neil Deaton Jones. Experiments with implementations of two theoretical constructions. In Albert Ronald da Silva Meyer and Михаил Абрамович Тайцлин, editors, *Logic at Botik '89, symposium on Logical Foundations of Computer Science, Переславль-Залесский, USSR, July 3–8, 1989, Proceedings*, volume 363 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1989.
14. Cliff Bryn Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983.
15. Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. A unifying approach for multistack pushdown automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger,

and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, Budapest, Hungary, August 25–29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 377–389. Springer, 2014.

16. Alexander Malkis. *Cartesian abstraction and verification of multithreaded programs.* PhD thesis, Albert-Ludwigs-Universität Freiburg, 2010.
17. Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko. Thread-modular verification is Cartesian abstract interpretation. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *ICTAC'06*, volume 4281 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2006.
18. Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1984.
19. Torben Ægidius Mogensen. WORM-2DPDAs: An extension to 2DPDAs that can be simulated in linear time. *Inf. Process. Lett.*, 52(1):15–22, 1994.
20. Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Normalizable Horn clauses, strongly recognizable relations, and spi. In Manuel V. Hermenegildo and Germán Puebla, editors, *Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 20–35, Madrid, Spain, September 2002. Springer.
21. Susan Speer Owicki. *Axiomatic proof techniques for parallel programs.* PhD thesis, Cornell University, department of computer science, TR 75-251, July 1975.
22. Shaz Qadeer, Sriram K. Rajamani, and Jakob Rehof. Summarizing procedures in concurrent programs. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 245–255, New York, NY, USA, 2004. ACM.
23. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 2005*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.
24. Ganesan Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, March 2000.
25. Stefan Schwoon. *Model-checking pushdown systems.* Ph.D. thesis, Technische Universität München, June 2002.
26. Michael Sipser. *Introduction to the theory of computation.* Cengage learning, third edition, 2013.
27. Alfred Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

Appendix

In the appendix we recollect or generalize some known results in order theory and abstract interpretation and prove new results concerning multithreaded Cartesian abstract interpretation, supporting the claims of the main part of the paper. We assume that the reader is acquainted with a version of the Knaster-Tarski fixpoint theorem and a version of the pumping lemma for context-free languages. Up to these two results, the appendix is meant to be relatively self-contained and rigorous, accompanying formal proofs by informal ideas in long or unclear cases. A reader who feels comfortable with particular claims is welcome to skip their proofs.

Throughout the appendix, an exclamation mark over a relation sign, as in $A \overset{!}{\subseteq} B$, indicates that the containing claim is not yet proven but will be proven next.

For a (potentially partial) map $f : X \dashrightarrow Y$, we write

$$\mathrm{dom}\ f \ = \ \{x \mid \exists\, y\colon (x,y) \in f\}$$

for the *domain* of f, and

$$\mathrm{img}\ f \ = \ \{y \mid \exists\, x\colon (x,y) \in f\}$$

for the *image* of f.

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

For maps $f : Y{\to}Z$ and $g : X{\to}Y$ we write $f{\circ}g : X{\to}Z$ for the *function composition* "f after g":

$$(f \circ g)(x) \ = \ f(g(x)) \qquad (x \in X)\,.$$

This composition is sometimes called *right composition*, referring to the fact that the map to the right of $\circ$ is applied first.

For binary relations A and B we denote the *left composition* of A with B by

$$A \mathbin{⨟} B \ = \ \{(x,z) \mid \exists\, y\colon (x,y) \in A \,\wedge\, (y,z) \in B\}\,.$$

"Left" refers to the fact that the relation to the left of the thick open semicolon is applied first. Sometimes the left composition is called *relational composition* or *diagrammatic composition* (referring to the way the diagram $\pi_1(A) \xrightarrow{A} \pi_2(A){\cap}\pi_1(B) \xrightarrow{B} \pi_2(B)$ is drawn, where π_i is the projection on the i^{th} component).

A Reflexive-Transitive Hulls and Partial Orders

The results of this section concern general mathematics and are well known. They are stated here for completeness and for being in exactly the form we need them in the main paper.

Proposition 3. *Let X be an arbitrary set and B a binary relation on X. For $i \in \mathbb{N}_+$ let S^i be inductively defined by*

$$\begin{aligned} S^1 &= \mathrm{id}_X \cup B\,, \\ S^i &= S^1 \mathbin{⨟} S^{i-1} \qquad \textit{for } i \geq 2\,. \end{aligned}$$

Then $S = \bigcup_{i\in\mathbb{N}_+} S^i$ is the reflexive-transitive closure of B on X.

Proof.

S is reflexive on X: The reason is that S contains $\mathrm{id}_X \subseteq S^1$.

S is transitive:

Notice that the sequence is isotone: for $i \in \mathbb{N}_+$ we have $S^i \subseteq S^{i+1}$.

Now we are going to show that $\forall\, i, j \in \mathbb{N}_+ \colon S^i ⨾ S^j \subseteq S^{i+j}$. We perform induction on i.

So let $i, j \in \mathbb{N}_+$ be arbitrary; we assume that $\forall\, \hat{\imath}{\in}\mathbb{N}_+ \colon \hat{\imath}{<}i \Rightarrow \forall\, j{\in}\mathbb{N}_+ \colon S^{\hat{\imath}} ⨾ S^j \subseteq S^{\hat{\imath}+j}$.

Case $i = 1$. By definition.

Case $i > 1$. Let $(a, b) \in S^i$ and $(b, c) \in S^j$. There is d such that $(a, d) \in S^1$ and $(d, b) \in S^{i-1}$. By induction hypothesis, $(d, c) \in S^{i-1+j}$. By definition, $(a, c) \in S^{i+j}$.

Every reflexive-transitive binary relation $\hat{S} \subseteq \text{Glob}{\times}\text{Loc}$ that is a superset of B contains S:

We will show that $\forall\, i{\in}\mathbb{N}_+ \colon S^i \subseteq \hat{S}$ by induction. So let $i \in \mathbb{N}_+$ be arbitrary and $\forall\, \hat{\imath} \in \mathbb{N}_+ \colon \hat{\imath} < i \Rightarrow S^{\hat{\imath}} \subseteq \hat{S}$.

Case $i = 1$. Since $S^1 = \text{id}_X \cup B \subseteq \hat{S}$.

Case $i > 1$. Since $S^i = S^1 ⨾ S^{i-1} \subseteq$ [by induction hypothesis] $\hat{S} ⨾ \hat{S} \subseteq$ [transitivity] $\hat{S}$.

□

Given posets $(X, \leq_X)$ and $(Y, \leq_Y)$, a map $f \colon X \to Y$ is called *isotone* if $\forall\, a, b \in X \colon a \leq_X b \Rightarrow f(a) \leq_Y f(b)$.

A *prefix point* (resp. *postfix point*, *fixpoint*)[1] of a map $f \colon X \to X$ on a poset $(X, \leq)$ is any $x {\in} X$ such that $f(x) \leq x$ (resp. $x \leq f(x)$, $f(x) = x$). We write $\text{prefp}\, f$ for the set of prefix points of f.

A poset is called a *complete lattice* if the supremum and the infimum of every subset exist.

Theorem 4 (Knaster-Tarski). *Let X be a complete lattice and $f \colon X \to X$ isotone. Then the least fixpoint of f exists and is equal to* $\inf(\text{prefp} f)$.

This fact, known as a part of the fixpoint theorem of Tarski, is proven, e.g., in Theorem 1 in [27].

In case of existence, we will write $\text{lfp}\, f$ for the least fixpoint of f.

Proposition 5. *The least-fixpoint operator on isotone maps of a complete lattice is isotone.*
Formally: If $(X, \leq)$ is a complete lattice and $f, g \colon X \to X$ isotone such that f is less than or equal to g pointwise, then $\text{lfp}\, f \leq \text{lfp}\, g$.

Proof. Let $(X, \leq)$ be a complete lattice, $f, g \colon X \to X$ isotone, and $\forall\, x \in X \colon f(x) \leq g(x)$. Then $\text{prefp}\, f \supseteq \text{prefp}\, g$. Then $\inf(\text{prefp}\, f) \leq \inf(\text{prefp}\, f)$. By Theorem 4, $\text{lfp}\, f \leq \text{lfp}\, g$. □

[1] The terminology varies. While earlier publications [10, 16] contained swapped definitions of prefix and postfix points, the viewpoint of the present paper is to refer to the location of the symbol f: since f comes *before* the inequality sign in the formula "$f(x) \leq x$", such x is called a *pre*fix point.

B Galois Connections

The results of this section concern isotone Galois connections and are mostly well known, perhaps with an exception of Prop. 10. They are stated here again for completeness and for being in exactly the form we need them in the main paper.

Given posets $(X, \leq_X)$ and $(Y, \leq_Y)$, a pair of maps (α, γ) such that $X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} Y$ is called an *isotone Galois-connection*, or simply *Galois connection*[2], iff

$$\forall\, x \in X,\, y \in Y: \quad \alpha(x) \leq_Y y \;\Leftrightarrow\; x \leq_X \gamma(y)\,.$$

The map α is called the *abstraction* map, the map γ is called the *concretization* map.

Proposition 6. *The maps of a Galois connection are isotone. Formally:*
If $(X, \leq_X) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (Y, \leq_Y)$ *is a Galois connection between posets, then* α *and* γ *are isotone.*

This lemma is proven, e.g., in Prop. 2 in [4].

Proof. Let $x, x' \in X$ such that $x \leq_X x'$. Since $\alpha(x') \leq_Y \alpha(x')$, we obtain $x' \leq_X \gamma(\alpha(x'))$, so $x \leq_X \gamma(\alpha(x'))$. Thus $\alpha(x) \leq_Y \alpha(x')$.

Let $y, y' \in Y$ such that $y \leq_Y y'$. Since $\gamma(y) \leq_X \gamma(y)$, we obtain $\alpha(\gamma(y)) \leq_Y y$, so $\alpha(\gamma(y)) \leq_Y y'$. Thus $\gamma(y) \leq_X \gamma(y')$. □

Proposition 7. *The abstraction map of a Galois connection between complete lattices is a join-morphism.*

This results is proven, e.g., in Prop. 3 in [4] (up to typos).

Proof. Let $(X, \leq_X) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (Y, \leq_Y)$ be a Galois connection between complete lattices.

Let $S \subseteq X$; we show $\alpha(\sup S) = \sup(\alpha(S))$.

"$\leq_Y$:" Let $s \in S$. Then $\alpha(s) \leq_Y \sup\{\, \alpha(x) \mid x \in S \,\} = \sup(\alpha(S))$. By definition of a Galois connection, $s \leq_X \gamma(\sup(\alpha(S)))$.
Thus, $\sup S \leq_X \gamma(\sup(\alpha(S)))$. By definition of a Galois connection, $\alpha(\sup S) \leq_Y \sup(\alpha(S))$.

"$\geq_Y$:" By Prop. 6, α is isotone, so $\alpha(\sup S)$ is an upper bound for the set $\{\alpha(x) | x{\in}S\}$, whence $\alpha(\sup S) \geq_Y \sup\{\, \alpha(x) \mid x \in S \,\}$.

□

A map $f : X \to X$ on a poset $(X, \leq)$ is called
- *extensive* iff $\forall\, x \in X : x \leq \rho(x)$,
- *reductive* iff $\forall\, x \in X : \rho(x) \leq x$,
- *idempotent* iff $\rho \circ \rho = \rho$.

Proposition 8. *Let* $(X, \leq_X) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (Y, \leq_Y)$ *be a Galois connection between posets. Then* $\gamma \circ \alpha$ *is extensive and* $\alpha \circ \gamma$ *is reductive.*

[2] In the literature, there is also a different notion of so-called *antitone* Galois-connections (a pair of maps (f, g) such that $X \underset{f}{\overset{g}{\leftrightarrows}} Y$ and $\forall\, x{\in}X, y{\in}Y:\; y \leq_Y f(x) \;\Leftrightarrow\; x \leq_X g(y)$; then both maps can be shown to be order-reversing). In this paper we consider only *isotone* Galois-connections, in which both maps are order-preserving, and drop the prefix *isotone* for brevity.

This results is proven, e.g., in Prop. 2 in [4].

Proof. Since $\alpha(x) \leq_Y \alpha(x)$, the definition of a Galois connection implies $x \leq_X \gamma(\alpha(x))$ $(x \in X)$.

Since $\gamma(y) \leq_X \gamma(y)$, the definition of a Galois connection implies $\alpha(\gamma(y)) \leq_Y y$ $(y \in Y)$. □

An *upper closure operator* on a poset $(X, \leq)$ is a map $\rho : X \to X$ which is isotone, extensive, and idempotent.

Proposition 9. *Let* $(X, \leq_X) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} (Y, \leq_Y)$ *be a Galois connection between posets. Then* $\gamma \circ \alpha$ *is an upper closure operator on* $(X, \leq_X)$.

This results is proven, e.g., in Prop. 1.8.1 in [16].

Proof. Let $\rho = \gamma \circ \alpha$. By Prop. 6, ρ is isotone. By Prop. 8, ρ is extensive. To show idempotence, let $x \in D$. Extensivity implies $\rho(x) \leq_X \rho(\rho(x))$. Now notice that $\rho \circ \rho(x) = (\gamma \circ \alpha) \circ (\gamma \circ \alpha)(x) = \gamma(\underbrace{\alpha \circ \gamma}_{\text{reductive}}(\alpha(x))) \leq_X \gamma(\alpha(x)) = \rho(x)$. □

Proposition 10. *Let* $(X, \leq_X) \overset{\gamma}{\underset{\alpha}{\leftrightarrows}} (Y, \leq_Y)$ *be a Galois connection between complete lattices, and let* $f : X \to X$ *be isotone. Then* $\mathrm{lfp}(\gamma \circ \alpha \circ f) = \gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma))$.

This result is proven in, e.g., Theorem 1.8.2 in [16]. The proof we present here is simpler. Notice that the functions $\gamma \circ \alpha \circ f$ and $\alpha \circ f \circ \gamma$ are isotone and that the fixpoints exist by Theorem 4.

Proof. "$\leq_X$": Since

$$(\gamma \circ \alpha \circ f)(\gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma))) = \gamma((\alpha \circ f \circ \gamma)(\mathrm{lfp}(\alpha \circ f \circ \gamma))) = \\ [\text{definition of a fixpoint}]\ \gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma)),$$

$\gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma))$ is a fixpoint of $\gamma \circ \alpha \circ f$; thus $\gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma))$ is greater than or equal to the least fixpoint of $\gamma \circ \alpha \circ f$.

"$\geq_X$": Notice that $(\alpha \circ f \circ \gamma)\big((\alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f))\big) = (\alpha \circ f)\big((\gamma \circ \alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f))\big)$ = [definition of a fixpoint] $(\alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f))$. Thus, $(\alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f)) \geq_Y \mathrm{lfp}(\alpha \circ f \circ \gamma)$. Then $\mathrm{lfp}(\gamma \circ \alpha \circ f) = (\gamma \circ \alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f)) = \gamma((\alpha \circ f)(\mathrm{lfp}(\gamma \circ \alpha \circ f)))$ = [since concretizations are isotone by Prop. 6] $\geq_X \gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma))$.

□

C Inference rules

In this section we give precise meaning to a set of inference rules. The results are folklore, but here we state them in exactly the form which we will need later. Compared to § 1.2 in [16], we remove a part of finiteness assumptions here.

Sometimes sets are defined inductively by means of inference rules like

$$\frac{a \in X_0 \qquad b \in X_1}{b \in X_0 \qquad c \in X_2} \qquad \frac{c \in X_2 \qquad d \in X_2}{c \in X_1} \qquad \dots \tag{3}$$

where X_0, X_1, X_2 are set variables and a, b, c, d are constants. A rule scheme might be given which represents a collection of many (perhaps infinitely many) rules:

$$\frac{x \in X \qquad y \in Y}{xy \in Z} .$$

This rule scheme says that Z should contain the products of elements of X and Y.

Intuitively speaking, when each rule is interpreted as a constraint, the rules define the least set or set tuple that satisfies all the constraints. Now we formalize this notion.

Let us assume that for some index set I we already have fixed sets D_i $(i \in I)$ and we would like to define subsets $X_i \subseteq D_i$ $(i \in I)$ by means of rules. A *rule* is a triple $((A_i)_{i \in I}, k, x)$ where $A_i \subseteq D_i$ $(i \in I)$, $k \in I$, and $x \in D_k$. For example, the two rules from (3) are formalized as triples $((\{a\}, \{b\}, \emptyset), 0, b)$, $((\{a\}, \{b\}, \emptyset), 2, c)$ and $((\emptyset, \emptyset, \{c, d\}), 1, c)$ with $I = \{0, 1, 2\}$.

Consider the lattice $L := \prod_{i \in I} \mathfrak{P}(D_i)$ equipped with the componentwise order $\lesssim$ $:= \{((X_i)_{i \in I}, (X'_i)_{i \in I}) \in L^2 \mid \forall\, i \in I\colon X_i \subseteq X'_i\}$ and the least element $\bot = (\emptyset)_{i \in I}$.

Definition 11. *A* prefix point *of a set of rules R is a tuple $X = (X_i)_{i \in I}$ such that for any $(A, i, x) \in R$ where $A \lesssim X$ we have $x \in X_i$.*

A postfix point *of a set of rules R is a tuple $X = (X_i)_{i \in I}$ such that for any $i \in I$ and any $x \in X_i$ there is some $A \lesssim X$ such that $(A, i, x) \in R$.*

A fixpoint *of R is a pre- and postfix point of R.*

The derivation operator *of R is*

$$\begin{aligned} \mathbb{D}_R : L &\to L \\ X &\mapsto \big(\{\, x \in D_i \mid \exists\, A \lesssim X\colon (A, i, x) \in R \,\}\big)_{i \in I} . \end{aligned}$$

Proposition 12. *The prefix points (resp. postfix points, fixpoints) of a set of rules are exactly the prefix points (resp. postfix points, fixpoints) of its derivation operator.*

Proof. Follows from the definitions. □

The derivation operator of a set of rules is isotone; thus Theorem 4 implies the existence of the least fixpoint. The usual interpretation of a set of rules R is that they define the least fixpoint of $\mathbb{D}_R$.

Theorem 13. *If each rule has only finitely many premises, ω iterations suffice to get the least fixpoint. Formally:*

$$\left(\forall\, ((A_i)_{i \in I}, i, x) \in R \;\; \exists\, J \subseteq I\colon\; J \textit{ and } \bigcup_{j \in J} A_j \textit{ are finite } \wedge \bigcup_{i \in I \setminus J} A_i = \emptyset\right)$$
$$\Rightarrow \quad \operatorname{lfp} R \;=\; \sup\{\, \mathbb{D}_R^n(\bot) \mid n \in \omega \,\} .$$

We use the letter ω, denoting the least infinite ordinal, instead of $\mathbb{N}_0$ to emphasize the fact that we do not need *more* iterations, which would be the case without the finiteness assumption.

The result itself is folklore. For finite I, this result was proven, e.g., in Theorem 1.2.3 in [16].

Proof idea. From high-level view, $\mathbb{D}_R$ is Scott-continuous (i.e. preserves suprema of directed subsets), thus Kleene's fixpoint theorem can be applied. We give a direct proof here.

Proof. To show that $\sup\{\, \mathbb{D}_R^n(\bot) \mid n \in \omega \,\}$ is a prefix point, take any rule (A, k, x) of R where $A \lesssim \sup\{\, \mathbb{D}_R^n(\bot) \mid n \in \omega \,\}$. We write $(A_i)_{i\in I} = A$. There is some $J \subseteq I$ such that J and $\bigcup_{j\in J} A_j$ are finite and $\bigcup_{i\in I\setminus J} A_i = \emptyset$. Since for each $i \in I$ the set A_i is finite, the elements of A_i are in the union of finitely many iterates: $\forall\, i \in I\ \exists\, m(i) \in \omega \colon A_i \subseteq (\sup\{\, \mathbb{D}_R^n(\bot) \mid n \le m(i) \,\})_i$. Thus we may assume that $\forall\, i \in I \setminus J \colon m(i) = 0$. Since J is finite, with the convention that $\max\emptyset = 0$ we obtain that $\max\{m(i) \mid i \in J\} = \max\{m(i) \mid i \in I\}$ are both finite. Thus $A \lesssim \sup\{\, \mathbb{D}_R^n(\bot) \mid n \le \max\{m(i) \mid i \in J\} \,\}$. The derivation operator is isotone, so are its finite powers. Thus $\bot \lesssim \mathbb{D}_R(\bot)$ implies $\mathbb{D}_R^n(\bot) \lesssim \mathbb{D}_R^{n+1}(\bot)$ for all $n \in \omega$. Especially $\sup\{\mathbb{D}_R^n(\bot) \mid n \le \max\{m(i) | i \in I\}\} \lesssim \mathbb{D}_R^{\max\{m(i)|i\in I\}}(\bot)$. So $x \in \left(\mathbb{D}_R^{\max\{m(i)|i\in I\}+1}(\bot)\right)_k$.

To show that this prefix point is the least one, it suffices to show that it is less than or equal to any prefix point. So let $C = (C_i)_{i\in I}$ be any prefix point of R. We show by induction on n that $\forall\, n \in \omega \colon \mathbb{D}_R^n(\bot) \lesssim C$.

For $n = 0$ we have $\mathbb{D}_R^0(\bot) = \bot \lesssim C$.

Assume that the statement is proven for some $n \in \omega$, i.e. $\mathbb{D}_R^n(\bot) \lesssim C$. Take any $i \in I$; it suffices to show that $\{x \in D_i \mid \exists\, A \lesssim \mathbb{D}_R^n(\bot) \colon (A, i, x) \in R\} \subseteq C_i$. So let $x \in D_i$ such that there is some $A \lesssim \mathbb{D}_R^n(\bot)$ with $(A, i, x) \in R$. Then $A \lesssim C$. Since C is closed under the rule (A, i, x), we have $x \in C_i$.

So $\sup\{\, \mathbb{D}_R^n(\bot) \mid n \in \omega \,\}$ is the least prefix point of the derivation operator. By Tarski's fixpoint theorem 4 it is also the least fixpoint. $\square$

Thus each element of each set X_i of the least fixpoint has a finite derivation tree ($i \in I$).

D Multithreaded Programs: Example

In this section we compute the strongest inductive invariant of the running example.

Example 14 (Set of reachable states). Consider the program defined in § 3 on page 4. We compute now the set of reachable states.

We claim that

$$
\begin{aligned}
&\mathrm{lfp}(\lambda\, S \in D.\, \mathrm{init} \cup \mathrm{post}(S)) \\
&= \\
&\{0\} \times \left(\begin{array}{l}
\{(\mathrm{A}y, \mathrm{A}y), (\mathrm{D}y, \mathrm{D}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\ \{(\mathrm{D}y, z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ z \text{ is a suffix of } y\} \\
\cup\ \{(y, \mathrm{D}z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\} \\
\cup\ \{(y, z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z\ \vee\ z \text{ is a suffix of } y)\}
\end{array}\right) \\
&\cup\ \{1\} \times \{(\mathrm{AB}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
&\cup\ \{2\} \times \{(\mathrm{AC}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
&\cup\ \{3\} \times \left(\begin{array}{l}
\{(\mathrm{D}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\ \{(y, \mathrm{A}z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\}
\end{array}\right).
\end{aligned}
$$

We prove the equality by considering the left and right inclusions separately. We write LHS (left-hand side) for the "lfp ..." term before the equality sign and RHS (right-hand side) for the union of the products of the set terms after the equality sign.

"$\subseteq$": By Theorem 4, it suffices to prove that RHS is a prefix point of $\lambda\, S{\in}D.\, \mathrm{init} \cup \mathrm{post}(S)$, i.e., that $\mathrm{init} \cup \mathrm{post}(\mathrm{RHS}) \overset{!}{\subseteq} \mathrm{RHS}$. Notice that $\mathrm{init} = \{(0, (\mathrm{A}, \mathrm{A}))\} \subseteq \{0\} \times \{(\mathrm{A}x, \mathrm{A}x) \mid x \in \{\mathrm{B}, \mathrm{C}\}^*\} \subseteq \mathrm{RHS}$. Now consider an arbitrary $(g', (r', v')) \in \mathrm{post}(\mathrm{RHS})$. Then there is some $(g, (r, v)) \in \mathrm{RHS}$ such that $((g, r) \rightsquigarrow_0 (g', r') \wedge v{=}v') \vee ((g, v) \rightsquigarrow_1 (g', v') \wedge r{=}r')$.

Case $(g, r) \rightsquigarrow_0 (g', r') \wedge v = v'$.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}0 \wedge r = \mathrm{A}u \wedge g'{=}1 \wedge r' = \mathrm{AB}u$.
From $(g, (r, v)) \in \mathrm{RHS}$ we obtain $v = \mathrm{A}u$ and $u \in \{\mathrm{B}, \mathrm{C}\}^*$. Thus, $(g', (r', v')) = (1, (\mathrm{AB}u, \mathrm{A}u)) \in \mathrm{RHS}$.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}0 \wedge r = \mathrm{A}u \wedge g'{=}2 \wedge r' = \mathrm{AC}u$.
From $(g, (r, v)) \in \mathrm{RHS}$ we obtain $v = \mathrm{A}u$ and $u \in \{\mathrm{B}, \mathrm{C}\}^*$. Thus, $(g', (r', v')) = (2, (\mathrm{AC}u, \mathrm{A}u)) \in \mathrm{RHS}$.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}0 \wedge r = \mathrm{A}u \wedge g'{=}3 \wedge r' = \mathrm{D}u$.
From $(g, (r, v)) \in \mathrm{RHS}$ we obtain $v = \mathrm{A}u$ and $u \in \{\mathrm{B}, \mathrm{C}\}^*$. Thus, $(g', (r', v')) = (3, (\mathrm{D}u, \mathrm{A}u)) \in \mathrm{RHS}$.

Case there are $y \in \{\mathrm{B}, \mathrm{C}, \mathrm{D}\}, z \in \mathrm{Frame}$, $u \in \mathrm{Frame}^*$ such that $r = yzu \wedge g'{=}g \wedge r' = zu$. By $(g, (r, v)) \in \mathrm{RHS}$ we have one of the following six cases.

Case $g{=}0 \wedge r{=}v{=}\mathrm{D}r' \wedge r' {\in} \{\mathrm{B}{,}\mathrm{C}\}^*$. Then $(g', (r'{,}v')) = (0, (r'{,}\mathrm{D}r')) \in$ [since $r' = zu \in \{\mathrm{B}, \mathrm{C}\}^+$ and r' is a suffix of itself] RHS.

Case $g{=}0 \wedge y{=}\mathrm{D} \wedge zu, v \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge v$ is a suffix of zu.
Then $(g', (r', v')) = (0, (zu, v)) \in \mathrm{RHS}$.

Case there is some w such that $g{=}0 \wedge v = \mathrm{D}w \wedge r, w \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge r$ is a suffix of w. Then $(g', (r', v')) = (0, (zu, \mathrm{D}w)) \in$ [since zu is also a suffix of w] RHS.

Case $g = 0 \wedge r, v \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge$ (r is a suffix of $v \vee v$ is a suffix of r). If r is a suffix of v, then zu is also a suffix of v. If r is not a suffix of v, then they are unequal, and, in particular, v must be a strict suffix of $r = yzu$, so v must be a suffix of zu. In any case we have $(g', (r', v')) = (0, (zu, v)) \in \mathrm{RHS}$.

Case $g = 3 \wedge y = \mathrm{D} \wedge zu \in \{\mathrm{B}, \mathrm{C}\}^* \wedge v = \mathrm{A}zu$. Then $(g', (r', v')) = (3, (zu, \mathrm{A}zu)) \in$ [since $zu \neq \varepsilon$ and zu is a suffix of itself] RHS.

Case there is some w such that $g{=}3 \wedge v = \mathrm{A}w \wedge r, w \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge r$ is a suffix of w. Then $(g', (r', v')) = (3, (zu, \mathrm{A}w)) \in$ [since $zu \neq \varepsilon$ and zu is a suffix of w] RHS.

Case $(g, v) \rightsquigarrow_1 (g', v') \wedge r = r'$.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}1 \wedge v = \mathrm{A}u \wedge g'{=}0 \wedge v' = \mathrm{AB}u$.
From $(g, (r, v)) \in \mathrm{RHS}$ we obtain $u \in \{\mathrm{B}, \mathrm{C}\}^*$ and $r = \mathrm{AB}u$. Then $(g', (r', v')) = (0, (\mathrm{AB}u, \mathrm{AB}u)) \in$ [since $\mathrm{B}u \in \{\mathrm{B}, \mathrm{C}\}^*$] RHS.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}2 \wedge v = \mathrm{A}u \wedge g'{=}0 \wedge v' = \mathrm{AC}u$.
From $(g, (r, v)) \in \mathrm{RHS}$ we obtain $u \in \{\mathrm{B}, \mathrm{C}\}^*$ and $r = \mathrm{AC}u$. Then $(g', (r', v')) = (0, (\mathrm{AC}u, \mathrm{AC}u)) \in$ [since $\mathrm{C}u \in \{\mathrm{B}, \mathrm{C}\}^*$] RHS.

Case there is $u \in \mathrm{Frame}^*$ such that $g{=}3 \wedge v = \mathrm{A}u \wedge g'{=}0 \wedge v' = \mathrm{D}u$.
By $(g, (r, v)) \in \mathrm{RHS}$ one of the following two cases holds.
Case $r = \mathrm{D}u \wedge u \in \{\mathrm{B}, \mathrm{C}\}^*$. Then $(g', (r', v')) = (0, (\mathrm{D}u, \mathrm{D}u)) \in \mathrm{RHS}$.

Case $r, u \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge r$ is a suffix of u. Then $(g', (r', v')) = (0, (r, \mathrm{D}u)) \in \mathrm{RHS}$.

Case there are $y \in \{\mathrm{B}, \mathrm{C}, \mathrm{D}\}$, $z \in \mathrm{Frame}$, $u \in \mathrm{Frame}^*$ such that $g = g' \wedge v = yzu \wedge v' = zu$.

By $(g, (r, v)) \in \mathrm{RHS}$ we have $g = 0$ and one of the following four cases holds.

Case $y = \mathrm{D} \wedge r = \mathrm{D}zu \wedge zu \in \{\mathrm{B},\mathrm{C}\}^*$. Then $(g', (r', v')) = (0, (\mathrm{D}v', v')) \in$ [since $v' = zu \in \{\mathrm{B}, \mathrm{C}\}^+$ and v' is a suffix of itself] RHS.

Case there is some w such that $w, v \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge r = \mathrm{D}w \wedge v$ is a suffix of w. Then $(g', (r', v')) = (0, (\mathrm{D}w, zu)) \in$ [since $zu \in \{\mathrm{B}, \mathrm{C}\}^+$ and zu is a suffix of w] RHS.

Case $r, zu \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge y = \mathrm{D} \wedge r$ is a suffix of zu. Then $(g', (r', v')) = (0, (r, zu)) \in \mathrm{RHS}$.

Case $r, v \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge (r$ is a suffix of $v \vee v$ is a suffix of $r)$. If v is a suffix of r, then zu (which is a suffix of v) is also a suffix of r. Otherwise $v \neq r$ and r is a strict suffix of yzu; then r is a suffix of zu. Since $r' = r$ and $v' = zu$, we obtain that $r', v' \in \{\mathrm{B}, \mathrm{C}\}^+$ and (v' is a suffix of $r' \vee r'$ is a suffix of v'). Thus, $(g', (r', v')) = (0, (r', v')) \in \mathrm{RHS}$.

"$\supseteq$": We show that how to obtain all parts of RHS in the order given below.

(i) $\{0\} \times \{(\mathrm{A}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\}$: We prove by induction on the length of y that for all $y \in \{\mathrm{B}, \mathrm{C}\}^*$ we have $(0, (\mathrm{A}y, \mathrm{A}y)) \in \mathrm{LHS}$.

So let $y \in \{\mathrm{B}, \mathrm{C}\}^*$ be arbitrary, and we assume $\forall \bar{y} \in \{\mathrm{B}, \mathrm{C}\}^*\colon |\bar{y}| < |y| \Rightarrow (0, (\mathrm{A}\bar{y}, \mathrm{A}\bar{y})) \in \mathrm{LHS}$.

Case $y = \varepsilon$. Then $(0, (\mathrm{A}y, \mathrm{A}y)) = (0, (\mathrm{A}, \mathrm{A})) \in \mathrm{init} \subseteq \mathrm{LHS}$.

Case $y = \mathrm{B}\bar{y}$ for some $\bar{y}$. Then $|\bar{y}| < |y|$. The induction hypothesis implies $(0, (\mathrm{A}\bar{y}, \mathrm{A}\bar{y})) \in \mathrm{LHS}$. Due to the push transition $((0, \mathrm{A}), (1, \mathrm{A}, \mathrm{B})) \in \uplus_0$, we obtain $(0, \mathrm{A}\bar{y}) \rightsquigarrow_0 (1, \mathrm{AB}\bar{y})$, and so $(1, (\mathrm{AB}\bar{y}, \mathrm{A}\bar{y})) \in \mathrm{post}(\mathrm{LHS}) \subseteq \mathrm{LHS}$. Due to the push transition $((1, \mathrm{A}), (0, \mathrm{A}, \mathrm{B})) \in \uplus_1$, we obtain $(1, \mathrm{A}\bar{y}) \rightsquigarrow_1 (0, \mathrm{AB}\bar{y})$, and so $(0, (\mathrm{AB}\bar{y}, \mathrm{AB}\bar{y})) \in \mathrm{post}(\mathrm{LHS}) \subseteq \mathrm{LHS}$. Thus, $(0, (\mathrm{A}y, \mathrm{A}y)) \in \mathrm{LHS}$.

Case $y = \mathrm{C}\bar{y}$ for some $\bar{y}$. Then $|\bar{y}| < |y|$. The induction hypothesis implies $(0, (\mathrm{A}\bar{y}, \mathrm{A}\bar{y})) \in \mathrm{LHS}$. Due to the push transition $((0, \mathrm{A}), (2, \mathrm{A}, \mathrm{C})) \in \uplus_0$, we obtain $(0, \mathrm{A}\bar{y}) \rightsquigarrow_0 (2, \mathrm{AC}\bar{y})$, and so $(2, (\mathrm{AC}\bar{y}, \mathrm{A}\bar{y})) \in \mathrm{post}(\mathrm{LHS}) \subseteq \mathrm{LHS}$. Due to the push transition $((2, \mathrm{A}), (0, \mathrm{A}, \mathrm{C})) \in \uplus_1$, we obtain $(2, \mathrm{A}\bar{y}) \rightsquigarrow_1 (0, \mathrm{AC}\bar{y})$, and so $(0, (\mathrm{AC}\bar{y}, \mathrm{AC}\bar{y})) \in \mathrm{post}(\mathrm{LHS}) \subseteq \mathrm{LHS}$. Therefore, $(0, (\mathrm{A}y, \mathrm{A}y)) \in \mathrm{LHS}$.

(ii) $\{1\} \times \{(\mathrm{AB}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \cup \{2\} \times \{(\mathrm{AC}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\}$: Obtained from (i) by executing one more transition $((0,\mathrm{A}), (1,\mathrm{A},\mathrm{B}))$ or $((0,\mathrm{A}), (2,\mathrm{A},\mathrm{C}))$ of the left thread.

(iii) $\{3\} \times \{(\mathrm{D}y, \mathrm{A}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\}$: Obtained from (i) by executing the thread transition $((0, \mathrm{A}), (3, \mathrm{D}))$ of the left thread.

(iv) $\{0\} \times \{(\mathrm{D}y, \mathrm{D}y) \mid y \in \{\mathrm{B}, \mathrm{C}\}^*\}$: Obtained from (iii) by executing the thread transition $((3, \mathrm{A}), (0, \mathrm{D}))$ of the right thread.

(v) $\{3\} \times \{(y, \mathrm{A}z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge y$ is a suffix of $z\}$: Obtained from (iii) by executing the thread transition of the form $((3, \mathrm{D}, c), (3, c))$ of the left thread for a suitable $c \in \{\mathrm{B}, \mathrm{C}\}$, followed by zero or more thread transitions of the form $((3, b, c), (3, c))$ of the left thread for suitable $b, c \in \{\mathrm{B}, \mathrm{C}\}$.

(vi) $\{0\} \times \{(\mathrm{D}y, z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ z \text{ is a suffix of } y\}$: Obtained from (iv) by executing thread transitions of the form $((0, \mathrm{D}, c), (0, c))$ of the right thread for a suitable $c \in \{\mathrm{B}, \mathrm{C}\}$, followed by zero or more thread transitions of the form $((0, b, c), (0, c))$ of the right thread for suitable $b, c \in \{\mathrm{B}, \mathrm{C}\}$.

(vii) $\{0\} \times \{(y, \mathrm{D}z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\}$: Obtained from (iv) by executing the thread transition of the form $((0, \mathrm{D}, c), (0, c))$ of the left thread for a suitable $c \in \{\mathrm{B}, \mathrm{C}\}$, followed by zero or more thread transitions of the form $((0, b, c), (0, c))$ of the left thread for suitable $b, c \in \{\mathrm{B}, \mathrm{C}\}$.

(viii) $\{0\} \times \{(y, z) \mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z\ \vee\ z \text{ is a suffix of } y)\}$: Consider a state $(0, (y, z))$ such that $y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge (y$ is a suffix of $z \vee z$ is a suffix of $y)$. If y is a suffix of z, this state is obtained from (vii) by executing the transition $((0, \mathrm{D}, z_0), (0, z_0))$ of the right thread. Otherwise z is a strict suffix of y, and the state is obtained from (vi) by executing the transition $((0, \mathrm{D}, y_0), (0, y_0))$ of the left thread.

The claim is shown.

Let us now view the strongest inductive invariant as the set of words

$$
\begin{aligned}
L := \{&0\mathrm{A}y\dagger\mathrm{A}y,\ 0\mathrm{D}y\dagger\mathrm{D}y &&\mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\, \{&0\mathrm{D}y\dagger z &&\mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ z \text{ is a suffix of } y\} \\
\cup\, \{&0y\dagger\mathrm{D}z &&\mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\} \\
\cup\, \{&0y\dagger z &&\mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z\ \vee\ z \text{ is a suffix of } y)\} \\
\cup\, \{&1\mathrm{AB}y\dagger\mathrm{A}y &&\mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\, \{&2\mathrm{AC}y\dagger\mathrm{A}y &&\mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\, \{&3\mathrm{D}y\dagger\mathrm{A}y &&\mid y \in \{\mathrm{B}, \mathrm{C}\}^*\} \\
\cup\, \{&3y\dagger\mathrm{A}z &&\mid y, z \in \{\mathrm{B}, \mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\},
\end{aligned}
$$

where $\dagger \notin \mathrm{Frame}$ is a fresh literal separating the local parts of the two threads. We show that L is not context-free.

For that, we use the pumping lemma for context-free languages in the version of [26], Theorem 2.34, and adapt the proof from [26], Example 2.38. Assume for the purpose of contradiction that L is context-free. Let p be the pumping length and $s = 0\mathrm{AB}^p\mathrm{C}^p\dagger\mathrm{AB}^p\mathrm{C}^p$. Since $s \in L$, there are words $u, v, x, y, z \in (\mathrm{Glob} \cup \mathrm{Frame} \cup \{\dagger\})^*$ such that $s = uvxyz$,

(I) for each $i \in \mathbb{N}_0$ we have $uv^ixy^iz \in L$,

(II) $|vy| > 0$, and

(III) $|vxy| \leq p$.

By (I), $uv^5xy^5z \in L$. Since each word of L has at most four occurrences of symbols from $\mathrm{Glob} \cup \{\mathrm{A}, \mathrm{D}, \dagger\}$, (II) implies $vy \in \{\mathrm{B}, \mathrm{C}\}^+$. Thus, 0, † must occur once and A twice in uxz. By the definition of L,

$$\forall\, i \in \mathbb{N}_0\ \exists\, w \in \{\mathrm{B}, \mathrm{C}\}^*: \quad uv^ixy^iz = 0\mathrm{A}w\dagger\mathrm{A}w\,. \tag{4}$$

If † would lie in u, then uv^2xy^2z would have strictly more symbols from $\{\mathrm{B}, \mathrm{C}\}$ to the right of † than to its left, contradicting (4). If † would lie in z, then uv^2xy^2z would have strictly more symbols from $\{\mathrm{B}, \mathrm{C}\}$ to the left of † than to its right, also contradicting (4). So † lies in x. By the choice of s and (III), there are some $i, j \in \mathbb{N}_0$ such that $v = \mathrm{C}^i$, $y = \mathrm{B}^j$, and $i + j < p$. Then $uxz = 0\mathrm{AB}^p\mathrm{C}^{p-i}\dagger\mathrm{AB}^{p-j}\mathrm{C}^p$. By

(II), $i{+}j > 0$, so $\mathrm{B}^p\mathrm{C}^{p-i} \neq \mathrm{B}^{p-j}\mathrm{C}^p$. Thus, uxz is not in the form required by (4). Contradiction! Thus, our original assumption was wrong, and L is not context-free. □

E Multithreaded-Cartesian Abstract Interpretation for General Multithreaded Programs

Within this section we forget for a moment that $\rightsquigarrow_t$ $(t \in n)$ and Loc are not given but constructed. We consider arbitrary multithreaded programs, which are given by some set of shared states Glob, some set of local states Loc, and n transition relations $\rightsquigarrow_t \subseteq (\mathrm{Glob} \times \mathrm{Loc})^2$ for $t \in n$. The set of program states, the concrete domain, the successor map, and the multithreaded-Cartesian approximation are defined in terms of these sets as before.

The *abstract domain* of the multithreaded-Cartesian analysis is

$$D^{\#} \;=\; (\mathfrak{P}(\mathrm{Glob} \times \mathrm{Loc}))^n \,,$$

an abstract element being an n-tuple of sets of thread states.

The *multithreaded-Cartesian abstraction* is the map

$$\alpha_{\mathrm{mc}} : D \to D^{\#} \,, \quad S \mapsto (\{(g, l_t) \mid (g, l) \in S\})_{t \in n} \,,$$

which takes a set of program states S and returns the tuple in which the t^{th} component contains the projections of S to the shared part and to the part of thread t $(t{\in}n)$.

The *multithreaded-Cartesian concretization* is the map

$$\gamma_{\mathrm{mc}} : D^{\#} \to D \,, \quad (S_t)_{t \in n} \mapsto \{(g, l) \in \mathrm{State} \mid \forall\, t \in n \colon (g, l_t) \in S_t\} \,,$$

which takes a tuple $(S_t)_{t \in n}$ of sets of thread states and returns the set of all program states that can be built, loosely speaking, by combining thread states from the sets S_t $(t \in n)$ that have the same shared part.

Proposition 15. $\rho_{\mathrm{mc}} \;=\; \gamma_{\mathrm{mc}} \circ \alpha_{\mathrm{mc}}$.

Proof. Let $S \in D$. Then $\rho_{\mathrm{mc}}(S) = \{(g, l) \in \mathrm{State} \mid \forall\, t{\in}n\, \exists\, \hat{l}{\in}\mathrm{Loc}^n \colon (g, \hat{l}) \in S \,\wedge\, l_t = \hat{l}_t\} = \{(g, l) \in \mathrm{State} \mid \forall\, t{\in}n \colon (g, l_t) \in (\alpha_{\mathrm{mc}}(S))_t\} = \gamma_{\mathrm{mc}}(\alpha_{\mathrm{mc}}(S))$. □

We equip $D^{\#}$ with the componentwise partial order $\sqsubseteq$, defined as usual:

$$S \sqsubseteq T \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \forall\, t \in n \colon S_t \subseteq T_t \qquad (S, T \in D^{\#}) \,.$$

Proposition 16. *The pair of maps* $(\alpha_{\mathrm{mc}}, \gamma_{\mathrm{mc}})$ *is a Galois connection.*

This result was proven for finite n in Proposition and Definition 1.5.1 in [16].

Proof. Let us show that for all $S \in D, T \in D^{\#}$ we have

$$\alpha_{\mathrm{mc}}(S) \sqsubseteq T \text{ iff } S \subseteq \gamma_{\mathrm{mc}}(T) \,.$$

"$\Rightarrow$": Let $(g, l) \in S$. Let $T' = \alpha_{\mathrm{mc}}(S)$. Then, for each $i{<}n$, the definition of α_{mc} implies $(g, l_i) \in T'_i \subseteq T_i$. So $(g, l) \in \gamma_{\mathrm{mc}}(T)$ by definition of γ_{mc}.
"$\Leftarrow$": Let $T' = \alpha_{\mathrm{mc}}(S)$. Fix some $i{<}n$, and let $(g, l_i) \in T'_i$. By definition of α_{mc}

for each $j \in n\backslash\{i\}$ there is an l_j so that the tuple containing all these elements $(g,(l_k)_{k<n})$ is in S. But $S \subseteq \gamma_{\mathrm{mc}}(T)$, so $(g,(l_k)_{k<n}) \in \gamma_{\mathrm{mc}}(T)$. By definition of γ_{mc} we have $(g,l_i) \in T_i$. So $T'_i \subseteq T_i$. Since i was arbitrarily chosen, we have $T' \sqsubseteq T$ by definition of $\sqsubseteq$. $\square$

Corollary 17. *Multithreaded-Cartesian approximation is an upper closure operator. Formally: ρ_{mc} is an upper closure operator on $(D,\subseteq)$.*

Proof. By Prop. 15, $\rho_{\mathrm{mc}} = \gamma_{\mathrm{mc}} \circ \alpha_{\mathrm{mc}}$. By Prop. 16, $(D,\subseteq) \underset{\alpha_{\mathrm{mc}}}{\overset{\gamma_{\mathrm{mc}}}{\leftrightarrows}} (D^{\#},\sqsubseteq)$ is a Galois connection. By Prop. 9, $\gamma_{\mathrm{mc}} \circ \alpha_{\mathrm{mc}}$ is an upper closure operator. $\square$

Example 18 (Computation of multithreaded-Cartesian semantics). Consider the program from page 4. We are going to show that

$$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S)))$$
$$=$$
$$\begin{pmatrix} \{0\} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \\ \cup\, \{1\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{2\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{3\} \times (\{\mathrm{D}x \mid x \in \{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \end{pmatrix} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+)\,.$$

(Notice that we sloppily identify Glob×Loc×Loc with Glob×Loc^2.) We prove the equality by considering the left and right inclusions separately. We write LHS (left-hand side) for the "lfp ..." term before the equality sign and RHS (right-hand side) for the combination of the set terms after the equality sign.

"LHS $\overset{!}{\subseteq}$ RHS": By Theorem 4, it suffices to show that RHS is a prefix point of $\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))$, i.e., that $\rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(\mathrm{RHS})) \overset{!}{\subseteq} \mathrm{RHS}$. We compute:

$$\rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(\mathrm{RHS}))$$
$$=$$
$$\rho_{\mathrm{mc}} \begin{pmatrix} \{0\} \times \{(\mathrm{A},\mathrm{A})\} \\ \cup \begin{pmatrix} \{1\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{2\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{3\} \times \{\mathrm{D}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{0,3\} \times \{\mathrm{B,C}\}^+ \end{pmatrix} \times (\{\mathrm{A}x,\mathrm{D}x | x{\in}\{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \\ \cup\, \{0\} \times (\{\mathrm{A}x,\mathrm{D}x \mid x{\in}\{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \times \{\mathrm{B,C}\}^+ \\ \cup\, \{0\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B,C}\}^*\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{1\} \times \{\mathrm{AB}x \mid x \in \{\mathrm{B,C}\}^*\} \times \{\mathrm{B,C}\}^+ \\ \cup\, \{0\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B,C}\}^*\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{2\} \times \{\mathrm{AC}x \mid x \in \{\mathrm{B,C}\}^*\} \times \{\mathrm{B,C}\}^+ \\ \cup\, \{0\} \times (\{\mathrm{D}x \mid x \in \{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \times \{\mathrm{D}x \mid x \in \{\mathrm{B,C}\}^*\} \\ \cup\, \{3\} \times (\{\mathrm{D}x \mid x \in \{\mathrm{B,C}\}^*\} \cup \{\mathrm{B,C}\}^+) \times \{\mathrm{B,C}\}^+ \end{pmatrix}$$
$$= \ \mathrm{RHS}\,.$$

"LHS $\overset{!}{\supseteq}$ RHS": Notice that $\lambda\, S{\in}D.\, \mathrm{init} \cup \mathrm{post}(S)$ is pointwise less than or equal to $\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))$. By Prop. 5, $\mathrm{lfp}(\lambda\, S{\in}D.\, \mathrm{init} \cup \mathrm{post}(S)) \subseteq \mathrm{LHS}$. By

Example 14,

$$\{0\} \times \begin{pmatrix} \{(\mathrm{A}y,\mathrm{A}y),(\mathrm{D}y,\mathrm{D}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{(\mathrm{D}y,z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge\ z \text{ is a suffix of } y\} \\ \cup\, \{(y,\mathrm{D}z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge\ y \text{ is a suffix of } z\} \\ \cup\, \{(y,z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge (y \text{ is a suffix of } z\ \vee\ z \text{ is a suffix of } y)\} \end{pmatrix}$$
$$\cup\, \{1\} \times \{(\mathrm{AB}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\}$$
$$\cup\, \{2\} \times \{(\mathrm{AC}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\}$$
$$\cup\, \{3\} \times \begin{pmatrix} \{(\mathrm{D}y,\mathrm{A}y) \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{(y,\mathrm{A}z) \mid y,z \in \{\mathrm{B},\mathrm{C}\}^+ \wedge y \text{ is a suffix of } z\} \end{pmatrix}$$
$$\subseteq\ \mathrm{LHS}.$$

Since upper closures are isotone, applying ρ_{mc} to both sides of the above equation results in

$$\begin{aligned} &\{0\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+)^2 \\ \cup\, &\{1\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{A}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, &\{2\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{A}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, &\{3\} \times (\{\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times \{\mathrm{A}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \end{aligned}$$
$$\subseteq\ \rho_{\mathrm{mc}}(\mathrm{LHS})\ =\ \mathrm{LHS}.$$

Since post is isotone, we obtain

$$\mathrm{init} \cup \mathrm{post} \begin{pmatrix} \{0\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+)^2 \\ \cup\, \{1\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{A}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{2\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{A}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, \{3\} \times (\{\mathrm{D}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times \{\mathrm{A}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \end{pmatrix}$$
$$\subseteq\ \mathrm{init} \cup \mathrm{post}(\mathrm{LHS})\ \subseteq\ \mathrm{LHS}\,,$$

whence

$$\begin{aligned} &\{0\} \times \{(\mathrm{A},\mathrm{A})\} \\ \cup\, &\{1\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{2\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{3\} \times \{\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{0\} \times \{\mathrm{B},\mathrm{C}\}^+ \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{0\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times \{\mathrm{B},\mathrm{C}\}^+ \\ \cup\, &\{0\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, &\{0\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, &\{3\} \times \{\mathrm{B},\mathrm{C}\}^+ \times \{\mathrm{A}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \\ \cup\, &\{0\} \times (\{\mathrm{D}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times \{\mathrm{D}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \end{aligned}$$
$$\subseteq\ \mathrm{LHS}.$$

Applying isotone ρ_{mc} to both sides, we obtain

$$\begin{aligned} &\{0\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times (\{\mathrm{A}y,\mathrm{D}y \mid y{\in}\{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{1\} \times \{\mathrm{AB}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{2\} \times \{\mathrm{AC}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \\ \cup\, &\{3\} \times (\{\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \times (\{\mathrm{A}y,\mathrm{D}y \mid y \in \{\mathrm{B},\mathrm{C}\}^*\} \cup \{\mathrm{B},\mathrm{C}\}^+) \end{aligned}$$
$$\subseteq\ \rho_{\mathrm{mc}}(\mathrm{LHS})\ =\ \mathrm{LHS}\,,$$

i.e. RHS $\subseteq$ LHS.

$\square$

F Thread-Modular Reasoning and Inference System FQ

In this section we show that the inference system FQ for general multithreaded programs implements multithreaded-Cartesian abstract interpretation.

We are going to analyze a simpler system $\mathcal{FQ}'$ of inference rules, which doesn't collect changes of the shared state:

$$(\mathcal{FQ}'\ \mathcal{INIT})\ \frac{(g,l) \in \text{init}}{(g,l_t) \in \mathcal{R}_t}\ t{\in}n \qquad (\mathcal{FQ}'\ \mathcal{STEP})\ \frac{(g,w) \in \mathcal{R}_t \qquad (g,w) \rightsquigarrow_t (g',w')}{(g',w') \in \mathcal{R}_t}\ t{\in}n$$

$$(\mathcal{FQ}'\ \mathcal{ENV})\ \frac{(g,w) \in \mathcal{R}_t \qquad (g,\bar{w}) \in \mathcal{R}_s \qquad (g,\bar{w}) \rightsquigarrow_s (g',\bar{w}')}{(g',w) \in \mathcal{R}_t}\ s{\neq}t \text{ are in } n$$

We write $\mathcal{R} = (\mathcal{R}_t)_{t\in n}$ for the set family defined by $\mathcal{FQ}'$, and we write $\big(\widetilde{R},\widetilde{G}\big) = \big((\widetilde{R}_t)_{t\in n},(\widetilde{G}_t)_{t\in n}\big)$ for the set family defined by FQ.

We denote by $\sqcup$ the supremum on the complete lattice $D^{\#}$ (i.e., the componentwise union).

Proposition 19. *The least fixpoints of FQ and $\mathcal{FQ}'$ define the same sets of thread states. Formally: $\mathcal{R} = \widetilde{R}$.*

This result was proven for finite n in Prop. 1.7.1 in [16].

Proof idea. Syntactic transformation between the proof trees: ($\mathcal{FQ}'\ \mathcal{ENV}$) $\approx$ (FQ ENV) +(FQ STEP).

Proof. Let $\mathbb{D}_{\text{FQ}}$ and $\mathbb{D}_{\mathcal{FQ}'}$ be the derivation operators of FQ and $\mathcal{FQ}'$, respectively. For each $j \in \omega$ let

$$\big((\widetilde{R}^j_t)_{t<n},(\widetilde{G}^j_t)_{t<n}\big) \;:=\; \mathbb{D}^j_{\text{FQ}}\big((\emptyset)_{t<n},(\emptyset)_{t<n}\big) \quad \text{and} \quad (\mathcal{R}^j_t)_{t<n} \;:=\; \mathbb{D}^j_{\mathcal{FQ}'}\big((\emptyset)_{t<n}\big)$$

(where j is an upper index on the left-hand side and an exponent on the right-hand side of the assignments). By Theorem 13, $(\widetilde{R},\widetilde{G}) = \sup\big\{\big((\widetilde{R}^j_t)_{t<n},(\widetilde{G}^j_t)_{t<n}\big) \,\big|\, j\in\omega\big\}$ and $\mathcal{R} = \bigsqcup_{j\in\omega}(\mathcal{R}^j_t)_{t<n}$. It suffices to show that for all $j\in\omega$ we have $\forall\, t{\in}n\colon \widetilde{R}^j_t \subseteq \mathcal{R}_t \wedge \mathcal{R}^j_t \subseteq \widetilde{R}_t$. We prove this claim by induction on j.

So let $j\in\omega$ be arbitrary. Let $t\in n$.

"$\widetilde{R}^j_t \overset{!}{\subseteq} \mathcal{R}_t$": Let $(g,w) \in \widetilde{R}^j_t$. There is a rule of FQ that generated it.

- FQ INIT. Then there is some l such that $(g,l) \in \text{init}$ and $l_t = w$. Apply $\mathcal{FQ}'\ \mathcal{INIT}$.
- FQ STEP. Then $j>1$ and there is some $(\bar{g},\bar{w}) \in \widetilde{R}^{j-1}_t$ such that $(\bar{g},\bar{w}) \rightsquigarrow_t (g,w)$. By induction hypothesis, $(\bar{g},\bar{w}) \in \mathcal{R}_t$. By $\mathcal{FQ}'\ \mathcal{STEP}$, $(g,w) \in \mathcal{R}_t$.
- FQ ENV. Then $j>1$ and there is some $\hat{t} \in n\setminus\{t\}$ and $\bar{g}\in\text{Glob}$ such that $(\bar{g},g) \in \widetilde{G}^{j-1}_{\hat{t}}$ and $(\bar{g},w) \in \widetilde{R}^{j-1}_t$. Since FQ STEP is the only rule that can introduce $(\bar{g},g) \in \widetilde{G}^{j-1}_{\hat{t}}$, we must have $j>2$, and there are some $\bar{w},\bar{w}'$ such that $(\bar{g},\bar{w}) \in \widetilde{R}^{j-2}_{\hat{t}}$ and $(\bar{g},\bar{w}) \rightsquigarrow_{\hat{t}} (g,\bar{w}')$. By induction assumption, $(\bar{g},\bar{w}) \in \mathcal{R}_{\hat{t}}$ and $(\bar{g},w) \in \mathcal{R}_t$. By $\mathcal{FQ}'\ \mathcal{STEP}$, $(g,w) \in \mathcal{R}_t$.

"$\mathcal{R}^j_t \overset{!}{\subseteq} \widetilde{R}_t$": Let $(g,w) \in \mathcal{R}^j_t$. There is a rule of $\mathcal{FQ}'$ that generated it.

- $\mathcal{FQ}'\ \mathcal{INIT}$. Then there is some l such that $(g,l) \in \text{init}$ and $l_t = w$. Apply FQ INIT.
- $\mathcal{FQ}'\ \mathcal{STEP}$. Then $j>1$ and there is some $(\bar{g},\bar{w}) \in \mathcal{R}^{j-1}_t$ such that $(\bar{g},\bar{w}) \rightsquigarrow_t (g,w)$. By induction hypothesis, $(\bar{g},\bar{w}) \in \widetilde{R}_t$. By FQ STEP, $(g,w) \in \widetilde{R}_t$.

$\mathcal{FQ}'$ *ENV*. Then $j > 1$ and there are some $s \in n \setminus \{t\}$, $\bar{g} \in \text{Glob}$, $\bar{w}, \bar{w}' \in \text{Loc}$ such that $(\bar{g}, w) \in \mathcal{R}_t^{j-1}$, $(\bar{g}, \bar{w}) \in \mathcal{R}_s^{j-1}$, and $(\bar{g}, \bar{w}) \rightsquigarrow_s (g, \bar{w}')$. By induction hypothesis, $(\bar{g}, w) \in \widetilde{R}_t$ and $(\bar{g}, \bar{w}) \in \widetilde{R}_s$. By FQ STEP, $(\bar{g}, g) \in \widetilde{G}_s$. By FQ ENV, $(g, w) \in \widetilde{R}_t$.

□

Proposition 20. *The inference system $\mathcal{FQ}'$ is equivalent to multithreaded Cartesian abstract interpretation. Formally, both sides of the following equation are well defined, and equality holds:*

$$\text{lfp}(\mathcal{FQ}') \;=\; \text{lfp}(\lambda\, y.\, \alpha_{\text{mc}}(\text{init} \cup \text{post} \circ \gamma_{\text{mc}}(y)))\,.$$

For finite n this proposition was proven as a part of Theorem 1.7.2 in [16].

Proof idea. We have to show that two maps have the same least fixpoints. It suffices to show that the least fixpoint of one map is a prefix point of the other map and vice versa.

Proof. The left-hand side of the equation is well defined by definition. To show that the right-hand side is well defined, notice that by Prop. 16, $(\alpha_{\text{mc}}, \gamma_{\text{mc}})$ is a Galois connection, and by Prop. 6, the abstraction and concretization maps are isotone. Since post is also isotone, the map $\lambda y.\, \alpha_{\text{mc}}(\text{init} \cup \text{post} \circ \gamma_{\text{mc}}(y))$ is isotone, hence its fixpoint exists by Tarski's fixpoint theorem 4.

Using Prop. 7, the function on the right-hand side can be written as $F = \lambda y.\, \alpha_{\text{mc}}(\text{init}) \sqcup \alpha_{\text{mc}} \circ \text{post} \circ \gamma_{\text{mc}}(y)$. Let $X := \text{lfp}\, F$. Recall that $\mathcal{R} = \text{lfp}(\mathcal{FQ}')$. We prove $\mathcal{R} = X$ by considering the left and right componentwise inclusions separately.

"$\mathcal{R} \overset{!}{\sqsubseteq} X$": Since $\mathcal{R}$ is the least prefix point of the derivation operator of $\mathcal{FQ}'$, it suffices to show that X is a prefix point of the derivation operator of $\mathcal{FQ}'$, i.e., that $\mathbb{D}_{\mathcal{FQ}'}(X) \sqsubseteq X$. So let $t \in n$ and $(g, l) \in (\mathbb{D}_{\mathcal{FQ}'}(X))_t$. Then there is a rule $(A, t, (g, l))$ of $\mathcal{FQ}'$ such that $A \sqsubseteq X$. Let us look at the rule type.

$\mathcal{FQ}'$ *INIT*: Then there is some $\boldsymbol{l}$ such that $\boldsymbol{l}_t = l$ and $(g, \boldsymbol{l}) \in \text{init}$. Then $(g, l) = (g, \boldsymbol{l}_t) \in (\alpha_{\text{mc}}(\text{init}))_t$. Since X is a fixpoint, it is especially a prefix point, i.e. $F(X) \sqsubseteq X$. Thus $\alpha_{\text{mc}}(\text{init}) \sqsubseteq X$ and so $(\alpha_{\text{mc}}(\text{init}))_t \subseteq X_t$, whence $(g, l) \in X_t$.

$\mathcal{FQ}'$ *STEP*: Then there is some $(\bar{g}, \bar{l}) \in X_t$ such that $(\bar{g}, \bar{l}) \rightsquigarrow_t (g, l)$. Notice that $X = F(X)$ and thus $X_k = \{(\tilde{g}, \boldsymbol{l}_k) \mid (\tilde{g}, \boldsymbol{l}) \in \text{init} \cup \text{post} \circ \gamma_{\text{mc}}(X)\}$ for all $k \in n$. Thus, there is some $\boldsymbol{l}$ such that $(\bar{g}, \boldsymbol{l}) \in \text{init} \cup \text{post} \circ \gamma_{\text{mc}}(X)$ and $\boldsymbol{l}_t = \bar{l}$. Then for all $k \in n$ we have $(\bar{g}, \boldsymbol{l}_k) \in X_k$. Then $(\bar{g}, \boldsymbol{l}) \in \gamma_{\text{mc}}(X)$. Then $(g, \boldsymbol{l}[t \mapsto l]) \in \text{post} \circ \gamma_{\text{mc}}(X) \subseteq \text{init} \cup \text{post} \circ \gamma_{\text{mc}}(X)$. Then $(g, l) \in X_t$.

$\mathcal{FQ}'$ *ENV*: Then there are some $s \in n \setminus \{t\}$, $\bar{g} \in \text{Glob}$, $\tilde{l}, \tilde{l}' \in \text{Loc}$ such that $(\bar{g}, l) \in X_t$, $(\bar{g}, \tilde{l}) \in X_s$ and $(\bar{g}, \tilde{l}) \rightsquigarrow_s (g, \tilde{l}')$. Notice that $X = F(X)$ and thus $X_k = \{(\hat{g}, \boldsymbol{l}_k) \mid (\hat{g}, \boldsymbol{l}) \in \text{init} \cup \text{post} \circ \gamma_{\text{mc}}(X)\}$ for all $k \in n$. So there are tuples $\bar{\boldsymbol{l}}, \tilde{\boldsymbol{l}} \in \text{Loc}^n$ such that for all $k \in n$ we have $(\bar{g}, \bar{\boldsymbol{l}}_k), (\bar{g}, \tilde{\boldsymbol{l}}_k) \in X_k$, and $\bar{\boldsymbol{l}}_t = l$, and $\tilde{\boldsymbol{l}}_s = \tilde{l}$. Then $(\bar{g}, \bar{\boldsymbol{l}}[s \mapsto \tilde{l}]) \in \gamma_{\text{mc}}(X)$. Then $(g, \bar{\boldsymbol{l}}[s \mapsto \tilde{l}']) \in \text{post} \circ \gamma_{\text{mc}}(X) \subseteq \text{init} \cup \text{post} \circ \gamma_{\text{mc}}(X)$. Then $(g, l) \in X_t$.

"$\mathcal{R} \overset{!}{\sqsupseteq} X$": Since X is the least prefix point of F, it suffices to show that $\mathcal{R}$ is a prefix point of F, i.e., that $F(\mathcal{R}) \sqsubseteq \mathcal{R}$. So let $t \in n$ and $(g, l) \in (F(\mathcal{R}))_t$. There are two cases.

Case $(g,l) \in (\alpha_{\mathrm{mc}}(\mathrm{init}))_t$. By *FQ' INIT*, $(g,l) \in (\mathbb{D}_{FQ'}((\emptyset)_{s<n}))_t$. By Theorem 13, $(g,l) \in \mathcal{R}_t$.

Case $(g,l) \in (\alpha_{\mathrm{mc}} \circ \mathrm{post} \circ \gamma_{\mathrm{mc}}(\mathcal{R}))_t$. Then there is some $\boldsymbol{l}$ such that $(g,\boldsymbol{l}) \in \mathrm{post}\circ \gamma_{\mathrm{mc}}(\mathcal{R})$ and $\boldsymbol{l}_t = l$. Thus there is some thread $s \in n$ and a thread state $(\bar{g},\bar{l})$ such that $(\bar{g},\bar{l}) \rightsquigarrow_s (g,\boldsymbol{l}_s)$ and $(\bar{g},\boldsymbol{l}[s\mapsto\bar{l}]) \in \gamma_{\mathrm{mc}}(\mathcal{R})$. Thus for all $k \in n \setminus \{s\}$ we have $(\bar{g},\boldsymbol{l}_k) \in \mathcal{R}_k$ and $(\bar{g},\bar{l}) \in \mathcal{R}_s$. There are two cases.

Case $s = t$. The following instance of the *FQ' STEP* rule ensures that $(g,l) \in \mathcal{R}_t$:

$$\textit{FQ' STEP}\ \frac{(\bar{g},\bar{l}) \in \mathcal{R}_t \qquad (\bar{g},\bar{l}) \rightsquigarrow_t (g,l)}{(g,l) \in \mathcal{R}_t}$$

Case $s \neq t$. The following instance of the *FQ' ENV* rule ensures that $(g,l) \in \mathcal{R}_t$:

$$\textit{FQ' ENV}\ \frac{(\bar{g},l) \in \mathcal{R}_t \qquad (\bar{g},\bar{l}) \in \mathcal{R}_s \qquad (\bar{g},\bar{l}) \rightsquigarrow_s (g,\boldsymbol{l}_s)}{(g,l) \in \mathcal{R}_t}$$

□

Corollary 21. *The inference system FQ is equivalent to multithreaded Cartesian abstract interpretation. Formally, the right-hand side of the following equation is well defined, and equality holds:*

$$(\mathrm{lfp}(\mathrm{FQ}))_1 \;=\; \mathrm{lfp}(\lambda y.\, \alpha_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post} \circ \gamma_{\mathrm{mc}}(y)))\,.$$

Proof. Follows from Props. 19 and 20. □

G Multithreaded-Cartesian Abstract Interpretation for Multithreaded Recursive Programs

This section is dedicated to computations and proofs of statements in the main body of the paper regarding the TMR algorithm. All the claims are new.

Lemma 22. $(\widetilde{R},\widetilde{G}) \preceq (R,G)$.

Proof. Let $\mathbb{D}_{\mathrm{FQ}}\colon (\mathfrak{P}(\mathrm{Glob}{\times}\mathrm{Loc}))^n \times (\mathfrak{P}(\mathrm{Glob}^2))^n \to (\mathfrak{P}(\mathrm{Glob}{\times}\mathrm{Loc}))^n \times (\mathfrak{P}(\mathrm{Glob}^2))^n$ be the derivation operator of FQ. ($\mathbb{D}_{\mathrm{FQ}}$ applies each rule exactly once to the argument; its formal definition is in Sect. C.) We now show that $\mathbb{D}_{\mathrm{FQ}}(R,G) \preceq (R,G)$.

Let $(\check{R},\check{G}) = \mathbb{D}_{\mathrm{FQ}}(R,G)$. Let $t \in n$. We prove now componentwise inclusion:

"$\check{R}_t \subseteq R_t$": Let $(g,w) \in \check{R}_t$. There is a rule that generated it.

FQ INIT: Then there is some l such that $(g,l) \in \mathrm{init}$ and $l_t = w$. By TMR INIT, $g \xrightarrow[t]{w} \mathfrak{f}$. So $(g,w) \in R_t$.

FQ STEP: Then there is some $(\bar{g},\bar{w}) \in R_t$ such that $(\bar{g},\bar{w}) \rightsquigarrow_t (g,w)$. According to the definition of $\rightsquigarrow_t$, there are three cases:

Case there are $a,b,c \in \mathrm{Frame}$, $u \in \mathrm{Frame}^*$ such that $\bar{w} = au \wedge w = bcu \wedge ((\bar{g},a),(g,b,c)) \in \uplus_t$. From $\bar{g} \xrightarrow[t]{\bar{w}}{}^* \mathfrak{f}$ we obtain some v such that $\bar{g} \xrightarrow[t]{a} v \xrightarrow[t]{u}{}^* \mathfrak{f}$. By TMR PUSH, $g \xrightarrow[t]{b} (g,b) \xrightarrow[t]{c} v$, so $g \xrightarrow[t]{bcu}{}^* \mathfrak{f}$ and $(g,w) \in R_t$.

Case there are $a,b \in \mathrm{Frame}$, $u \in \mathrm{Frame}^*$ such that $\bar{w} = au \wedge w = bu \wedge ((\bar{g},a),(g,b)) \in \boxminus_t$. From $\bar{g} \xrightarrow[t]{\bar{w}}{}^* \mathfrak{f}$ we obtain some v such that $\bar{g} \xrightarrow[t]{a} v \xrightarrow[t]{u}{}^* \mathfrak{f}$. By TMR STEP, $g \xrightarrow[t]{b} v$. So $g \xrightarrow[t]{bu}{}^* \mathfrak{f}$ and $(g,w) \in R_t$.

Case there are $a, b, c \in \text{Frame}$, $u \in \text{Frame}^*$ such that $((\bar{g}, a, b), (g, c)) \in \sqcup^{\uparrow}_t \wedge \bar{w} = abu \wedge w = cu$. From $(\bar{g}, abu) \in R_t$ we obtain some $v, \bar{v}$ such that $\bar{g} \xrightarrow[t]{a} v \xrightarrow[t]{b} \bar{v} \xrightarrow[t]{u}{}^* \mathfrak{f}$. By TMR POP, $g \xrightarrow[t]{c} \bar{v}$. So $g \xrightarrow[t]{cu}{}^* \mathfrak{f}$. So $(g, w) \in R_t$.

FQ ENV: Then there is $\hat{t} \neq t$ and $\bar{g}$ such that $(\bar{g}, g) \in G_{\hat{t}}$ and $(\bar{g}, w) \in R_t$. From $\bar{g} \xrightarrow[t]{w}{}^* \mathfrak{f}$ we get some $a \in \text{Frame}$, $u \in \text{Frame}^*$, and v such that $w = au$ and $\bar{g} \xrightarrow[t]{a} v \xrightarrow[t]{u}{}^* \mathfrak{f}$. By TMR ENV, $g \xrightarrow[t]{a} v$, so $g \xrightarrow[t]{au}{}^* \mathfrak{f}$. Thus $(g, w) \in R_t$.

"$\breve{G}_t \subseteq G_t$": Let $(g, g') \in \breve{G}_t$. By FQ STEP, there are some w, w' such that $(g, w) \in R_t$ and $(g, w) \rightsquigarrow_t (g', w')$. According to the definition of $\rightsquigarrow_t$, there are three cases.

Case there are $a, b, c \in \text{Frame}$, $u \in \text{Frame}^*$ such that $((g, a), (g', b, c)) \in \sqcup^{\downarrow}_t \wedge w = au \wedge w' = bcu$. From $g \xrightarrow[t]{au}{}^* \mathfrak{f}$ we obtain some v such that $g \xrightarrow[t]{a} v$. By TMR PUSH, $(g, g') \in G_t$.

Case there are $a, b \in \text{Frame}$, $u \in \text{Frame}^*$ such that $((g, a), (g', b)) \in \boxminus_t \wedge w = au \wedge w' = bu$. From $g \xrightarrow[t]{au}{}^* \mathfrak{f}$ we obtain some v such that $g \xrightarrow[t]{a} v$. By TMR STEP, $(g, g') \in G_t$.

Case there are $a, b, c \in \text{Frame}$, $u \in \text{Frame}^*$ such that $((g, a, b), (g', c)) \in \sqcup^{\uparrow}_t \wedge w = abu \wedge w' = cu$. From $g \xrightarrow[t]{abu}{}^* \mathfrak{f}$ we obtain some $v, \bar{v}$ such that $g \xrightarrow[t]{a} v \xrightarrow[t]{b} \bar{v}$. By TMR POP, $(g, g') \in G_t$.

We have shown that (R, G) is a prefix point of $\mathbb{D}_{\text{FQ}}$. By Theorem 4, $\text{lfp}(\mathbb{D}_{\text{FQ}}) \preceq (R, G)$. That is, $(\widetilde{R}, \widetilde{G}) \preceq (R, G)$. ☐

Proposition 23 (Soundness).
$\text{lfp}(\lambda\, S \in D.\, \rho_{\text{mc}}(\text{init} \cup \text{post}(S))) \;\subseteq\; \bigcup_{g \in \text{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}$.

Proof. $\text{lfp}(\lambda\, S \in D.\, \rho_{\text{mc}}(\text{init} \cup \text{post}(S))) \subseteq$ [applying Prop. 10 to $\lambda\, S.\, \text{init} \cup \text{post}(S)$] $\gamma_{\text{mc}}(\text{lfp}(\lambda\, y.\, \alpha_{\text{mc}}(\text{init} \cup \text{post} \circ \gamma_{\text{mc}}(y)))) =$ [by Cor. 21] $\gamma_{\text{mc}}(\widetilde{R}) \subseteq$ [by Lemma 22 and using the fact that concretization maps are isotone by Prop. 6] $\gamma_{\text{mc}}(R) = \{(g, l) \in \text{State} \mid \forall\, t \in n\colon (g, l_t) \in R_t\} = \{(g, l) \in \text{State} \mid \forall\, t \in n\colon g \xrightarrow[t]{l_t}{}^* \mathfrak{f}\} = \bigcup_{g \in \text{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}$. ☐

Let

$$\mathbb{D}_{\text{TMR}} : (\mathfrak{P}(V \times \text{Frame} \times V))^n \times (\mathfrak{P}(\text{Glob}^2))^n \to (\mathfrak{P}(V \times \text{Frame} \times V))^n \times (\mathfrak{P}(\text{Glob}^2))^n$$

be the derivation operator of TMR. We inductively define

$$\begin{aligned} (\rightarrow_0, (G_{t,0})_{t \in n}) &:= ((\emptyset)_{t \in n}, (\emptyset)_{t \in n}) \text{ and} \\ (\rightarrow_{i+1}, (G_{t,i+1})_{t \in n}) &:= \mathbb{D}_{\text{TMR}}(\rightarrow_i, (G_{t,i})_{t \in n}) \text{ for } i \in \mathbb{N}_0. \end{aligned}$$

Since $\mathbb{D}_{\text{TMR}}$ is isotone, the above sequence of iterates is ascending. Since each rule of TMR has finitely many premises, Theorem 13 implies

$$(\rightarrow, G) \quad = \quad \text{lfp}\, \mathbb{D}_{\text{TMR}} \quad = \quad \left(\lambda\, t \in n.\, \bigcup_{i \in \mathbb{N}_0} \xrightarrow[t]{}_i,\;\; \lambda\, t \in n.\, \bigcup_{i \in \mathbb{N}_0} G_{t,i} \right).$$

We will use underscore for innermost existentially quantified unnamed variables, writing $v \xrightarrow[t]{\_}_i \bar{v}$ for $\exists\, a \in \text{Frame}\colon v \xrightarrow[t]{a}_i \bar{v}$ and $v \xrightarrow[t]{\_}{}^*_i \bar{v}$ for $\exists\, w \in \text{Frame}^*\colon v \xrightarrow[t]{w}{}^*_i \bar{v}$ ($i \in \mathbb{N}_0$, $t \in n$).

Lemma 24. *For each edge in an iterate i, the source of the edge is not $\mathfrak{f}$, the target of the edge has a walk to $\mathfrak{f}$ in the same iterate i, and the target is not a shared state. Formally:*
Let $t{\in}n$, $v, \bar{v} \in V$, $i{\in}\mathbb{N}_0$ such that $v \underset{t}{\Rightarrow}_i \bar{v}$. Then $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$, $v \in \mathrm{Glob} \,\dot{\cup}\, \mathrm{Glob}{\times}\mathrm{Frame}$, and $\bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}$.

Proof. By induction on i. Assume some $i{\in}\mathbb{N}_0$ is given and $\forall\, \hat{\imath}{<}i\colon (\forall\, v, \bar{v} \in V\colon v \underset{t}{\Rightarrow}_{\hat{\imath}} \bar{v} \Rightarrow (\bar{v} \underset{t}{\Rightarrow}^*_{\hat{\imath}} \mathfrak{f} \wedge v \in \mathrm{Glob} \,\dot{\cup}\, \mathrm{Glob}{\times}\mathrm{Frame} \wedge \bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}))$. Let $v, \bar{v} \in V$, $a{\in}\mathrm{Frame}$ be given such that $v \overset{a}{\underset{t}{\to}}_i \bar{v}$. There is a rule that generated it.
TMR INIT: Then $v \in \mathrm{Glob}$ and $\bar{v} = \mathfrak{f}$.
TMR STEP: Then $i{>}1$, $v{\in}\mathrm{Glob}$, and there are some $\bar{g}{\in}\mathrm{Glob}$, $\bar{a} \in \mathrm{Frame}$ such that $\bar{g} \overset{\bar{a}}{\underset{t}{\to}}_{i-1} \bar{v}$. By induction hypothesis, $\bar{v} \underset{t}{\Rightarrow}^*_{i-1} \mathfrak{f}$ and $\bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}$. Notice that $\underset{t}{\to}_{i-1} \subseteq \underset{t}{\to}_i$, so $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$.
TMR PUSH: Then $i{>}1$. There are two cases.
Case $v \in \mathrm{Glob}$: Then there are some $g{\in}\mathrm{Glob}$, $\bar{a}, c \in \mathrm{Frame}$ and $\hat{v}{\in}V$ such that $g \overset{\bar{a}}{\underset{t}{\to}}_{i-1} \hat{v}$ and $\bar{v} = (v, a) \overset{c}{\underset{t}{\to}}_i \hat{v}$. By induction hypothesis, $\hat{v} \underset{t}{\Rightarrow}^*_{i-1} \mathfrak{f}$. So $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$.
Case $v \in \mathrm{Glob}{\times}\mathrm{Frame}$: Then there are some $g{\in}\mathrm{Glob}$ and $\bar{a} \in \mathrm{Frame}$ such that $g \overset{\bar{a}}{\underset{t}{\to}}^*_{i-1} \bar{v}$. By induction hypothesis, $\bar{v} \underset{t}{\Rightarrow}^*_{i-1} \mathfrak{f}$ and $\bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}$. Then $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$.

TMR POP: Then $i{>}1$, $v{\in}\mathrm{Glob}$, and there are $\hat{v}{\in}V$, $b{\in}\mathrm{Frame}$ such that $\hat{v} \overset{b}{\underset{t}{\to}}_{i-1} \bar{v}$. By induction hypothesis, $\bar{v} \underset{t}{\Rightarrow}^*_{i-1} \mathfrak{f}$ and $\bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}$. Then $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$.
TMR ENV: Then $i{>}1$, $v{\in}\mathrm{Glob}$, and there is $g{\in}\mathrm{Glob}$ such that $g \overset{a}{\underset{t}{\to}}_{i-1} \bar{v}$. By induction hypothesis, $\bar{v} \underset{t}{\Rightarrow}^*_{i-1} \mathfrak{f}$ and $\bar{v} \in \mathrm{Glob}{\times}\mathrm{Frame} \,\dot{\cup}\, \{\mathfrak{f}\}$. Then $\bar{v} \underset{t}{\Rightarrow}^*_i \mathfrak{f}$.
□

Let $S^i \subseteq (\mathrm{Glob}{\times}\mathrm{Loc})^2$ be defined recursively as follows for $i \in \mathbb{N}_+$:

$$\begin{array}{rcll} S^1 & = & \mathrm{id}_{\mathrm{Glob}\times\mathrm{Loc}} \cup \overset{\widetilde{G}}{\rightsquigarrow}_t, & \\ S^i & = & S^1 \,\mathbin{;}\, S^{i-1} & \text{for } i \geq 2. \end{array}$$

By Prop. 3, $\bigcup_{i\in\mathbb{N}_+} S^i \;=\; \overset{\widetilde{G}}{\rightsquigarrow}^*_t$.

By induction on i in S^i we obtain Lemmas 25 and 26:

Lemma 25. *For each thread, the sets of thread states described by FQ are closed under the bigstep operational semantics with FQ-context. Formally:*
Let $t{\in}n$, $(g, w) \in \widetilde{R}_t$, and $(g, w) \overset{\widetilde{G}}{\rightsquigarrow}^*_t (g', w')$. Then $(g', w') \in \widetilde{R}_t$.

Proof. So fix $t{\in}n$. We will show by induction on i that

$$\forall\, i{\in}\mathbb{N}_+\ \forall\, (g,w) \in \widetilde{R}_t,\ (g',w') \in \mathrm{Glob}{\times}\mathrm{Loc}\colon ((g,w),(g',w')) \in S^i \Rightarrow (g',w') \in \widetilde{R}_t. \quad (5)$$

Let $i \in \mathbb{N}_+$ be arbitrary and $\forall\, j{\in}\mathbb{N}_+\colon (j{<}i \Rightarrow (\forall\, (g, w) \in \widetilde{R}_t,\ (g', w') \in \mathrm{Glob}{\times}\mathrm{Loc}\colon (((g, w), (g', w')) \in S^j \Rightarrow (g', w') \in \widetilde{R}_t)))$. Let $(g, w) \in \widetilde{R}_t$ and $(g', w') \in \mathrm{Glob}{\times}\mathrm{Loc}$ be such that $((g, w), (g', w')) \in S^i$.
Case $i = 1$.
Case $(g, w) = (g', w')$. Then $(g', w') \in \widetilde{R}_t$ by assumption.

Case $(g,w) \rightsquigarrow_t (g',w')$. Then $(g',w') \in \widetilde{R}_t$ by FQ STEP.

Case $w = w'$ and there is $s \in n\backslash\{t\}$ such that $(g,g') \in \widetilde{G}_s$. Then $(g',w) \in \widetilde{R}_t$ by FQ ENV, so $(g',w') \in \widetilde{R}_t$.

Case $i{>}1$. Then there are $\bar{g},\bar{w}$ such that $((g,w),(\bar{g},\bar{w})) \in S^1$ and $((\bar{g},\bar{w}),(g',w')) \in S^{i-1}$. By the previous case, $(\bar{g},\bar{w}) \in \widetilde{R}_t$. By the induction hypothesis applied to $i-1$, we get $(g',w') \in \widetilde{R}_t$.

We have proven (5). Thus, $\forall\,(g,w) \in \widetilde{R}_t,\ (g',w') \in \text{Glob}{\times}\text{Loc}\colon ((g,w),(g',w')) \in \bigcup_{i\in\mathbb{N}_+} S^i \Rightarrow (g',w') \in \widetilde{R}_t$. Since $\bigcup_{i\in\mathbb{N}_+} S^i = \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t$, the lemma is proven. $\square$

Lemma 26. *The bigstep operational semantics with FQ-context of each thread is closed under appending stacks to the bottom. Formally:*
Let $t{\in}n$, $(g,w) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g',w')$, and $v{\in}\text{Frame}^*$. Then $(g,wv) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g',w'v)$.

Proof. Fix an arbitrary $t{\in}n$. We will show

$$\forall\, i{\in}\mathbb{N}_+\ \forall\,((g,w),(g',w')) \in S^i,\ v \in \text{Frame}^*\colon ((g,wv),(g',w'v)) \in S^i \tag{6}$$

by induction on i. So let $i \in \mathbb{N}_+$ be arbitrary, and assume that

$$\forall\, j{\in}\mathbb{N}_+\colon j{<}i \Rightarrow \left(\forall\,((g,w),(g',w')) \in S^j,\ v{\in}\text{Frame}^*\colon ((g,wv),(g',w'v)) \in S^j\right).$$

Let $((g,w),(g',w')) \in S^i$ and $v{\in}\text{Frame}^*$.

Case $i{=}1$.

Case $(g,w) = (g',w')$. Then $(g,wv) = (g',w'v)$, so $((g,wv),(g',w'v)) \in S^1$ by definition.

Case $(g,w) \rightsquigarrow_t (g',w')$. In each of the three cases defining $\rightsquigarrow_t$ (push, internal, or pop thread transition) we obtain $(g,wv) \rightsquigarrow_t (g',w'v)$. So $((g,wv),(g',w'v)) \in S^1$.

Case $w = w'$ and there is $s \in n\backslash\{t\}$ such that $(g,g') \in \widetilde{G}_s$. Then $wv = w'v$, so $((g,wv),(g',w'v)) \in S^1$.

Case $i{>}1$. Then there are $\bar{g},\bar{w}$ such that $((g,w),(\bar{g},\bar{w})) \in S^1$ and $((\bar{g},\bar{w}),(g',w')) \in S^{i-1}$. Applying the induction hypothesis twice, we obtain $((g,wv),(\bar{g},\bar{w}v)) \in S^1$ and $((\bar{g},\bar{w}v),(g',w'v)) \in S^{i-1}$. Then $((g,wv),(g',w'v)) \in S^i$.

The claim (6) is proven. Thus, $\forall\, g,g' \in \text{Glob},\ w,w' \in \text{Loc},\ v{\in}\text{Frame}^*\colon (g,w) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g',w') \Rightarrow (g,wv) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g',w'v)$. $\square$

In the following formal definition we will use interval notation $[0,|w|]$ (resp. $[0,|w|[$) to denote $\{x{\in}\mathbb{N}_0 \mid 0 \le x \le |w|\}$ (resp. $\{x{\in}\mathbb{N}_0 \mid 0 \le x < |w|\}$); further, we index letters of words starting from 0.

Definition 27. *For each $t \in n$, $v,\bar{v} \in V$, $i \in \mathbb{N}_0$, and $w \in \text{Frame}^*$, let*

$$\text{tr}(t,i,v,\bar{v},w) \quad := \quad \min\left\{ \left| \left\{ k \in [0,|w|[\;\middle|\; (p(k),w_k,p(k{+}1)) \notin \bigcup_{\substack{j\in\mathbb{N}_0\\ j<i}} \underset{t}{\rightarrow}_j \right\} \right| \right.$$
$$\left.\;\middle|\; p \in ([0,|w|] \to V) \wedge p(0){=}v \wedge p(|w|){=}\bar{v} \wedge \forall\, k \in [0,|w|[\colon p(k) \xrightarrow[t]{w_k}{}_i\, p(k{+}1) \right\},$$

where $\min\emptyset = \infty$, *so that* $\text{tr}(t,i,v,\bar{v},w) \in \mathbb{N}_0 \,\dot\cup\, \{\infty\}$.

Restatement of Lemma 2. *For all $i \in \mathbb{N}_0$ and all $j \in \mathbb{N}_0$ we have:*

(i) $\forall\, g{\in}\mathrm{Glob}, t{\in}n, w{\in}\mathrm{Frame}^*\colon\ \mathrm{tr}(t,i,g,\mathfrak{f},w) = j \ \Rightarrow\ (g,w) \in \widetilde{R}_t$,

(ii) $\forall g,\bar{g}{\in}\mathrm{Glob}, t{\in}n, b{\in}\mathrm{Frame}, w{\in}\mathrm{Frame}^*\colon \mathrm{tr}(t,i,g,(\bar{g},b),w){=}j \Rightarrow (\bar{g},b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g,w)$,

(iii) $\forall\, t \in n\colon\ G_{t,i} \subseteq \widetilde{G}_t$.

Proof. Consider the usual strict lexicographic ordering on $\mathbb{N}_0{\times}\mathbb{N}_0$ over the usual ordering on natural numbers: $(\hat{\imath},\hat{\jmath}) <_{\mathrm{lex}} (i,j) \overset{\mathrm{def}}{\iff} ((\hat{\imath}{=}i \wedge \hat{\jmath}{<}j) \vee \hat{\imath}{<}i)$ for $(\hat{\imath},\hat{\jmath}),(i,j) \in \mathbb{N}_0{\times}\mathbb{N}_0$.

We proceed by induction on $<_{\mathrm{lex}}$. So let $i,j \in \mathbb{N}_0$ be arbitrary, and we assume that (i)–(iii) hold for all pairs lexicographically smaller that (i,j).

(a) Let $g{\in}\mathrm{Glob}$, $t{\in}n$, $b{\in}\mathrm{Frame}$, and $w{\in}\mathrm{Frame}^*$ be given such that $\mathrm{tr}(t,i,g,\mathfrak{f},w) = j$. Then there is a walk $p \in ([0,|w|] \to V)$ such that $p(0) = g$, $p(|w|) = \mathfrak{f}$, and $\forall\, k \in [0,|w|[\colon p(k) \xrightarrow[t]{w_k}_i p(k{+}1)$. Since $g{\neq}\mathfrak{f}$, w is nonempty. Since $\xrightarrow[t]{}_0$ is the empty relation, $i{\geq}1$. If $j{=}0$, then $\forall\, k \in [0,|w|[\colon p(k) \xrightarrow[t]{w_k}_{i-1} p(k{+}1)$, then $\mathrm{tr}(t,i{-}1,g,\mathfrak{f},w) \in \mathbb{N}_0$, and, by induction hypothesis, $(g,w) \in \widetilde{R}_t$. Thus let $j{>}0$ from now on. Let $m = \min\Big\{k \in [0,|w|[\ \Big|\ (p(k), w_k, p(k{+}1)) \notin \xrightarrow[t]{}_{i-1}\Big\}$. There are $u,x \in \mathrm{Frame}^*$ such that $w = uw_mx$, $g \xrightarrow[t]{u}{}^*_{i-1} p(m) \xrightarrow[t]{w_m}_i p(m{+}1) \xrightarrow[t]{x}{}^*_i \mathfrak{f}$, and in some walk proving $p(m{+}1) \xrightarrow[t]{x}{}^*_i \mathfrak{f}$ we have less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. There is a rule that generated $p(m) \xrightarrow[t]{w_m}_i p(m{+}1)$:

TMR INIT. Then $p(m) \in \mathrm{Glob}$ and $p(m{+}1) = \mathfrak{f}$. By Lemma 24, $p(m)$ is not a target of any edge and $p(m{+}1)$ is not a source of any edge. Thus, $u{=}x{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule, there is some l such that $(g,l) \in \mathrm{init}$ and $l_t = w_0$. By FQ INIT, $(g,l_t) \in \widetilde{R}_t$.

TMR STEP. Then $p(m) \in \mathrm{Glob}$. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule there are some $\hat{g}$ and a such that $((\hat{g},a),(g,w_0)) \in \mapsto_t$ and $\hat{g} \xrightarrow[t]{a}_{i-1} p(1)$. There is a walk $\hat{g} \xrightarrow[t]{a}_{i-1} p(1) \xrightarrow[t]{x}{}^*_i \mathfrak{f}$ that contains less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$, so $\mathrm{tr}(t,i,\hat{g},\mathfrak{f},ax) < j$. By induction hypothesis, part (i), $(\hat{g},ax) \in \widetilde{R}_t$. Notice that $(\hat{g},ax) \rightsquigarrow_t (g,w_0x) = (g,w)$. By FQ STEP, $(g,w) \in \widetilde{R}_t$.

TMR PUSH and $p(m) \in \mathrm{Glob}$. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule, $p(1) = (g,w_0) \neq \mathfrak{f}$. So there are $c{\in}\mathrm{Frame}$ and $y{\in}\mathrm{Frame}^*$ such that $p(1) \xrightarrow[t]{c}_i p(2) \xrightarrow[t]{y}{}^*_i \mathfrak{f}$, and the number of edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$ in a walk proving this is less than j, and $x = cy$. The edge $(g,w_0) \xrightarrow[t]{c}_i p(2)$ can be formed only by a (potentially different) TMR PUSH rule, by which then $\hat{g} \xrightarrow[t]{a}_{i-1} p(2)$ and $((\hat{g},a),(g,w_0,c)) \in \uplus_t$ for some $\hat{g}$ and a. In at least one walk that shows $\hat{g} \xrightarrow[t]{a}_{i-1} p(2) \xrightarrow[t]{y}{}^*_i \mathfrak{f}$ we have less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. Thus $\mathrm{tr}(t,i,\hat{g},\mathfrak{f},ay) < j$. By induction hypothesis, part (i), $(\hat{g},ay) \in \widetilde{R}_t$. Notice that $(\hat{g},ay) \rightsquigarrow_t (g,w_0cy) = (g,w)$. By FQ STEP, $(g,w) \in \widetilde{R}_t$.

TMR PUSH and $p(m) \in \mathrm{Glob}{\times}\mathrm{Frame}$. So $p(m) = (\bar{g},b)$ for some $\bar{g}$ and b. From $g \xrightarrow[t]{u}{}^*_{i-1} (\bar{g},b)$ and induction hypothesis, part (ii), we obtain $(\bar{g},b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g,u)$. By Lemma 26, $(\bar{g},bw_mx) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g,uw_mx)$. By the rule, there are some $\hat{g}$ and a

such that $\hat g \xrightarrow[t]{a}_{i-1} p(m{+}1)$ and $((\hat g, a), (\bar g, b, w_m)) \in \uplus_t$. There is a walk proving $\hat g \xrightarrow[t]{a}_{i-1} p(m{+}1) \xrightarrow[t]{x}{}^*_i \mathfrak{f}$ which uses less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. By induction hypothesis, part (i), $(\hat g, ax) \in \widetilde{R}_t$. Notice that $(\hat g, ax) \rightsquigarrow_t (\bar g, bw_m x)$. By FQ STEP, $(\bar g, bw_m x) \in \widetilde{R}_t$. By Lemma 25, $\widetilde{R}_t \ni (g, uw_m x) = (g, w)$.

TMR POP. Them $p(m) \in$ Glob. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule, there are $\hat g$, a, b, and v such that $\hat g \xrightarrow[t]{a}_{i-1} v \xrightarrow[t]{b}_{i-1} p(1)$ and $((\hat g, a, b), (g, w_0)) \in \uplus_t$. There is a walk proving $\hat g \xrightarrow[t]{abx}{}^*_i \mathfrak{f}$ with less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. By the induction hypothesis, part (i), $(\hat g, abx) \in \widetilde{R}_t$. Notice that $(\hat g, abx) \rightsquigarrow_t (g, w_0 x)$. By FQ STEP, $\widetilde{R}_t \ni (g, w_0 x) = (g, w)$.

TMR ENV. Then $p(m) \in$ Glob. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule there are $\hat t \in n \setminus \{t\}$ and $\hat g$ such that $(\hat g, g) \in G_{\hat t, i-1}$ and $\hat g \xrightarrow[t]{w_0}_{i-1} p(1)$. Then there is a walk $\hat g \xrightarrow[t]{w_0}_{i-1} p(1) \xrightarrow[t]{x}{}^*_i \mathfrak{f}$ with less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$, so $\operatorname{tr}(t, i, \hat g, \mathfrak{f}, w) < j$. By induction hypothesis, part (i), $(\hat g, w) \in \widetilde{R}_t$. By induction hypothesis, part (iii), $(\hat g, g) \in \widetilde{G}_{\hat t}$. By FQ ENV, $(g, w) \in \widetilde{R}_t$.

(b) Let $g, \bar g \in$ Glob, $t{\in}n$, $b{\in}$Frame, and $w{\in}$Frame* such that $\operatorname{tr}(t, i, g, (\bar g, b), w) = j$. Then there is a walk $p \in ([0, |w|] \to V)$ such that $p(0) = g$, $p(|w|) = (\bar g, b)$ and $\forall\, k \in [0, |w|[\colon p(k) \xrightarrow[t]{w_k}_i p(k{+}1)$. Since $g \neq (\bar g, b)$, w is nonempty. Since $\xrightarrow[t]{}_0$ is the empty relation, $i{\geq}1$. If $j{=}0$, then $\forall\, k \in [0, |w|[\colon p(k) \xrightarrow[t]{w_k}_{i-1} p(k{+}1)$, then $\operatorname{tr}(t, i{-}1, g, (\bar g, b), w) \in \mathbb{N}_0$, and, by induction hypothesis, $(\bar g, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$. Thus let $j{>}0$ from now on. Let $m = \min\Big\{k \in [0, |w|[\;\Big|\; (p(k), w_k, p(k{+}1)) \notin \xrightarrow[t]{}_{i-1}\Big\}$. There are $u, x \in$ Frame* such that $w = uw_m x$, $g \xrightarrow[t]{u}{}^*_{i-1} p(m) \xrightarrow[t]{w_m}_i p(m{+}1) \xrightarrow[t]{x}{}^*_i (\bar g, b)$, and in the walk $(p(m{+}i{+}1))_{0 \leq i \leq |w|-m-1}$ proving $p(m{+}1) \xrightarrow[t]{x}{}^*_i (\bar g, b)$ we have less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. There is a rule that generated $p(m) \xrightarrow[t]{w_m}_i p(m{+}1)$.

TMR INIT. Then $p(m) \in$ Glob and $p(m{+}1) = \mathfrak{f}$. By Lemma 24, $p(m{+}1)$ is not a source of any edge. But $(\bar g, b) \neq \mathfrak{f}$ and $\mathfrak{f} = p(m{+}1) \xrightarrow[t]{x}{}^*_{i-1} (\bar b, g)$. Contradiction!

TMR STEP. Then $p(m) \in$ Glob. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $p(m) = g$. By the rule there are $\hat g$, a such that $((\hat g, a), (g, w_0)) \in \uplus_t$ and $\hat g \xrightarrow[t]{a}_{i-1} p(1)$. The walk $\hat g, p(1), \ldots, p(|w|)$ proves that $\hat g \xrightarrow[t]{a}_{i-1} p(1) \xrightarrow[t]{x}{}^*_i (\bar g, b)$ and has less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. So $\operatorname{tr}(t, i, \hat g, (\bar g, b), ax) < j$. By induction hypothesis, part (ii), $(\bar g, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\hat g, ax)$. Notice that $(\hat g, ax) \rightsquigarrow_t (g, w_0 x) = (g, w)$. So $(\bar g, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.

TMR PUSH and $p(m) \in$ Glob. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $p(m) = g$. By the rule, $p(1) = (g, w_0)$.

Case $x{=}\varepsilon$. Then $|w| = 1$ and $(\bar g, b) = (g, w_0)$. By reflexivity, $(\bar g, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.
Case $x = cy$ for some $c{\in}$Frame and $y{\in}$Frame*. Then there is a walk $p(1) \xrightarrow[t]{c}_i p(2) \xrightarrow[t]{y}{}^*_i (\bar g, b)$ with less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. The edge $(g, w_0) \xrightarrow[t]{c}_i$

$p(2)$ can be constructed only by a TMR PUSH rule, by which then $\hat{g} \xrightarrow[t]{a}_{i-1} p(2)$ and $((\hat{g}, a), (g, w_0, c)) \in \sqcup^{\downarrow}_t$ for some $\hat{g}$ and a. Then there is a walk proving $\hat{g} \xrightarrow[t]{a}_{i-1} p(2) \xrightarrow[t]{y}{}^*_i (\bar{g}, b)$ with less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$, so $\mathrm{tr}(t, i, \hat{g}, (\bar{g}, b), ay) < j$. By induction hypothesis, part (ii), $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\hat{g}, ay)$. Notice that $(\hat{g}, ay) \rightsquigarrow_t (g, w_0cy) = (g, w)$. By definition of $\overset{\widetilde{G}}{\rightsquigarrow}{}^*_t$, $(\hat{g}, ay) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$. By transitivity, $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.

TMR PUSH and $p(m) \in \mathrm{Glob}{\times}\mathrm{Frame}$. So $p(m) = (\check{g}, \check{b})$ for some $\check{g}{\in}\mathrm{Glob}$ and $\check{b}{\in}\mathrm{Frame}$. By the rule, there are some $\hat{g}$ and a such that $((\hat{g}, a), (\check{g}, \check{b}, w_m)) \in \sqcup^{\downarrow}_t$ and $\hat{g} \xrightarrow[t]{a}_{i-1} p(m{+}1)$. By induction hypothesis, part (ii), applied to $g \xrightarrow[t]{u}{}^*_{i-1} (\check{g}, \check{b})$, we obtain $(\check{g}, \check{b}) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, u)$. By Lemma 26, $(\check{g}, \check{b}w_mx) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, uw_mx) = (g, w)$. Notice that $(\hat{g}, ax) \rightsquigarrow_t (\check{g}, \check{b}w_mx)$, so $(\hat{g}, ax) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\check{g}, \check{b}w_mx)$. By transitivity, $(\hat{g}, ax) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$. In some walk proving $\hat{g} \xrightarrow[t]{a}_{i-1} p(m{+}1) \xrightarrow[t]{x}{}^*_i (\bar{g}, b)$ we have less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. By the induction hypothesis, part (ii), $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\hat{g}, ax)$. By transitivity, $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.

TMR POP. Then $p(m) \in \mathrm{Glob}$. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule, there are $\hat{g}$, a, b, and v such that $\hat{g} \xrightarrow[t]{a}_{i-1} v \xrightarrow[t]{b}_{i-1} p(1)$ and $((\hat{g}, a, b), (g, w_0)) \in \sqcup^{\uparrow}_t$. There is a walk proving $\hat{g} \xrightarrow[t]{abx}{}^*_i (\bar{g}, b)$ with less than j edges from $\xrightarrow[t]{}_i \setminus \xrightarrow[t]{}_{i-1}$. By induction hypothesis, part (ii), $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\hat{g}, abx)$. Notice that $(\hat{g}, abx) \rightsquigarrow_t (g, w_0x) = (g, w)$. Thus, $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.

TMR ENV. Then $p(m) \in \mathrm{Glob}$. By Lemma 24, $p(m)$ is not a target of any edge. Thus $u{=}\varepsilon$, $m{=}0$, and $g = p(m)$. By the rule, there are $\hat{g}{\in}\mathrm{Glob}$ and $s \in n{\setminus}\{t\}$ such that $\hat{g} \xrightarrow[t]{w_0}_{i-1} p(1)$ and $(\hat{g}, g) \in G_{s,i-1}$. Notice that $\mathrm{tr}(t, i, \hat{g}, (\bar{g}, b), w) < j$, so, by induction assumption, part (ii), we have $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (\hat{g}, w)$. By by induction assumption, part (iii), we obtain $(\hat{g}, g) \in \widetilde{G}_s$. By definition of $\overset{\widetilde{G}}{\rightsquigarrow}{}^*_t$, $(\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}^*_t (g, w)$.

(c) Let $t{\in}n$ and $(g, g') \in G_{t,i}$. There is a rule that generated it.

TMR STEP. There are a, b, v such that $((g, a), (g', b)) \in \sqcup^{-}_t$ and $g \xrightarrow[t]{a}_{i-1} v$. By Lemma 24, there is some w such that $v \xrightarrow[t]{w}{}^*_{i-1} \mathfrak{f}$. Then $g \xrightarrow[t]{aw}{}^*_{i-1} \mathfrak{f}$. By induction hypothesis, part (i), $(g, aw) \in \widetilde{R}_t$. Notice that $(g, aw) \rightsquigarrow_t (g', bw)$. By FQ STEP, $(g, g') \in \widetilde{G}_t$.

TMR PUSH. There are a, b, c, v such that $((g, a), (g', b, c)) \in \sqcup^{\downarrow}_t$ and $g \xrightarrow[t]{a}_{i-1} v$. By Lemma 24, there is some w such that $v \xrightarrow[t]{w}{}^*_{i-1} \mathfrak{f}$. Then $g \xrightarrow[t]{aw}{}^*_{i-1} \mathfrak{f}$. By induction hypothesis, part (i), $(g, aw) \in \widetilde{R}_t$. Notice that $(g, aw) \rightsquigarrow_t (g', bcw)$. By FQ STEP, $(g, g') \in \widetilde{G}_t$.

TMR POP. There are $a, b, c, v, \bar{v}$ such that $((g, a, b), (g', c)) \in \sqcup^{\uparrow}_t$ and $g \xrightarrow[t]{a}_{i-1} v \xrightarrow[t]{b}_{i-1} \bar{v}$. By Lemma 24, there is some w such that $\bar{v} \xrightarrow[t]{w}{}^*_{i-1} \mathfrak{f}$. Then $g \xrightarrow[t]{abw}{}^*_{i-1}$

$\mathfrak{f}$. By induction hypothesis, part (i), $(g, abw) \in \widetilde{R}_t$. Notice that $(g, abw) \rightsquigarrow_t (g', cw)$. By FQ STEP, $(g, g') \in \widetilde{G}_t$.

$\square$

Lemma 28. $(\widetilde{R}, \widetilde{G}) \succeq (R, G)$.

Proof. Let $t \in n$.

- Let $(g, w) \in R_t$. Then $g \xrightarrow[t]{w}{}^* \mathfrak{f}$. Fix a walk proving this. Each edge in this walk belongs to $\xrightarrow[t]{}_i$ for some $i \in \mathbb{N}_0$. Since $(\xrightarrow[t]{}_i)_{i \in \mathbb{N}_0}$ is ascending, there is some $i \in \mathbb{N}_0$ such that $g \xrightarrow[t]{w}{}^*_i \mathfrak{f}$. Notice that $\mathrm{tr}(t, i, g, \mathfrak{f}, w) \in \mathbb{N}_0$. By Lemma 2(i), $(g, w) \in \widetilde{R}_t$.
- Let $(g, g') \in G_t$. There is some $i \in \mathbb{N}_0$ such that $(g, g') \in G_{t,i}$. By Lemma 2(iii), $(g, g') \in \widetilde{G}_t$.

$\square$

Proposition 29 (Completeness).
$\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) \;\supseteq\; \bigcup_{g \in \mathrm{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}$.

Proof. $\mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) = \mathrm{lfp}(\lambda\, S \in D.\, \rho_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post}(S))) =$ [by Prop. 10] $\gamma_{\mathrm{mc}}(\mathrm{lfp}(\lambda\, T \in D^{\#}.\, \alpha_{\mathrm{mc}}(\mathrm{init} \cup \mathrm{post} \circ \gamma_{\mathrm{mc}}(T)))) =$ [by Cor. 21] $\gamma_{\mathrm{mc}}(\widetilde{R}) \supseteq$ [by Lemma 28 and Prop. 6] $\gamma_{\mathrm{mc}}(R) = \{(g, l) \in \mathrm{State} \mid \forall\, t \in n \colon (g, l_t) \in R_t\} = \{(g, l) \in \mathrm{State} \mid \forall\, t \in n \colon g \xrightarrow[t]{l_t}{}^* \mathfrak{f}\} = \bigcup_{g \in \mathrm{Glob}} \{g\} \times \{l \in \mathrm{Loc}^n \mid \forall\, t \in n \colon g \xrightarrow[t]{l_t}{}^* \mathfrak{f}\} = \bigcup_{g \in \mathrm{Glob}} \{g\} \times \prod_{t \in n} L_{g,t}$.
$\square$

H Implementation of TMR and Its Running Time

Now we are going to show the algorithm for executing the inference system TMR. Then we will determine an asymptotic upper bound on the worst-case running time of the algorithm.

We implement TMR as a traditional worklist-based algorithm. Our construction will aim for a possibly low worst-case running time on a well-defined machine model. Especially, we employ data structures which are geared towards the worst case. (If one would like to implement the TMR algorithm in real software, we suggest using BDDs, as in Crocopat, or, in the infinite case, constraint solvers as QARMC.) In particular, our main goal is optimizing the worst-case running time towards an asymptotically possibly small function of the number of threads n. Our second goal is optimization towards low worst-case running time in the number of shared states and frames. (At this point, an eager reader would be directed to algorithms on CFL reachability and "database-join" operations. However, the results on CFL reachability and database-join are not *directly* usable here, since, to the best of our knowledge, they all assume certain simplifications, say, unit-cost measure and constant-cost perfect hashing, which, as we will shortly explain, are not justified in our computation model.) To simplify the presentation of our implementation, we omit optimizations that reduce the running time just by a constant factor.

From now on till the remainder of § H we assume that n, Glob, Frame, and init are finite. For our convenience, we assume that $n \geq 1$, that Glob is the set of first Γ nonnegative integers for some $\Gamma \in \mathbb{N}_+$, that Frame is the set of first F nonnegative integers for some $F \in \mathbb{N}_+$, and that $I = |\mathrm{init}| \geq 1$. (If any of n, Γ, F, or I is zero, the task becomes much easier—an exercise for the reader.) Each element of n (viewed as a set), Glob, and Frame will be represented in binary, as well as n (viewed as a number), Γ, and F.

We assume that the execution model for our algorithms is a RAM with logarithmic cost measure [18].

(An aside is worth to be made. First, there is a 50-year-old heated discussion on how a model should look like: it is about the instruction set, unbounded registers, indirection, program-in-memory, etc. To cut it short, we are using a well-known and easily comprehensible computational model which we believe offers all the features required to encode our algorithms, and we believe that other reasonable models and variations, broadly speaking, /e.g., RASP/ would give similar results. Second, the discussion about the cost of single operations is similarly heated, controversial, and old. To cut it short again, we see that many multithreaded programs, say, parts of the operating system, possess huge shared memory, so it is reasonable to assume that accessing a shared state takes non-constant time. Further, since we account for the size of objects of at least one type /shared/, we generally account for the size of objects of all types /shared, frame, thread identifier, etc./, and we are using an established and well-explained measure to do so. For similar reasons, we disallow perfect hashing, since the standard universal hash function family involves precomputing large prime numbers.)

Following Mehlhorn [18], we define

$$L(x) = \begin{cases} 1\,, & \text{if } x = 0\,, \\ \lfloor \log_2 x \rfloor + 1\,, & \text{otherwise} \end{cases} \qquad (x \in \mathbb{N}_0)$$

as the function giving the length of the binary representation of its argument.

H.1 Implementing containers

We will need to maintain sets of tuples of fixed length, bounded-size sets of integers, and maps from tuples of integers to sets of bounded-size.

Throughout the remainder of § H, we will use NULL $= 0$ for the fixed address of a cell not belonging to any data structure.

Sets of tuples of fixed length. The algorithm will need to maintain sets of tuples of fixed length (e.g., a set of quintuples). Such sets will be implemented as multidimensional arrays containing lists as follows.

Let us assume that we wish to store a subset S of an m-dimensional product over natural numbers $\prod_{i=0}^{m-1}(\mathbb{N}_0 \cap [0, k_i[)$, for some $m \in \mathbb{N}_+$, which is constant and independent on the input, and some $k_0, \dots, k_{m-1} \in \mathbb{N}_+$, which are constants stored in cells at constant addresses and are part of the input. We will show how to store S, roughly speaking, as a doubly-linked list in an array. The construction will occupy exactly $3\prod_{i=0}^{m-1} k_i + 2$ cells as follows. Given available space from some address B onward, we store the pointers to list head and list tail in cells B and $B+1$. For storing the actual list, we use $3\prod_{i=0}^{m-1} k_i$ consecutive cells starting from address $B+2$. We maintain the invariant that for all $(x_i)_{i<m} \in \prod_{i<m}(\mathbb{N}_0 \cap [0, k_i[)$:

- If $(x_i)_{i<m} \in S$, then the cell at address $B+3\sum_{i<m}(\prod_{j<i} k_i)x_i+2$ will contain 1, the cell at address $B+3\sum_{i<m}(\prod_{j<i} k_i)x_i+3$ will contain the address of the previous member of the list (or zero if at the list head), and $B+3\sum_{i<m}(\prod_{j<i} k_i)x_i+4$ will contain the address of the next member of the list (or zero if at the list tail).

– If $(x_i)_{i<m} \notin S$, then the three cells at addresses $B + 3\sum_{i<m}(\prod_{j<i} k_i)x_i + 2$ till $B + 3\sum_{i<m}(\prod_{j<i} k_i)x_i + 4$ contain previously stored, arbitrary information.

Since in general there are many ways of linearly ordering the elements of a set, the constructed representation is in general not unique. The following operations on this data structure can be readily implemented (an easy exercise for the reader), while we are going to provide just the running times for later references:

`construct()`/`cleanup()` (initialization / cleaning up): $\mathcal{O}((\prod_{i<m} k_i)L(B+\prod_{i<m} k_i))$.

`add(`$(x_i)_{i<m}$`)` (adding an element $(x_i)_{i<m}$ to S if not yet there and doing nothing otherwise): $\mathcal{O}(L(B) + \sum_{i<m} L(k_i))$.

`remove_any()` (removing an arbitrary element from S and returning the address of the first element of the tuple or NULL, if none found): $\mathcal{O}(L(B + \prod_{i<m} k_i))$.

`contains(`$(x_i)_{i<m}$`)` (testing membership in S): $\mathcal{O}(L(B) + \sum_{i<m} L(k_i)))$.

`next` / `prev` (given a pointer to a list element, getting the address of the next/previous element of the list, or determining that there is no next/previous one): $L(B + \prod_{i<m} k_i))$.

`pointer` $\to$ `data` (given a pointer to a list element, determining the element itself): $\mathcal{O}(L(B + \prod_{i<m} k_i))$.

Bounded-size sets. We will also need containers for sets of bounded size which does not depend on the input (say, at most 2) over elements of bounded size which does depend on the input (say, less than k). A set $S \subseteq \mathbb{N}_0 \cap [0, k[$ of at most p (e.g., $p = 2$) elements is implemented as $p + 1$ cells in a straightforward way. Assume that the addresses we are using to store the set are B till $B + p$ for some B. We store $|S|$ in cell B. Cells $B + 1$ till $B + |S|$ contain the elements of S in an unspecified order, and the contents of the remaining cells $B + |S| + 1$ till $B + p$ is arbitrary. We do not touch the contents of the cells $B + |S| + 1$ till $B + p$ when removing elements from the set or when initializing the set. The standard add, remove, and membership-test operations need $\mathcal{O}(L(B) + L(k))$ time.

Partial maps from tuples of integers to bounded-size sets. We will also need containers storing partial maps from tuples of integers to sets of bounded size $p \geq 1$ (which does not depend on the input).

More precisely, let us assume that we wish to store a partial map $S : \big(\prod_{i=0}^{m-1}(\mathbb{N}_0 \cap [0, k_i[)\big) \dashrightarrow \mathfrak{P}(T)$ such that $m, k_0, \ldots, k_{m-1} \in \mathbb{N}_+$ and for each $y \in \operatorname{img} S$ we have $1 \leq |y| \leq p$ and $y \subseteq \mathbb{N}_0 \cap [0, k_m[$ for some $k_m \in \mathbb{N}_+$. (Purely set-theoretically, such S are isomorphic to subsets $\hat{S} \subseteq \prod_{i=0}^{m}(\mathbb{N}_0 \cap [0, k_i[)$ of $(m+1)$-tuples such that for each $z \in \mathbb{N}_0 \cap [0, k_m[$ there are at most p different m-tuples $x \in \prod_{i=0}^{m-1}(\mathbb{N}_0 \cap [0, k_i[)$ such that $(x, z) \in \hat{S}$.)

We will combine the above data structures and store S again in a doubly-linked list in an array. The construction will occupy exactly $(p+3)\prod_{i=0}^{m-1} k_i + 2$ cells as follows. Given available space from some address B onward, we store the pointers to list head and list tail in cells B and $B + 1$. For storing the actual list, we use $(p + 3)\prod_{i=0}^{m-1} k_i$ consecutive cells starting from address $B + 2$. We maintain the invariant that for all $x \in \prod_{i=0}^{m-1}(\mathbb{N}_0 \cap [0, k_i[)$:

– If $x \in \operatorname{dom} S$, then the cell at address $B+(p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i+2$ will contain $|S(x)|$, which is positive, the cell at address $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + 3$ will contain the address of the previous member of the list (or zero if at the list head),

$B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + 4$ will contain the address of the next member of the list (or zero if at the list tail), cells $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + 5$ till $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + |S(x)| + 4$ will contain members of $S(x)$, and the contents of the cells $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + |S(x)| + 5$ till $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + p + 4$ is arbitrary.

– If $x \notin \operatorname{dom} S$, then the cell at address $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + 2$ will contain 0, and the $p+2$ cells at addresses $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + 3$ till $B + (p+3)\sum_{i<m}(\prod_{j<i} k_i)x_i + p + 4$ contain previously stored, arbitrary information.

Again, this representation is not unique. The following operations on this data structure can be readily implemented (an exercise for the reader), while we are going to provide just the running times for later references. We use the fact that p does not depend on the input and hide it in the asymptotic notation $\mathcal{O}(\dots)$.

`construct()`/`cleanup()` (initialization / cleaning up): $\mathcal{O}((\prod_{i<m} k_i) L(B + \prod_{i<m} k_i))$.

`add(`x, t`)` where $x = (x_i)_{i<m} \in \prod_{i<m}(\mathbb{N}_0 \cap [0, k_i[)$ and $t \in \mathbb{N}_0 \cap [0, k_m[$: If $x \in \operatorname{dom} S$ and $|S(x) \cup \{t\}| \le p$, update S to $S[x \mapsto S(x) \cup \{t\}]$. If $x \in \operatorname{dom} S$ and $|S(x) \cup \{t\}| > p$, do nothing. If $x \notin \operatorname{dom} S$, update S to $S[x \mapsto \{t\}]$. The running time is $\mathcal{O}(L(B) + \sum_{i \le m} L(k_i))$.

`is_nonempty()` (testing whether $S \neq \emptyset$): $\mathcal{O}(L(B))$.

`remove_any()` If S is nonempty, return an arbitrary pair (x, t) such that $x \in \operatorname{dom} S$ and $t \in S(x)$ and update S to

$$S \;:=\; \begin{cases} S[x \mapsto S(x) \setminus \{t\}]\,, & \text{if } S(x) \supsetneq \{t\}\,, \\ S \setminus \{(x, \{t\})\}\,, & \text{if } S(x) = \{t\}\,. \end{cases}$$

If S is empty, return an arbitrary result in $m+1$ cells. The running time is $\mathcal{O}(L(B) + \sum_{i \le m} L(k_i))$.

`contains(`x, t`)` where $x = (x_i)_{i<m} \in \prod_{i<m}(\mathbb{N}_0 \cap [0, k_i[)$ and $t \in \mathbb{N}_0 \cap [0, k_m[$ (testing whether $x \in \operatorname{dom} S \wedge t \in S(x)$): $\mathcal{O}(L(B) + \sum_{i \le m} L(k_i))$.

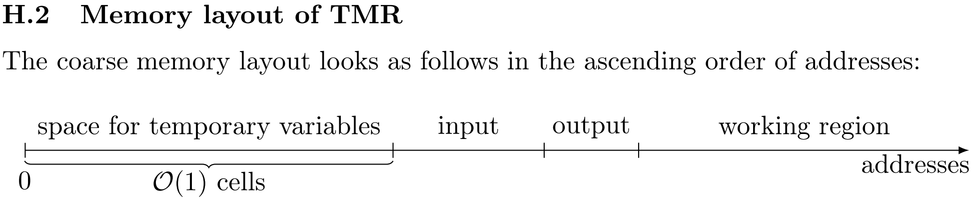
`next` / `prev` (given a pointer to a list element, getting the address of the next/previous element of the list, or determining that there is no next/previous one): $\mathcal{O}(L(B) + \sum_{i<m} L(k_i))$.

Function application: Given $x \in \prod_{i<m}(\mathbb{N}_0 \cap [0, k_i[)$, return $S(x)$ if $x \in \operatorname{dom} S$, and $\emptyset$ otherwise. $\mathcal{O}(L(B) + \sum_{i \le m} L(k_i))$.

Checking "$x \in \operatorname{dom} S \Rightarrow |S(x)| < y$" for arguments $x = (x_i)_{i<m} \in \prod_{i<m}(\mathbb{N}_0 \cap [0, k_i[)$ and $1 \le y \le p$ (checking whether $S(x)$ is undefined or is defined and has less than y elements): $\mathcal{O}(L(B + \prod_{i<m} k_i))$.

H.2 Memory layout of TMR

The coarse memory layout looks as follows in the ascending order of addresses:



Temporary variables are those which serve as additional registers (Mehlhorn's model allows 4 built-in registers, while or algorithms might need more).

Input layout. The input is laid out in the memory as follows:

$$n \mid \Gamma \mid F \mid \sqcup^{\downarrow}_0 \ \sqcup^{-}_0 \ \sqcup^{\uparrow}_0 \ \ldots \ \sqcup^{\downarrow}_{n-1} \ \sqcup^{-}_{n-1} \ \sqcup^{\uparrow}_{n-1} \mid I \mid \text{init} \longrightarrow$$

We assume that n, Γ, F, and I are stored at three different fixed addresses. For each $t < n$:

- The set $\sqcup^{\downarrow}_t$ is stored as an $\Gamma \times F$-array of containers
  $push_t(g, a) \ = \ \{(g', b, c) \mid ((g, a), (g', b, c)) \in \sqcup^{\downarrow}_t\}$ ($g \in \text{Glob}$, $a \in \text{Frame}$).
- The set $\sqcup^{-}_t$ is stored as an $\Gamma \times F$-array of containers
  $int_t(g, a) \ = \ \{(g', b) \mid ((g, a), (g', b)) \in \sqcup^{-}_t\}$ ($g \in \text{Glob}$, $a \in \text{Frame}$).
- The set $\sqcup^{\uparrow}_t$ is stored as an $\Gamma \times F \times F$-array of containers
  $pop_t(g, a, b) \ = \ \{(g', c) \mid ((g, a, b), (g', c)) \in \sqcup^{\uparrow}_t\}$ ($g \in \text{Glob}$, $a, b \in \text{Frame}$).

The set init of tuples of length $n + 1$ (recall that n is unknown in advance) is maintained just as a simple list as follows. Let us assume some enumeration of elements of init: $\text{init}[0] = (g^{[0]}, (l_i^{[0]})_{i<n})$, $\ldots$, $\text{init}[I-1] = (g^{[I-1]}, (l_i^{[I-1]})_{i<n})$. Then the set init is assumed to be stored in ascending order as follows:

$$g^{[0]}\ l_0^{[0]} \ \ldots \ l_{n-1}^{[0]} \mid g^{[1]}\ l_0^{[1]} \ \ldots \ l_{n-1}^{[1]} \mid \ \ldots \ \mid g^{[I-1]}\ l_0^{[I-1]} \ \ldots \ l_{n-1}^{[I-1]} \longrightarrow$$

Output layout. From a high-level view, the output consists of the sets $\xrightarrow[t]{}$ of triples from $V \times \text{Frame} \times V$ ($t{<}n$). (We do not consider the sets G_t as a part of the output; rather we view them as internal data ($t < n$).) These sets will be stored as containers right after the initial states:

$$\xrightarrow[0]{} \ | \ \ldots \ | \ \xrightarrow[n-1]{} \ .$$

For each $t < n$, the set $\xrightarrow[t]{}$ will be stored as a family of 4 containers as follows:

$$relSF_t \mid relSI_t \mid relII_t \mid relIF_t \longrightarrow$$

The four containers will store the following elements.

- The container $relSF_t$ for $\xrightarrow[t]{} \cap (\text{Glob} \times \text{Frame} \times \{\mathfrak{f}\})$ will store pairs (g, a) denoting $g \xrightarrow[t]{a} \mathfrak{f}$).
- The container $relSI_t$ for $\xrightarrow[t]{} \cap (\text{Glob} \times \text{Frame} \times (\text{Glob} \times \text{Frame}))$ will store quadruples (g, a, g', b) denoting $g \xrightarrow[t]{a} (g', b)$.
- The container $relII_t$ for $\xrightarrow[t]{} \cap ((\text{Glob} \times \text{Frame}) \times \text{Frame} \times (\text{Glob} \times \text{Frame}))$ will store quintuples (g, a, b, g', c) denoting $(g, a) \xrightarrow[t]{b} (g', c)$.
- The container $relIF_t$ for $\xrightarrow[t]{} \cap ((\text{Glob} \times \text{Frame}) \times \text{Frame} \times \{\mathfrak{f}\})$ will store triples (g, a, b) denoting $(g, a) \xrightarrow[t]{b} \mathfrak{f}$.

One can memorize the container names by reading S as start, I as internal, and F as final.

Internal data layout. The major auxiliary data is the representation of the sets G_t $(t < n)$. It is a partial map $guar : \mathrm{Glob}{\times}\mathrm{Glob} \dashrightarrow \mathfrak{P}(n)$ such that for each $(g, g') \in \mathrm{dom}\, guar$ we have $1 \le |guar(g, g')| \le 2$. Our algorithm will use the data structure as follows:

– if there are at most two $t \in n$ such that $(g, g') \in G_t$, then all such t will eventually appear in $guar(g, g')$,
– if there are at least two $t \in n$ such that $(g, g') \in G_t$, then some $s, t \in n$ such that $s \ne t \wedge (g, g') \in G_t \cap G_s$ will eventually appear in $guar(g, g')$, and
– if $t \in guar(g, g')$, then $(g, g') \in G_t$.

This data structure allows the following checks to be executed in a constant number of operations:

– Given $g, g' \in \mathrm{Glob}$ and $t \in n$, is there some $s \in n \setminus \{t\}$ such that $(g, g') \in G_s$?
– Given $g, g' \in \mathrm{Glob}$, is there at most one $t \in n$ such that $(g, g') \in G_t$?

By preprocessing the input sets $\overset{\uparrow}{\sqcup}_t$, we obtain a container $\widehat{pop}(t, g, a) = \{(b, g', c) \mid ((g, a, b), (g', c)) \in \overset{\uparrow}{\sqcup}_t\}$.

The following two auxiliary data structures will serve executing the TMR POP rule.

– For each $t{\in}n$, $g{\in}\mathrm{Glob}$, and $b, b' \in \mathrm{Frame}$ we maintain a container $\widehat{relSI}(t, g, b, b') \subseteq \mathrm{Glob}\times\mathrm{Frame}$ such that (g', c) will arrive in $\widehat{relSI}(t, g, b, b')$ iff there are some $\bar{g} \in \mathrm{Glob}$ and $\bar{a} \in \mathrm{Frame}$ such that $((\bar{g}, \bar{a}, b'), (g', c)) \in \overset{\uparrow}{\sqcup}_t$ and the algorithm has already found out that $\bar{g} \xrightarrow[t]{\bar{a}} (g, b)$.
– For each $t{\in}n$, $g{\in}\mathrm{Glob}$, and $b, b' \in \mathrm{Frame}$ we maintain a container $\widehat{relSIG}(t, g, b, b') \subseteq \mathrm{Glob}\times\mathrm{Glob}$ such that $(\bar{g}, g')$ will arrive in $\widehat{relSIG}(t, g, b, b')$ iff there are some $\bar{a}, c \in \mathrm{Frame}$ such that $((\bar{g}, \bar{a}, b'), (g', c)) \in \overset{\uparrow}{\sqcup}_t$ and the algorithm has already found out that $\bar{g} \xrightarrow[t]{\bar{a}} (g, b)$.

The worklist will contain items which are already generated by the rules but which are not yet committed to the *rel*... sets, i.e, the items temporarily stored for later application of the inference rules to these items. Since the worklist has to store tuples of different length, technically, we have to split the worklist into several auxiliary containers, stored in the working region after the already described data structures:

– *workG*, which is a partial map $\mathrm{Glob}{\times}\mathrm{Glob} \dashrightarrow \mathfrak{P}(n)$ such that for all $(g, g') \in \mathrm{dom}\, workG$ we have $1 \le |workG(g, g')| \le 2$. This map holds just generated elements that will be processed later and added to *guar*.
– *workSF*, which is a container for triples $(t, g, a) \in n{\times}\mathrm{Glob}{\times}\mathrm{Frame}$ such that (g, a) will be processed later and added to $relSF_t$.
– *workSI*, which is a container for quintuples $(t, g, a, g', b) \in n \times \mathrm{Glob} \times \mathrm{Frame} \times \mathrm{Glob} \times \mathrm{Frame}$ such that (g, a, g', b) will be processed later and added to $relSI_t$.
– *workII*, which is a container for sixtuples $(t, g, a, b, g', c) \in n \times \mathrm{Glob} \times \mathrm{Frame} \times \mathrm{Frame} \times \mathrm{Glob} \times \mathrm{Frame}$ such that (g, a, b, g', c) will be processed later and added to $relII_t$.
– *workIF*, which is a container for quadruples $(t, g, a, b) \in n{\times}\mathrm{Glob}{\times}\mathrm{Frame}{\times}\mathrm{Frame}$ such that (g, a, b) will be processed later and added to $relIF_t$.

H.3 Code

We describe our implementation bottom-up, starting with simple procedures and ending with the main program. All the procedures should be understood as pieces of code that will be inlined (we will not implement a call stack on a RAM; there will be no activation records).

We are going to use the following auxiliary procedures.

```
// Add (t,g,a) to workSF if not yet processed:
procedure Try2Add2SF(t,g,a):
    if ¬relSF_t.contains(g,a) then workSF.add(t,g,a);
```

```
// Add (t,g,a,g',b) to workSI if not yet processed:
procedure Try2Add2SI(t,g,a,g',b):
    if ¬relSI_t.contains(g,a,g',b) then workSI.add(t,g,a,g',b);
```

```
// Add (t,g,a,b,g',c) to workII if not yet processed:
procedure Try2Add2II(t,g,a,b,g',c):
    if ¬relII_t.contains(g,a,b,g',c) then workII.add(t,g,a,b,g',c);
```

```
// Add (t,g,a,b) to workIF if not yet processed:
procedure Try2Add2IF(t,g,a,b):
    if ¬relIF_t.contains(g,a,b) then workIF.add(t,g,a,b);
```

```
// Add (t,g,g') to workG if not yet processed:
procedure Try2Add2G(t,g,g'):
    if ¬guar.contains((g,g'),t) ∧ ((g,g') ∉ dom guar ∨ |guar(g,g')| < 2) ∧
      ((g,g') ∉ dom workG ∨ |workG(g,g')| < 2) then
        workG.add((g,g'),t)
```

To understand why at most two thread identifiers inside $guar(g, g')$ or $workG(g, g')$ suffice, notice that the sets $G_{...}$ are used only in the TMR ENV rule, for which it does not matter whether there are exactly two $t \in n$ that fulfill $(g, g') \in G_t$ or more.

How let us turn to initialization.

```
// Initialize the working region and read init:
procedure Initialize():
    foreach t ∈ n do
        relSF_t := relSI_t := relII_t := relIF_t := ∅;
        foreach g ∈ Glob, a ∈ Frame do
            pop^(t, g, a).construct();
            foreach b ∈ Frame do
                relSI^(t, g, a, b).construct();
                relSIG^(t, g, a, b).construct();
                foreach (g', c) ∈ pop_t(g, a, b) do
                    pop^(t, g, a).add(b, g', c)
    workG.construct();
    workSF.construct();
    workSI.construct();
    workII.construct();
    workIF.construct();
    foreach (g, l) ∈ init, t ∈ n do workSF.add(t, g, l_t);
```

Now we introduce procedures `Process...` that process elements from different parts of the worklist. Actual removal of these elements from worklists is done elsewhere.

```
// Process a shared state change from workG:
procedure ProcessG(g ∈ Glob, g' ∈ Glob, t ∈ n):
    T := guar(g, g'); // T occupies three cells
    if |T| = 0 then
        guar.add((g, g'), t);
        foreach s ∈ n \ {t} do
            foreach a ∈ Frame such that (g, a) ∈ relSF_s do
                Try2Add2SF(s, g', a);
            foreach a, b ∈ Frame, ḡ∈Glob with (g, a, ḡ, b) ∈ relSI_s do
                Try2Add2SI(s, g', a, ḡ, b)
    else if |T| = 1 then
        guar.add((g, g'), t);
        s := the unique element of T;
        foreach a ∈ Frame with (g, a) ∈ relSF_s do Try2Add2SF(s, g', a);
        foreach a, b ∈ Frame, ḡ∈Glob with (g, a, ḡ, b) ∈ relSI_s do
            Try2Add2SI(s, g', a, ḡ, b)
    // Else |T| = 2 and we do nothing.
```

// Process a new edge from Glob to $\mathfrak{f}$:

procedure ProcessSF($itemAddr$):

- // Precondition: there is job to do, i.e., $itemAddr \neq \mathrm{NULL}$.
- $(t, g, a) := itemAddr \rightarrow data$;
- $relSF_t$.add(g, a);
- **foreach** $(g', a') \in int_t(g, a)$ **do**
  - Try2Add2SF(t, g', a');
  - Try2Add2G(t, g, g')
- **foreach** $(g', b, c) \in push_t(g, a)$ **do**
  - Try2Add2SI(t, g', b, g', b);
  - Try2Add2IF(t, g', b, c);
  - Try2Add2G(t, g, g')
- **foreach** $g' \in \mathrm{Glob}$ *such that* $guar(g, g') \setminus \{t\} \neq \emptyset$ **do**
  - Try2Add2SF(t, g', a)

// Process a new edge from Glob to $\mathrm{Glob} \times \mathrm{Frame}$:

procedure ProcessSI($itemAddr$):

- // Precondition: $itemAddr \neq \mathrm{NULL}$.
- $(t, g, a, \bar{g}, b) := itemAddr \rightarrow data$;
- $relSI_t$.add($g, a, \bar{g}, b$);
- **foreach** $(g', b) \in int_t(g, a)$ **do**
  - Try2Add2SI($t, g', b, \bar{g}, b$);
  - Try2Add2G(t, g, g')
- **foreach** $(g', b', c) \in push_t(g, a)$ **do**
  - Try2Add2SI(t, g', b', g', b');
  - Try2Add2II($t, g', b', c, \bar{g}, b$);
  - Try2Add2G(t, g, g')
- **foreach** $(\bar{b}, g', c) \in \widehat{pop}(t, g, a)$ **do**
  - $\widehat{relSI}(t, \bar{g}, b, \bar{b})$.add($g', c$);
  - $\widehat{relSIG}(t, \bar{g}, b, \bar{b})$.add($g, g'$);
  - **if** $(\bar{g}, b, \bar{b}) \in relIF_t$ **then** Try2Add2SF(t, g', c);
  - **foreach** $\hat{g}$, d *such that* $(\bar{g}, b, \bar{b}, \hat{g}, d) \in relII_t$ **do**
    - Try2Add2SI($t, g', c, \hat{g}, d$)
- **foreach** $g' \in \mathrm{Glob}$ *such that* $guar(g, g') \setminus \{t\} \neq \emptyset$ **do**
  - Try2Add2SI($t, g', a, \bar{g}, b$)

```
// Process a new edge from Glob × Frame to Glob × Frame:
procedure ProcessII(itemAddr):
    // Precondition: itemAddr ≠ NULL.
    (t, g, a, b, ḡ, c) := itemAddr → data;
    relII_t.add(g, a, b, ḡ, c);
    foreach (g', d) ∈ relSI^(t, g, a, b) do Try2Add2SI(t, g', d, ḡ, c);
    foreach (ĝ, g') ∈ relSIG^(t, g, a, b) do Try2Add2G(t, ĝ, g');

// Process a new edge from Glob × Frame to f:
procedure ProcessIF(itemAddr):
    // Precondition: itemAddr ≠ NULL.
    (t, g, a, b) := itemAddr → data;
    relIF_t.add(g, a, b);
    foreach (g', c) ∈ relSI^(t, g, a, b) do Try2Add2SF(t, g', c);
    foreach (ḡ, g') ∈ relSIG^(t, g, a, b) do Try2Add2G(t, ḡ, g');

// The topmost, main procedure:
procedure TMR_Main:
    Data: boolean proceed := true
    Initialize();
    while proceed do
        proceed := false;
        if workG.is_nonempty() then
            ((g, g'), t) := workG.remove_any();
            ProcessG(g, g', t);
            proceed := true
        itemAddr := workSF.remove_any();
        if itemAddr ≠ NULL then
            ProcessSF(itemAddr);
            proceed := true
        itemAddr := workSI.remove_any();
        if itemAddr ≠ NULL then
            ProcessSI(itemAddr);
            proceed := true
        itemAddr := workII.remove_any();
        if itemAddr ≠ NULL then
            ProcessII(itemAddr);
            proceed := true
        itemAddr := workIF.remove_any();
        if itemAddr ≠ NULL then
            ProcessIF(itemAddr);
            proceed := true
```

H.4 Determining the running time

Let us fix some arbitrary execution.

First we compute the number of cells occupied by the used sets and the largest used address.

- The input occupies $\Theta(n(F^3\Gamma^2 + F^2\Gamma^2 + F^3\Gamma^2) + I(1+n)) = \Theta(n(I + F^3\Gamma^2))$ cells.
- The output occupies $\Theta(n(F\Gamma + F^2\Gamma^2 + F^3\Gamma^2 + F^2\Gamma)) = \Theta(nF^3\Gamma^2)$ cells.
- The working region contains 9 data structures. They occupy $\Theta(\Gamma^2 + nF^3\Gamma^2 + nF^3\Gamma^2 + nF^2\Gamma^3 + \Gamma^2 + nF\Gamma + nF^2\Gamma^2 + nF^3\Gamma^2 + nF^2\Gamma) = \Theta(nF^2\Gamma^2(F+\Gamma))$ cells.

The largest used address from the input transition relations (before init) has length $\Theta(L(n) + L(F) + L(\Gamma))$. The start of the output (right after init) has address of length $\Theta(L(n(I + F^3\Gamma^2))) = \Theta(L(n) + L(F) + L(\Gamma) + L(I))$. The total number of used cells is $\Theta(n(I + F^2\Gamma^2(F+\Gamma)))$. So the largest overall used address has length $\Theta(L(n) + L(I + F^3\Gamma^2 + F^2\Gamma^3))) = \Theta(L(n) + L(F) + L(\Gamma) + L(I))$. We let C be the term $L(n) + L(F) + L(\Gamma) + L(I)$.

If we would use dynamic data structures instead of fixed-size arrays, we would obtain just an upper bound $\mathcal{O}(\dots)$ instead of lower-and-upper bounds $\Theta(\dots)$ in the description of the number of used cells.

Next we list subroutines and their running times.

- Every `Try2Add2...` procedure: $\mathcal{O}(C)$.
- `Initialize`: $\mathcal{O}\Big(n\Big(F^3\Gamma^2C + F\Gamma\big(F^2\Gamma C + F(F\Gamma C + \Gamma^2 C + F\Gamma C)\big)\Big) + \Gamma^2 C + nF\Gamma C + nF^2\Gamma^2 C + nF^3\Gamma^2 C + nF^2\Gamma C + InC\Big) = \mathcal{O}\Big(n\big(F^3\Gamma^2 C + F\Gamma \cdot F\Gamma(F + \Gamma)C\big) + nF^3\Gamma^2 C + InC\Big) = \mathcal{O}\Big(n\big(F^2\Gamma^2(F+\Gamma) + I\big)C\Big)$.
- `ProcessG`: $\mathcal{O}\Big(C + \max\big\{(n-1)(FC + F^2\Gamma C),\, C + FC + F^2\Gamma C\big\}\Big) = \mathcal{O}(nF^2\Gamma C)$.
- `ProcessSF`: $\mathcal{O}(C + F\Gamma(C+C) + F^2\Gamma(C+C+C) + \Gamma C) = \mathcal{O}(F^2\Gamma C)$.
- `ProcessSI`: $\mathcal{O}\big(C + F\Gamma(C+C) + F^2\Gamma(C+C+C) + F^2\Gamma(C+C+C+F\Gamma C) + \Gamma C\big) = \mathcal{O}(F^3\Gamma^2 C)$.
- `ProcessII`: $\mathcal{O}(C + F\Gamma C + \Gamma^2 C) = \mathcal{O}(\Gamma(F+\Gamma)C)$.
- `ProcessIF`: Same.

Now we turn to the `TMR_Main` routine.

Notice that *guar* and *rel*..., viewed as sets of tuples, isotonically grow in time. Notice that the *workG* map behaves as follows for each $g, g' \in \mathrm{Glob}$ along the execution:

- If at some point of time $guar(g,g')$ is undefined, and $workG$ is nonempty, and $workG$.`remove_any()` returns some $((g,g'),t)$, then $guar(g,g')$ grows from size 0 to size 1 in the immediately succeeding `ProcessG` call, and t will always stay in $guar(g,g') \setminus workG(g,g')$.
- If at some point of time $guar(g,g') = \{t\}$ for some t, and $workG$ is nonempty, and $workG$.`remove_any()` returns some $((g,g'),s)$, then $s \neq t$ and $guar(g,g')$ will grow to size 2, and s and t will always stay in $guar(g,g') \setminus workG(g,g')$, and $workG(g,g')$ will have size at most 1.

– If at some point of time $|guar(g,g')| = 2$, and *workG* is nonempty, and the call *workG*.`remove_any()` is executed, then $guar(g,g')$ will remain as it is, and $workG(g,g')$ was of size 1 before the call and will stay undefined forever after the call.

Hence, for each pair $(g,g') \in \mathrm{Glob}^2$, the calls *workG*.`remove_any()` inside the execution return $((g,g'),\_)$ at most thrice. Thus, `ProcessG` will be called $\mathcal{O}(\Gamma^2)$ times, and the cost of all calls of `ProcessG` will be $\mathcal{O}(\Gamma^2 \cdot nF^2\Gamma C) = \mathcal{O}(nF^2\Gamma^3 C)$.

For $XY \in \{SF, SI, II, IF\}$, the total cost of all invocations of `ProcessXY` is as follows.

– *workSF*.`remove_any()` will return a non-NULL result $\mathcal{O}(nF\Gamma)$ times. Thus the total cost of all the `ProcessSF` calls will be $\mathcal{O}(nF\Gamma \cdot F^2\Gamma C) = \mathcal{O}(nF^3\Gamma^2 C)$.
– *workSI*.`remove_any()` will return a non-NULL result $\mathcal{O}(nF^2\Gamma^2)$ times. Thus the total cost of all the `ProcessSI` calls will be $\mathcal{O}(nF^2\Gamma^2 \cdot F^3\Gamma^2 C) = \mathcal{O}(nF^5\Gamma^4 C)$.
– *workII*.`remove_any()` will return a non-NULL result $\mathcal{O}(nF^3\Gamma^2)$ times. Thus the total cost of all the `ProcessII` calls will be $\mathcal{O}(nF^3\Gamma^2 \cdot \Gamma(F+\Gamma)C) = \mathcal{O}(nF^3\Gamma^3(F+\Gamma)C)$.
– *workIF*.`remove_any()` will return a non-NULL result $\mathcal{O}(nF^2\Gamma)$ times. Thus the total cost of all the `ProcessIF` calls will be $\mathcal{O}(nF^2\Gamma \cdot \Gamma(F+\Gamma)C) = \mathcal{O}(nF^2\Gamma^2(F+\Gamma)C)$.

The number of our loop iterations is thus $\mathcal{O}(nF\Gamma + nF^2\Gamma^2 + nF^3\Gamma^2 + nF^2\Gamma) = \mathcal{O}(nF^3\Gamma^2)$. Thus the total cost of all the `remove_any` calls, checking non-emptiness, non-NULL-tests, and manipulations with *proceed* is $\mathcal{O}(nF^3\Gamma^2 C)$. Hence, the total cost of the `TMR_Main` routine is $\mathcal{O}\Big(n\big(F^2\Gamma^2(F+\Gamma)+I\big)C + nF^2\Gamma^3 C + nF^3\Gamma^2 C + nF^5\Gamma^4 C + nF^3\Gamma^3(F+\Gamma)C + nF^2\Gamma^2(F+\Gamma)C + nF^3\Gamma^2 C\Big) = \mathcal{O}(n(F^5\Gamma^4+I)C)$.

Notice that in terms of the size $s \geq n(F^3\Gamma^2+I)$ of the input, which does account for the (potentially small) lengths of data in the cells, the TMR algorithm runs in $\mathcal{O}\big((nF^5\Gamma^4+nI)C\big) \subseteq \mathcal{O}\Big(\big((nF^3\Gamma^2)^2+nI\big)\big(L(n)+L(F)+L(\Gamma)+L(I)\big)\Big) \subseteq \mathcal{O}\big((s^2+s)L(s)\big) = \mathcal{O}\big(s^2L(s)\big)$ time, in particular, subcubic in the size of the input.

A closer examination reveals that the TMR POP rule, applied to $\bar{v} \in \mathrm{Glob} \times \mathrm{Frame}$, is the main complexity generator. Compared to similar inference systems in the literature (which have restricted push and pop operations [25]), our rule is rather general. As the reader might have noticed, we are already employing some optimizations (data structures $\widehat{rel\ldots}$). We speculate that the TMR POP rule could be sped up more—which is not our main goal in this presentation.

The shown results improve on [12] by naming a machine model with precisely defined execution cost, providing a low-level algorithm, and showing a proof of the running time which is asymptotically smaller in n than the bound from [12]. Without going into details, an off-the-shelf application of [20] would give a cubic time in the size of their input[3], which is the number of initial/push/internal/pop rules that define the model-checked program, while our running-time bound is a better subcubic one in the size of our input. (To stay clear: we do not claim that our TMR is strictly

[3] We thank Helmut Seidl for deriving this result.

overall better or worse than the two approaches, since [12] and an approach based on [20] involve different inputs and different cost measures.)