Multithreaded-Cartesian Abstract Interpretation of Multithreaded Recursive Programs Is Polynomial

Alexander Malkis(✉)

Technische Universität München, Munich, Germany

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.

The author greatly acknowledges useful comments and suggestions from Neil Deaton Jones.

© Springer International Publishing Switzerland 2015
M. Bojańczyk et al. (Eds.): RP 2015, LNCS 9328, pp. 114–127, 2015.
DOI: 10.1007/978-3-319-24537-9_11

It is known that in the finite-state case the membership problem for the strongest multithreaded-Cartesian inductive invariant is in PTIME [16]. We extend this result to programs in which each thread has a potentially unbounded stack. Moreover, 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 ith automaton describes an overapproximation of the set of pairs (shared state, stack word of the ith 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 [17].
– 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 [15].

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 [20], followed by thread-modular reasoning of C. B. Jones [12], and the Flanagan-Qadeer model-checking method for nonrecursive programs [10]. 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 [9] and later rediscovered in [14,16].

Flanagan and Qadeer [10] 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 [10]. 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 [19] 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 [7,24]. Alternatively, one can allow certain forks and joins but restrict communication only to threads that do not have ongoing procedure calls [5]. 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 [21] or be satisfied with an approximate analysis, which can be sound [4] or complete [13,22] (for every choice of parameters), but never both. If shared-memory communication is replaced by rendezvous, verification is still undecidable [23].

The summarization idea behind TMR dates back to the works of Büchi [6]. 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 [18], for imperative stack-manipulating programs with a rich set of operations on the stack [2], and in implementations of partial evaluators [11].

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

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

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. E.g., 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.)
- $\mathrm{init} \subseteq \mathrm{Glob}\times(\mathrm{Frame}^+)^n$ is such that $\forall\,(g, l) \in \mathrm{init},\ t\in n\colon\ |l_t| = 1$. (By $l_t = l(t)$ we indicate the tth component of $l \in (\mathrm{Frame}^+)^n$. 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 $\overset{\downarrow}{\sqcup}_t \subseteq (\mathrm{Glob}\times \mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame}\times\mathrm{Frame})$, $\underline{\sqcup}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame})^2$, and $\overset{\uparrow}{\sqcup}_t \subseteq (\mathrm{Glob}\times \mathrm{Frame}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame})$. (These are sets of push, internal, and pop transitions of thread t, respectively.)

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

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

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

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

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

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

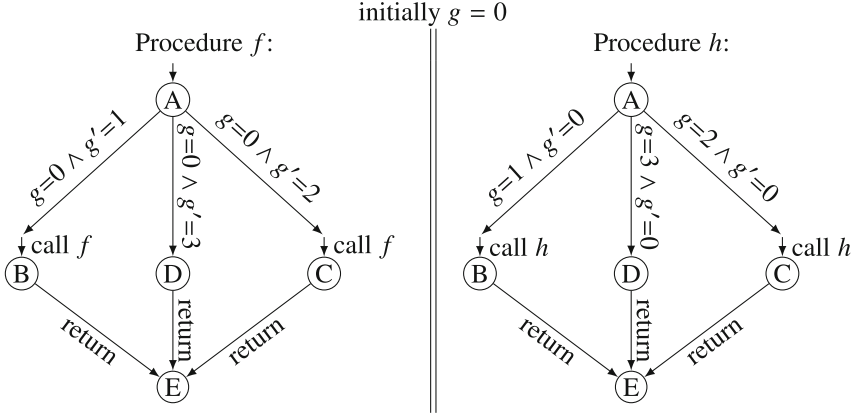
which is the power set of the set of program states, and the successor map

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

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

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

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 = $\{(0,(\mathrm{A,A}))\}$, $\sqcup\!\!\!\downarrow_0 = \{((0,\mathrm{A}),(1,\mathrm{A,B})), ((0,\mathrm{A}),(2,\mathrm{A,C}))\}$, $\uplus_0 = \{((0,\mathrm{A}),(3,\mathrm{D}))\}$, $\sqcup\!\!\!\uparrow_0 = \{((g,y,z),(g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B,C,D}\} \wedge z{\in}\mathrm{Frame}\}$, $\sqcup\!\!\!\downarrow_1 = \{((1,\mathrm{A}),(0,\mathrm{A,B})),((2,\mathrm{A}),(0,\mathrm{A,C}))\}$, $\uplus_1 = \{((3,\mathrm{A}),(0,\mathrm{D}))\}$, and $\sqcup\!\!\!\uparrow_1 = \{((g,y,z),(g,z)) \mid g{\in}\mathrm{Glob} \wedge y \in \{\mathrm{B,C,D}\} \wedge z{\in}\mathrm{Frame}\}$.

One can show that the strongest inductive invariant is

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

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 [14]. 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 $(D,\subseteq)$.

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

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

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 $\mathfrak{f} \notin \text{Glob} \,\dot\cup\, (\text{Glob}\times\text{Frame})$. Let

$$V = \text{Glob}\,\dot\cup\,\{(g',b) \mid \exists\, g\in\text{Glob}, a,c \in \text{Frame}, t\in n\colon ((g,a),(g',b,c)) \in \,\uplus_t\}\,\dot\cup\,\{\mathfrak{f}\}\,.$$

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

$$\text{(TMR INIT)}\ \frac{(g,l)\in\text{init}}{g \xrightarrow[t]{l_t} \mathfrak{f}}\ t\in n \qquad \text{(TMR STEP)}\ \frac{((g,a),(g',b))\in \uplus_t \qquad g\xrightarrow[t]{a} v}{g' \xrightarrow[t]{b} v \qquad (g,g')\in G_t}\ t\in n$$

$$\text{(TMR PUSH)}\ \frac{g\xrightarrow[t]{a} v \qquad ((g,a),(g',b,c))\in \uplus_t}{g'\xrightarrow[t]{b}(g',b)\xrightarrow[t]{c} v \qquad (g,g')\in G_t}$$

$$\text{(TMR POP)}\ \frac{g\xrightarrow[t]{a} v\xrightarrow[t]{b}\bar v \qquad ((g,a,b),(g',c))\in \uplus_t}{g'\xrightarrow[t]{c}\bar v \qquad (g,g')\in G_t}$$

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

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 118, 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.

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

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

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

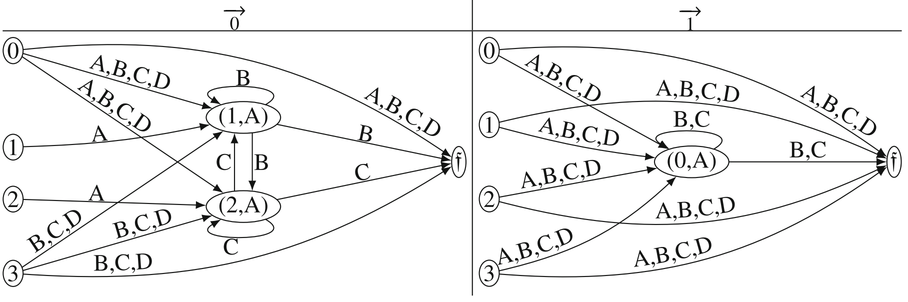


Fig. 1. Automata constructed by TMR for our example. Each arrow on the left carries the lower index 0; each arrow on the right carries the lower index 1.

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

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

5.3 Computing Multithreaded-Cartesian Semantics

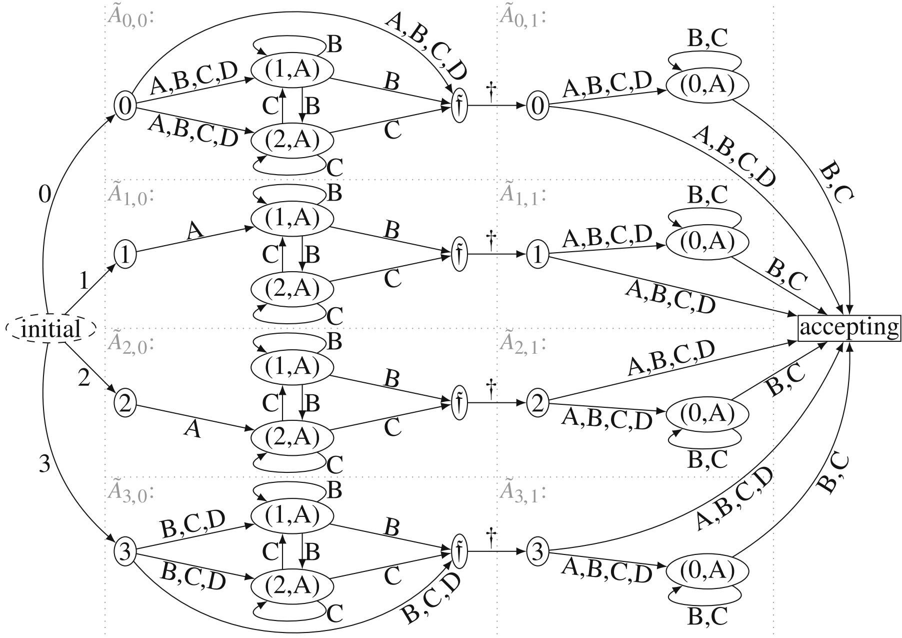
For actual computations, which we discuss now, let us assume finite n, Glob, and Frame till the end of § 5.3.

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 tth thread does not occur in computations of the program ($g\in$Glob, $l\in$Loc, $t\in n$). If we are interested in checking non-reachability of single program states, executing TMR also suffices: for $(g, \bar{l}) \in$ 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 $\mathcal{O}\big(n(|\text{init}| + |\text{Glob}|^4|\text{Frame}|^5)(L(|\text{init}|) + L(n) + L(|\text{Glob}|) + L(|\text{Frame}|))\big)$ time, where $L(x)$ is the length of the binary representation of $x\in\mathbb{N}_0$. 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 ($(g, t) \in$ Glob×Loc), obtaining automata $\tilde{A}_{g,t}$ ($(g, t) \in$ Glob×Loc). If we wish to obtain a deterministic automaton at the end, we additionally determinize all $\tilde{A}_{g,t}$ ($(g, t) \in$ Glob×Loc). Second, for each $g\in$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 Frame$\dot{\cup}\{\dagger\}$ (where $\dagger\notin$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

$\tilde{A}_{g,0}$ ($g\in$Glob). Thus, (1) can be viewed as a regular language. The nondeterministic, ε-free automaton can be constructed (including executing TMR) in the same $\mathcal{O}\big(n(|\text{init}| + |\text{Glob}|^4|\text{Frame}|^5)(L(|\text{init}|) + L(n) + L(|\text{Glob}|) + L(|\text{Frame}|))\big)$ 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:*

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

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 [10]. Its definition does not depend on the internal structure of Loc and $\rightsquigarrow_t$ ($t{\in}n$).

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:

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

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

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, 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 [15].

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,

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

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 $(\mathfrak{P}(\text{Glob}\times\text{Loc}))^n \times (\mathfrak{P}(\text{Glob}^2))^n$:

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

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:

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

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:

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

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

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

One can obtain G and $\rightarrow$ inductively by generating iterates $((\underset{t}{\rightarrow}_i)_{t<n}, (G_{t,i})_{t<n})$ 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 $\underset{t}{\rightarrow}_0$ for all $t{<}n$ and obtain $G_{t,i+1}$ and $\underset{t}{\rightarrow}_{i+1}$ for all $t{<}n$ by applying the rules of TMR exactly once to $G_{t,i}$ and $\underset{t}{\rightarrow}_i$ 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:

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

Sloppily speaking, the derivation operator of TMR produces graphs on V, and larger iterates contain larger graphs. Given a walk in an edge set $\underset{t}{\rightarrow}_i$, 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 $\underset{t}{\rightarrow}_i$.

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 $\mathrm{tr}(\dots)$ (inner induction). Formally, we show:

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

$$\mathrm{tr}(t, i, g, (\bar{g}, b), w) = j \Rightarrow (\bar{g}, b) \overset{\widetilde{G}}{\rightsquigarrow}{}_t^* (g, w),$$

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

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:

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

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, and "$\supseteq$" follows from § 7.2.

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. 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 [8]. 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. Part of this work was executed while at IMDEA Software, Spain. I also acknowledge the financial support of the projects of the German Federal Ministry for Education and Research, IDs 01IS11035 and 01IS13020. Furthermore, I acknowledge additional typesetting support of Springer-Verlag.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Symposium on Theory of Computing, pp. 202–211. ACM (2004)
2. Andersen, N., Jones, N.D.: Generalizing Cook's transformation to imperative stack programs. In: Karhumäki, J., Rozenberg, G., Maurer, H.A. (eds.) Results and Trends in Theoretical Computer Science. LNCS, vol. 812, pp. 1–18. Springer, Heidelberg (1994)
3. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of Multi-pushdown Automata Is 2ETIME-Complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
4. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. International Journal of Foundations of Computer Science **14**(4), 551–582 (2003)
5. Bozzelli, L., La Torre, S., Peron, A.: Verification of well-formed communicating recursive state machines. Theoretical Computer Science **203**(2-3), 382–405 (2008)
6. Büchi, J.R.: Regular canonical systems. Archiv für mathematische Logik und Grundlagenforschung 6, 91–111 (1962/1963)
7. Burkart, O., Steffen, B.: Pushdown processes: parallel composition and model checking. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 98–113. Springer, Heidelberg (1994)
8. Chaki, S., Clarke, E., Kidd, N., Reps, T., Touili, T.: Verifying Concurrent Message-Passing C Programs with Recursive Calls. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 334–349. Springer, Heidelberg (2006)

9. Cousot, R.: Fondements des méthodes de preuve d'invariance et de fatalité de programmes paralléles. Ph.D. thesis, Institut national polytechnique de Lorraine, pp. 4–118(4–119), pp. 4–120, 1985. §4.3.2.4.3
10. Flanagan, C., Qadeer, S.: Thread-Modular Model Checking. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 213–224. Springer, Heidelberg (2003)
11. Hansen, T.A., Nikolajsen, T., Träff, J.L., Jones, N.D.: Experiments with implementations of two theoretical constructions. In: Meyer, A.R., Taitslin, M.A. (eds.) Logic at Botik 1989. LNCS, vol. 363, pp. 119–133. Springer, Heidelberg (1989)
12. Jones, C. B.: Tentative steps toward a development method for interfering programs. ACM Trans. Program. Lang. Syst. 5(4), 596–619 (1983)
13. La Torre, S., Napoli, M., Parlato, G.: A Unifying Approach for Multistack Pushdown Automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 377–389. Springer, Heidelberg (2014)
14. Malkis, A.: Cartesian abstraction and verification of multithreaded programs. Ph.D. thesis, Albert-Ludwigs-Universität Freiburg (2010)
15. Malkis, A.: Multithreaded-Cartesian abstract interpretation of multithreaded recursive programsis polynomial. Technical report (2015). http://www4.in.tum.de/malkis/Malkis-MultCartAbstIntOfMultRecProgIsPoly_techrep.pdf
16. Malkis, A., Podelski, A., Rybalchenko, A.: Thread-Modular Verification Is Cartesian Abstract Interpretation. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) ICTAC 2006. LNCS, vol. 4281, pp. 183–197. Springer, Heidelberg (2006)
17. Mehlhorn, K.: Data structures and algorithms 1: sorting and searching. In: EATCS Monographs in Theoretical Computer Science, vol. 1. Springer (1984)
18. Mogensen, T.Æ.: WORM-2DPDAs: An extension to 2DPDAs that can be simulated in linear time. Inf. Process. Lett. **52**(1), 15–22 (1994)
19. Nielson, F., Nielson, R.H., Seidl, H.: Normalizable Horn Clauses, Strongly Recognizable Relations, and Spi. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 20–35. Springer, Heidelberg (2002)
20. Owicki, S.S.: Axiomatic proof techniques for parallel programs. Ph.D. thesis, Cornell University, department of computer science, TR 75–251, July 1975
21. Qadeer, S., Rajamani, S.K., Rehof, J.: Summarizing procedures in concurrent programs. In: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 245–255. ACM, New York (2004)
22. Qadeer, S., Rehof, J.: Context-Bounded Model Checking of Concurrent Software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
23. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. **22**(2), 416–430 (2000)
24. Schwoon, S.: Model-checking pushdown systems. Ph.D. thesis, Technische Universität München, June 2002