Proving (Security) Properties of Multithreaded Programs

Alexander Malkis

`malkis at sec dot in dot tum dot de`

Chair of IT security
Prof. Dr. Claudia Eckert

Technische Universität München

Advanced topics in IT security
4. May 2016

Acknowledgments:

- Neil D. Jones,
- IMDEA Software Institute, Manfred Broy's research group at Technische Universität München,
- H. Seidl, J. Esparza, C. Broadbent, L. Mauborgne, A. Podelski, ...

Classifying trace-based properties—an informal overview

- Invariance property: expressible by `assert(...)` statements.
- Safety property: "nothing bad happens", a superclass of invariance properties.
- Checking a safety property can be reduced to checking an invariance property of (program $\|$ monitor).
- Liveness property: "something good eventually happens", almost disjoint from safety properties.
- Each property can be shown to be a conjunction of a safety and a liveness property.

Formalization in: Baier, Katoen, *Principles of Model-Checking*.

Security properties are often trace-based. E.g., confidentiality can be viewed as a safety property.

In this lecture:
invariance properties of *multithreaded* programs with *recursion*.

Multithreaded programs

Program: $(\mathrm{Glob}, \mathrm{Frame}, \mathrm{init}, ({\sqcup\!\!\!\!\downarrow}_t, {\sqcup\!\!\!\!-}_t, {\sqcup\!\!\!\!\uparrow}_t)_{t<n})$

- n: number of threads (ordinal).
- Interleaving semantics.
- Local state = stack contents;
  $\mathrm{Loc} = \mathrm{Frame}^+$.
- Thread state: $(g, \mathrm{stack\_word}) \in \mathrm{Glob} \times \mathrm{Loc}$
- Program state:
  (shared, ($\mathrm{stack\_word}_0$, $\mathrm{stack\_word}_1$, $\mathrm{stack\_word}_2$, ...));
  $\mathrm{State} \;=\; \mathrm{Glob} \times \mathrm{Loc}^n$.
- Initially, each stack contains exactly one letter.

For each $t < n$:

- ${\sqcup\!\!\!\!\downarrow}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame}\times\mathrm{Frame})$,
- ${\sqcup\!\!\!\!-}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame})$,
- ${\sqcup\!\!\!\!\uparrow}_t \subseteq (\mathrm{Glob}\times\mathrm{Frame}\times\mathrm{Frame}) \times (\mathrm{Glob}\times\mathrm{Frame})$.

Semantics

Operational semantics of each thread $t<n$:

$$\frac{((g,a),(g',b,c)) \in \mathrel{\sqcup\!\!\!\!\downarrow}_t \qquad u \in \mathsf{Frame}^*}{(g,au) \rightsquigarrow_t (g',bcu)}$$

$$\frac{((g,a),(g',b)) \in \mathrel{\sqcup\!\!\!\!-}_t \qquad u \in \mathsf{Frame}^*}{(g,au) \rightsquigarrow_t (g',bu)}$$

$$\frac{((g,a,b),(g',c)) \in \mathrel{\sqcup\!\!\!\!\uparrow}_t \qquad u \in \mathsf{Frame}^*}{(g,abu) \rightsquigarrow_t (g',cu)}$$

As the operational semantics of the whole program we choose the so-called interleaving semantics. It is given by

the *concrete domain* $\quad D = \mathfrak{P}(\mathsf{State}),$

and the successor map

$$\begin{aligned} \mathsf{post} : D &\to D\,, \\ Q &\mapsto \{(g',\ell') \mid \exists\, t<n,\ (g,\ell)\in Q: \quad (g,\ell_t) \rightsquigarrow_t (g',\ell'_t) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \wedge \forall\, s<n: s\neq t \Rightarrow \ell_s = \ell'_s\}\,. \end{aligned}$$

Multithreaded shared-memory recursive programs

Multithreading + recursion is rare, but exists, both in the models and in real code.

- Model of the Bluetooth driver from Windows NT
- Model of the old synchronized `java.util.Vector`
- "Concurrent manipulation of binary search trees", H. T. Kung and Philip L. Lehman, 1980
- Parallel Merge Sort: merging sorted pairs of subsequences may happen in parallel for independent pairs of subsequences
- Cilkchess
- "A new multithreaded and recursive direct algorithm for parallel solution of the sparse linear systems", Ercan Selçuk Bölükbași, 2013
- ...

Invariants and inductive invariants

Regardless of the internal structure of init and post:
A set of states of a program is called

- an invariant iff it contains all states reachable from the initial ones:

$$S \textit{ invariant} \quad \stackrel{\text{def}}{\iff} \quad \mathrm{lfp}(\lambda\, Q \in D.\, \mathrm{init} \cup \mathrm{post}(Q)) \ \subseteq \ S\,.$$

- inductive iff it contains the initial states and is closed under the transition function, i.e.:

$$S \textit{ inductive} \quad \stackrel{\text{def}}{\iff} \quad \mathrm{init} \subseteq S \ \wedge \ \mathrm{post}(S) \subseteq S\,.$$

To prove an invariance property $P \subseteq \mathrm{State}$, it suffices to provide an inductive invariant $S \subseteq P$.

Escape undecidability through overapproximation

For multithreaded programs with unbounded stacks:

- Membership in the strongest inductive invariant: undecidable.
- Membership in a special-form inductive invariant, not necessarily the strongest one: perhaps decidable.

Multithreaded-Cartesian set of program states

Simplify for a moment: finite Glob $= \{0, 1, \ldots |\text{Glob}| - 1\}$, finite n.
A set $S \subseteq$ State is in multithreaded-Cartesian form iff
there are $L_{g,t} \subseteq$ Loc ($g \in$ Glob, $t < n$) s. t.

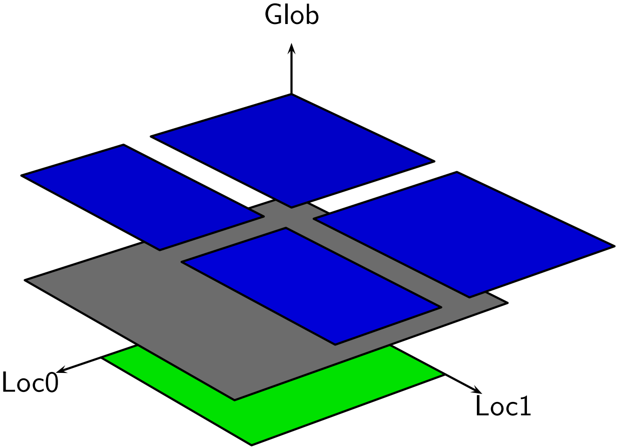
$$
\begin{aligned}
S \quad = \quad & \{0\} \times L_{0,0} \times \ldots \times L_{0,n-1} \\
\cup\ & \{1\} \times L_{1,0} \times \ldots \times L_{1,n-1} \\
& \vdots \\
\cup\ & \{|\text{Glob}| - 1\} \times L_{|\text{Glob}|-1,0} \times \ldots \times L_{|\text{Glob}|-1,n-1} \,.
\end{aligned}
$$

Multithreaded-Cartesian set of program states

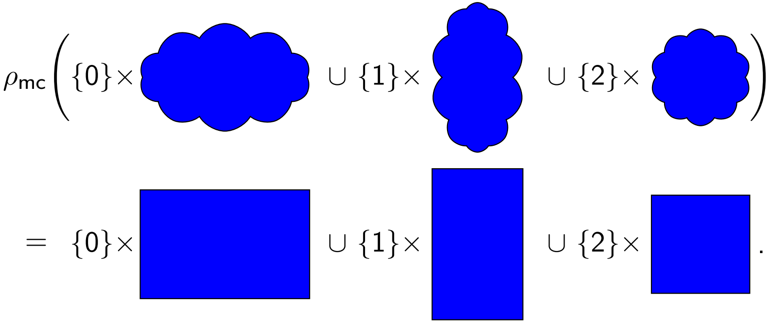
General case: arbitrary Glob, arbitrary *n*.

States $(g, \vec{\ell})$, $(g', \vec{\ell'})$ are equivalent iff $g = g'$.

A set $S \subseteq$ State is in multithreaded-Cartesian form iff the intersection of each equivalence class with S is a Cartesian product.



Multithreaded-Cartesian overapproximation



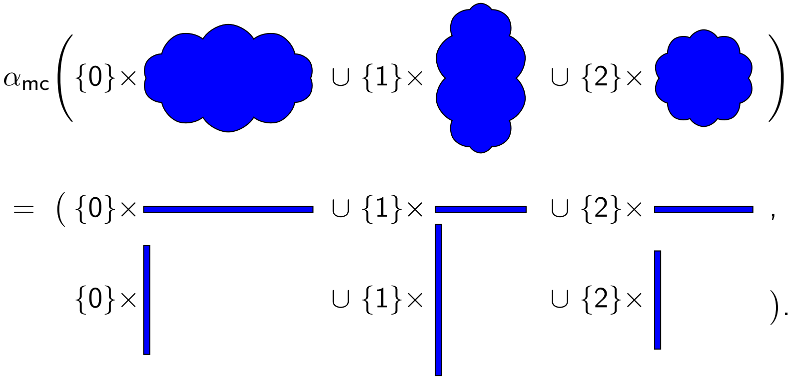
For $S \subseteq \mathsf{State}$,

$$\rho_{\mathsf{mc}}(S) \;=\; \{(g,\ell) \mid \forall\, t{<}n\, \exists\, \tilde{\ell} \in \mathsf{Loc}^n\colon\ \tilde{\ell}_t = \ell_t \wedge (g,\tilde{\ell}) \in S\}\,.$$

ρ_{mc} is an upper closure operator.

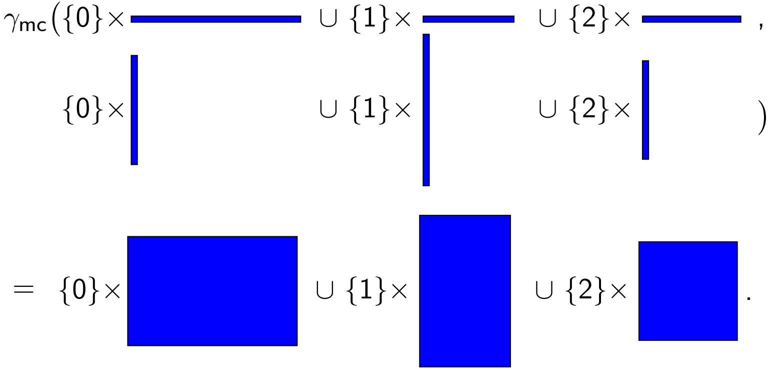
Multithreaded-Cartesian Galois-connection: abstraction

$$\begin{array}{llcl} \text{Concrete domain:} & D & = & \mathfrak{P}(\mathsf{State}) = \mathfrak{P}(\mathsf{Glob} \times \mathsf{Loc}^n). \\ \text{Abstract domain:} & D^{\#} & = & (\mathfrak{P}(\mathsf{Glob} \times \mathsf{Loc}))^n. \end{array}$$



Abstraction $\alpha_{\mathsf{mc}}(S) = \big(\{(g, \ell_t) \mid (g, \ell) \in S\}\big)_{t<n}$.

Multithreaded-Cartesian abstraction: concretization



Concretization $\gamma_{\mathsf{mc}}\big((A_t)_{t<n}\big) = \{(g,\ell) \mid \forall\, t<n\colon (g,\ell_t) \in A_t\}$.

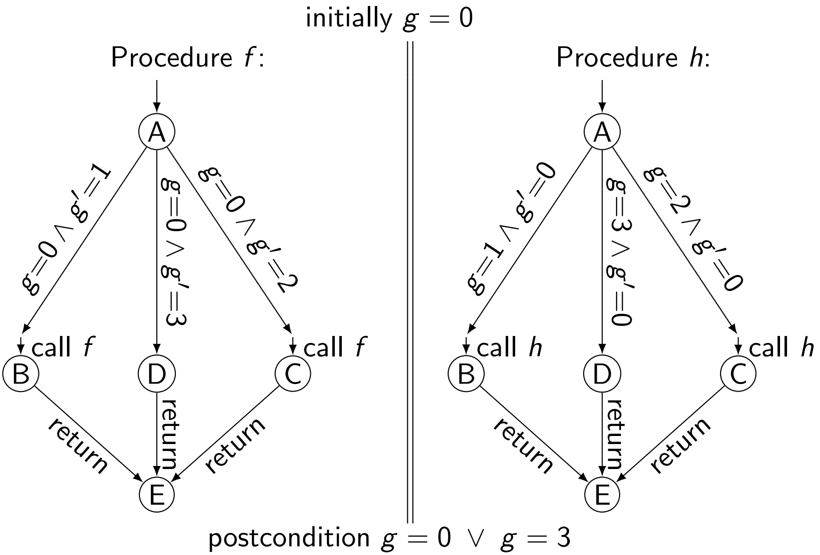
$\rho_{\mathsf{mc}} = \gamma_{\mathsf{mc}} \circ \alpha_{\mathsf{mc}}$

Multithreaded-Cartesian abstract interpretation

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

$$\mathsf{lfp}(\lambda\ A \in D^{\#}.\ \alpha_{\mathsf{mc}}(\mathsf{init} \cup \mathsf{post}(\gamma_{\mathsf{mc}}(A))))$$

Intricate example



The strongest inductive invariant is not context-free.

Strongest multithreaded-Cartesian inductive invariant for the example

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

Notice:

$$
(|w_1| = |w_2| = 1 \wedge (g, (w_1, w_2)) \in S) \quad \Rightarrow \quad (g = 0 \vee g = 3).
$$

The property would be proven by a multithreaded-Cartesian analysis generating (a finite representation of) S.
Viewed as a set of words, S is regular.
It can be generated!

TMR algorithm

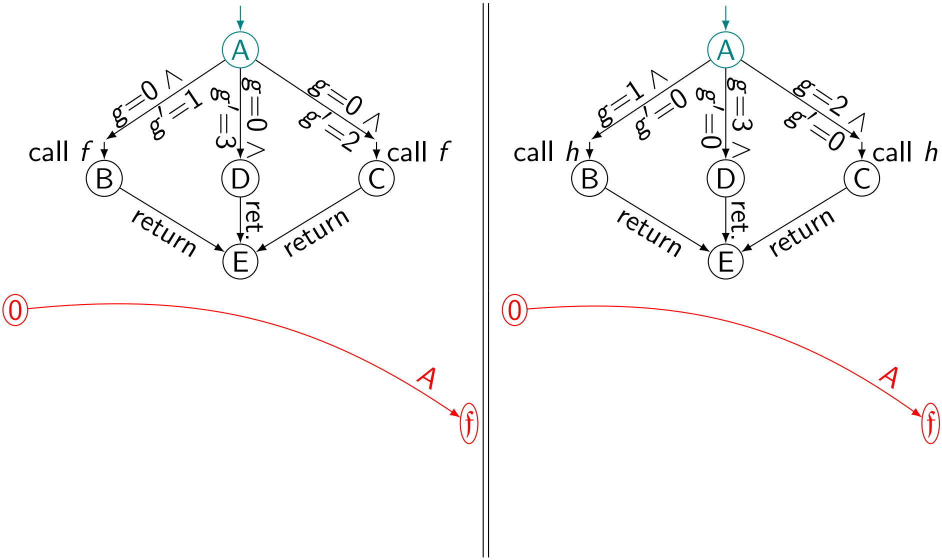
Generating a regular representation of

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

- Construct n NFAs simultaneously, one per thread.
- Each NFA describes a set of thread states ($\in$ Glob $\times$ Loc).
- Sequentially chain the NFAs.

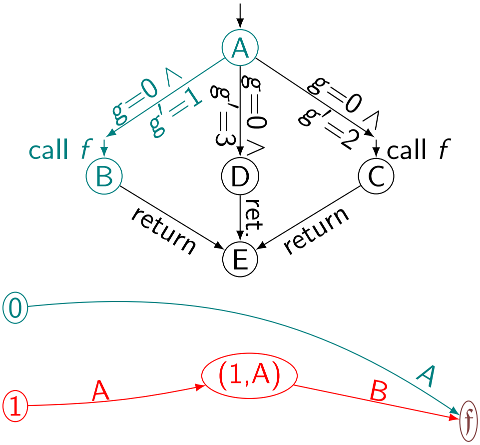
Generating automata for the left and right threads (1)

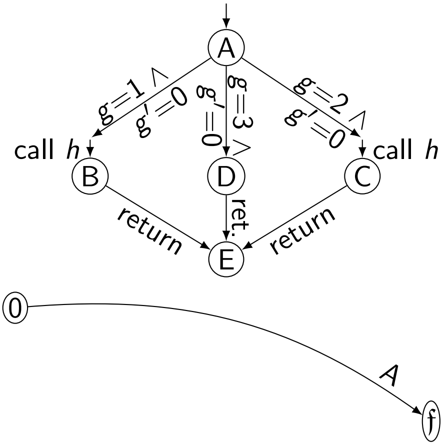
Procedure f: initially $g = 0$ Procedure h:



Generating automata for the left and right threads (2)

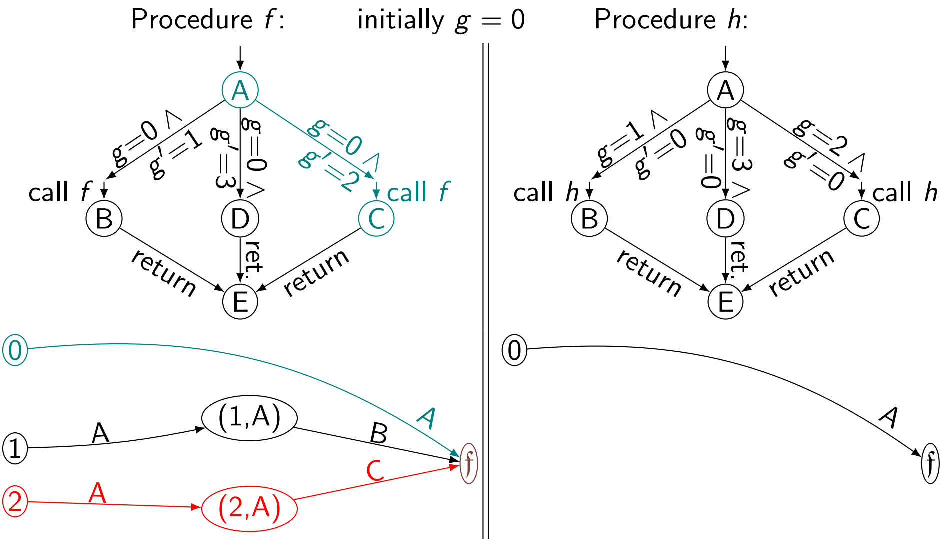
Procedure f: initially $g = 0$ Procedure h:





$$G_0 = \{(0,1)\}$$

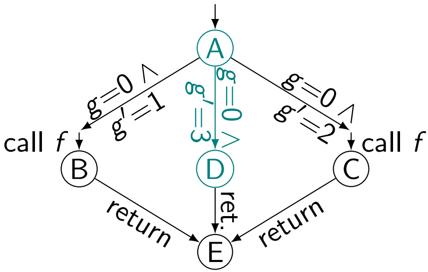
Generating automata for the left and right threads (3)

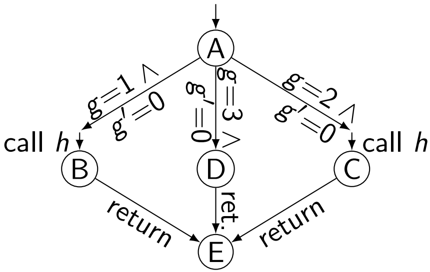


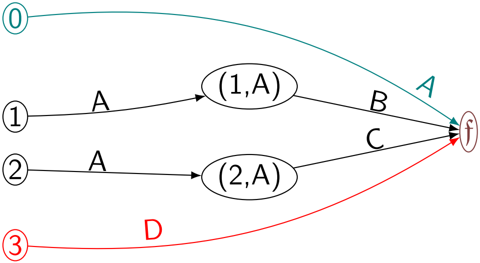
$$G_0 = \{(0,1), (0,2)\}$$

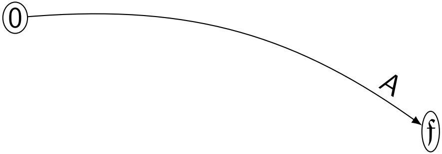
Generating automata for the left and right threads (4)

Procedure f: initially $g = 0$ Procedure h:





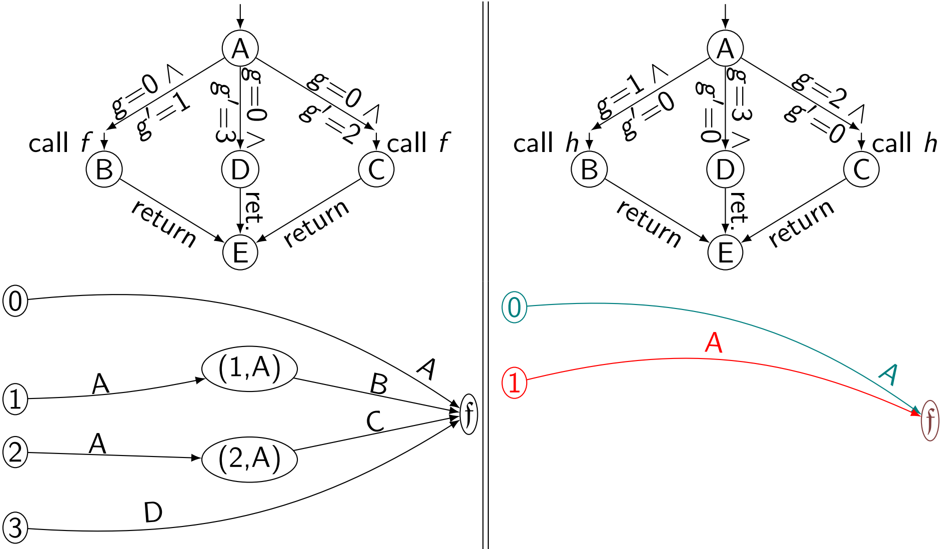




$$G_0 = \{(0,1),(0,2),(0,3)\}$$

Generating automata for the left and right threads (5)

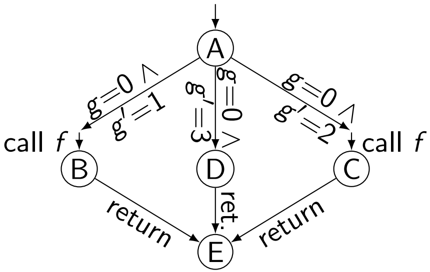
Procedure f: initially $g = 0$ Procedure h:

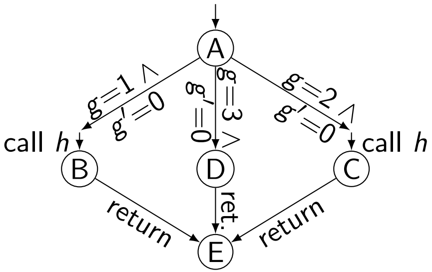


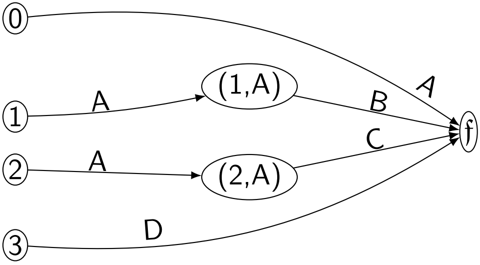
$$G_0 = \{(0,1),(0,2),(0,3)\}$$

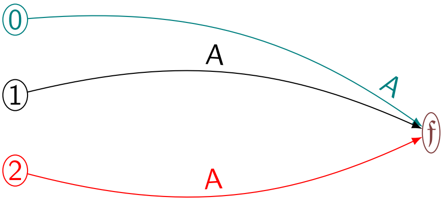
Generating automata for the left and right threads (6)

Procedure f: initially $g = 0$ Procedure h:





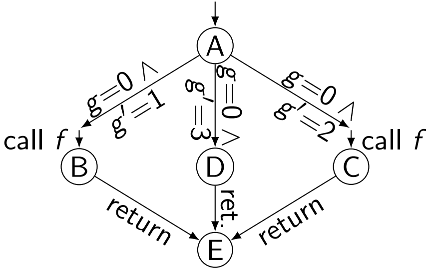


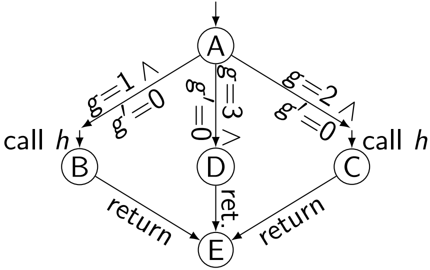


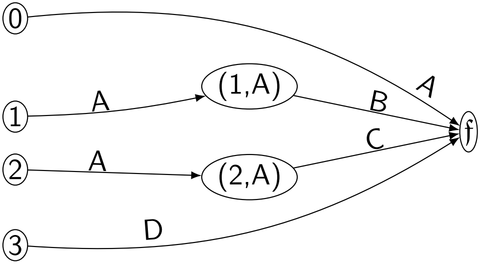
$$G_0 = \{(0,1), (0,2), (0,3)\}$$

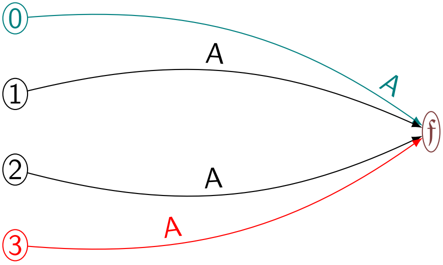
Generating automata for the left and right threads (7)

Procedure f: initially $g = 0$ Procedure h:





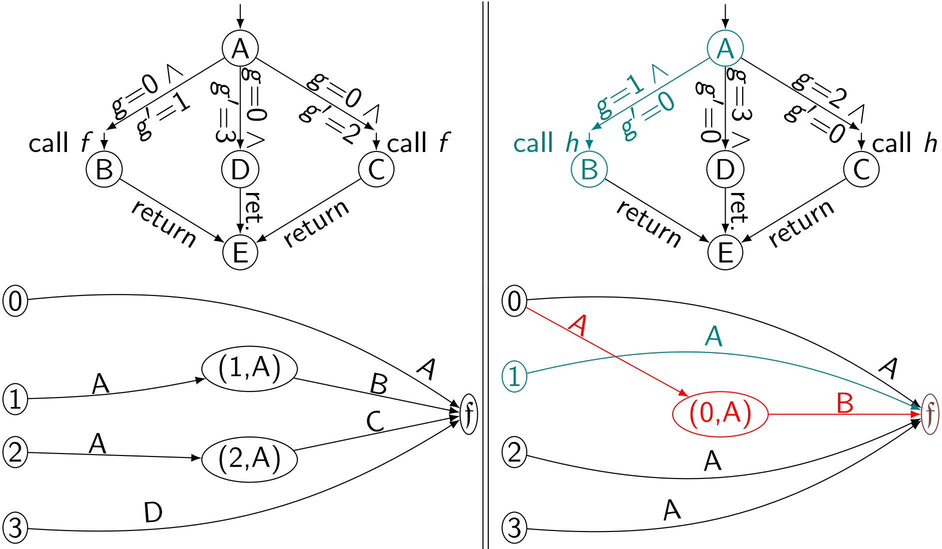




$$G_0 = \{(0,1),(0,2),(0,3)\}$$

Generating automata for the left and right threads (8)

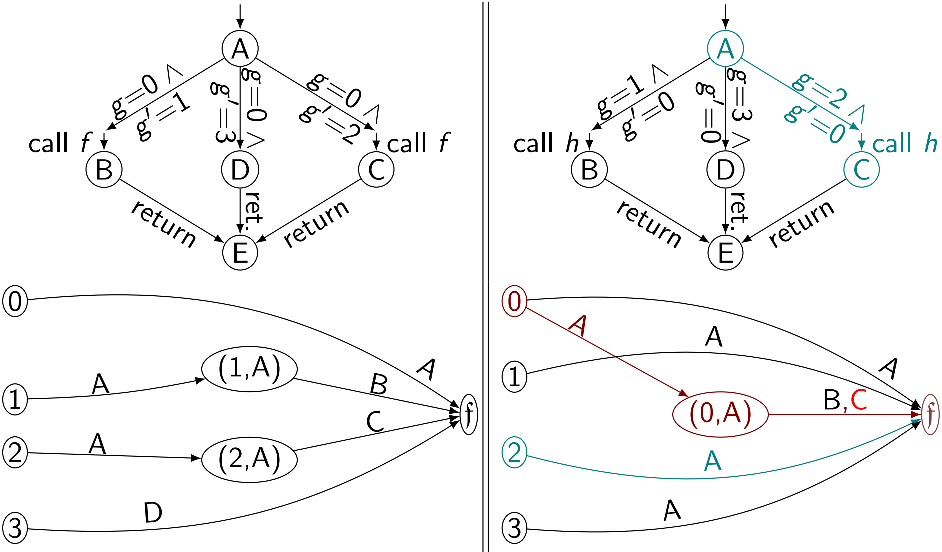
Procedure f: initially $g = 0$ Procedure h:



$G_0 = \{(0,1),(0,2),(0,3)\}$ $G_1 = \{(1,0)\}$

Generating automata for the left and right threads (9)

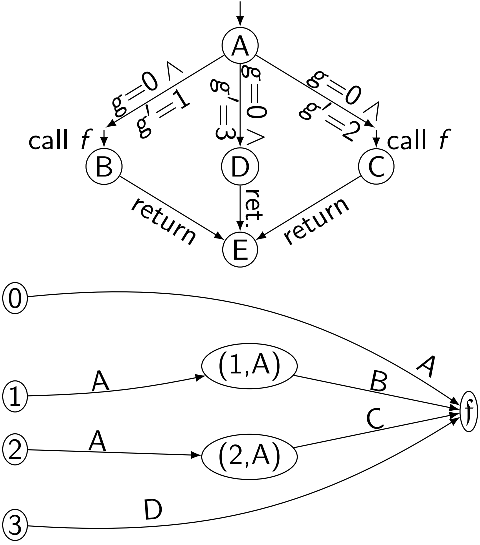
Procedure f: initially $g = 0$ Procedure h:



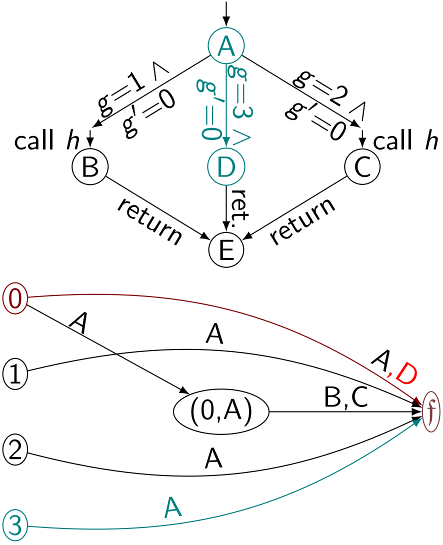
$G_0 = \{(0,1),(0,2),(0,3)\}$ $G_1 = \{(1,0),(2,0)\}$

Generating automata for the left and right threads (10)

Procedure f: initially $g = 0$ Procedure h:

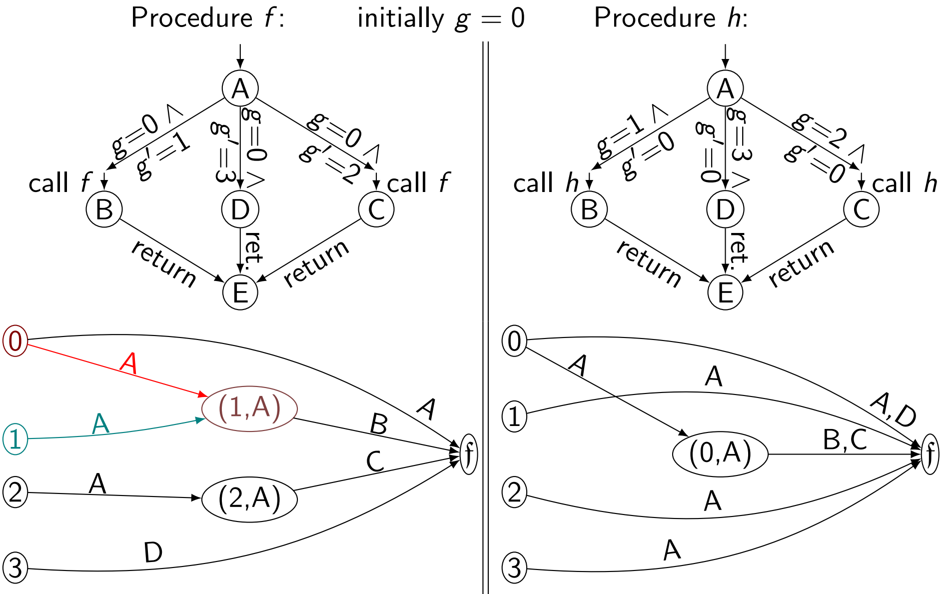


$G_0 = \{(0,1),(0,2),(0,3)\}$



$G_1 = \{(1,0),(2,0),(3,0)\}$

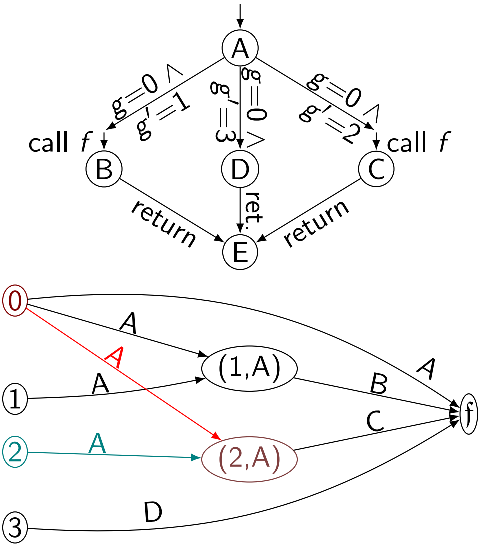
Generating automata for the left and right threads (11)



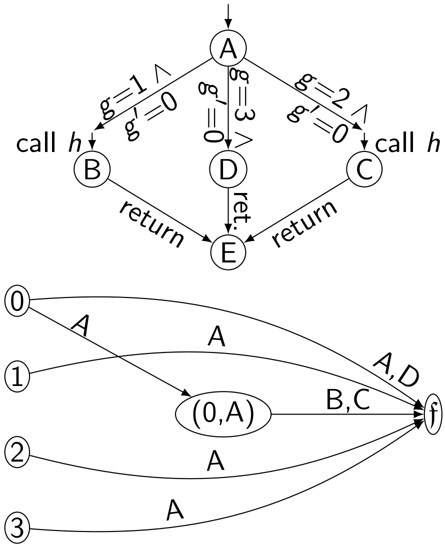
$G_0 = \{(0,1),(0,2),(0,3)\}$ $\qquad$ $G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (12)

Procedure f: initially $g = 0$ Procedure h:



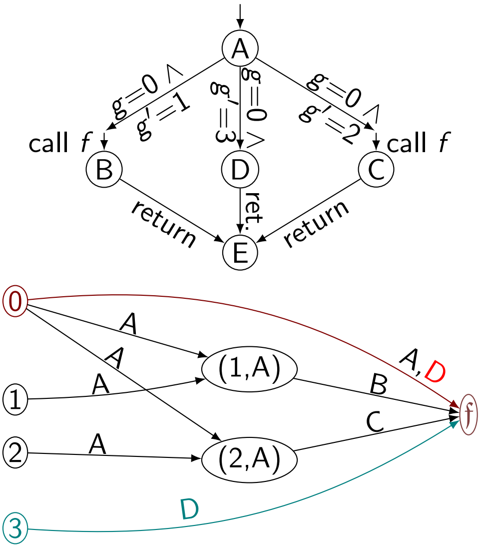
$G_0 \;=\; \{(0,1),(0,2),(0,3)\}$



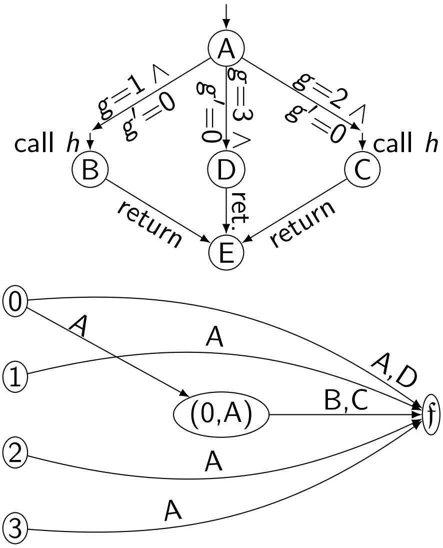
$G_1 \;=\; \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (13)

Procedure f: initially $g = 0$ Procedure h:



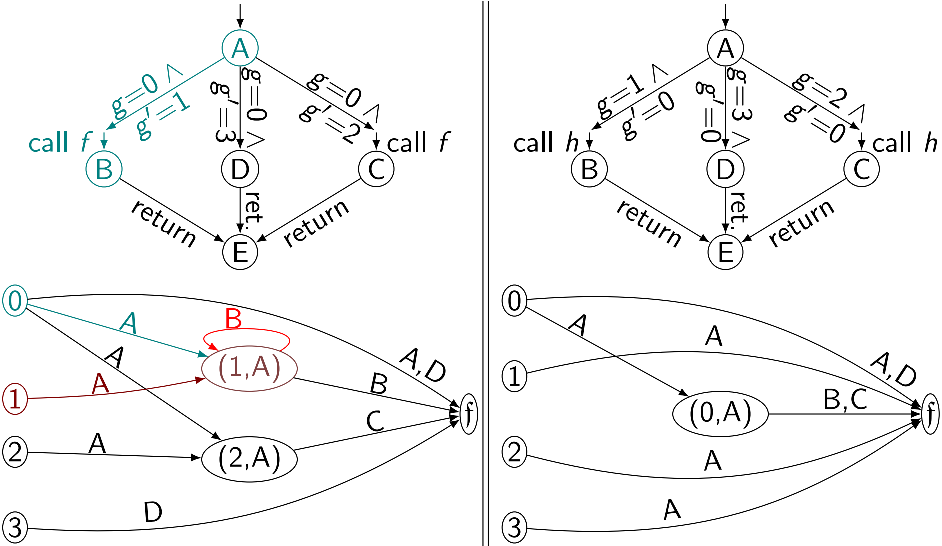
$G_0 = \{(0,1),(0,2),(0,3)\}$



$G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (14)

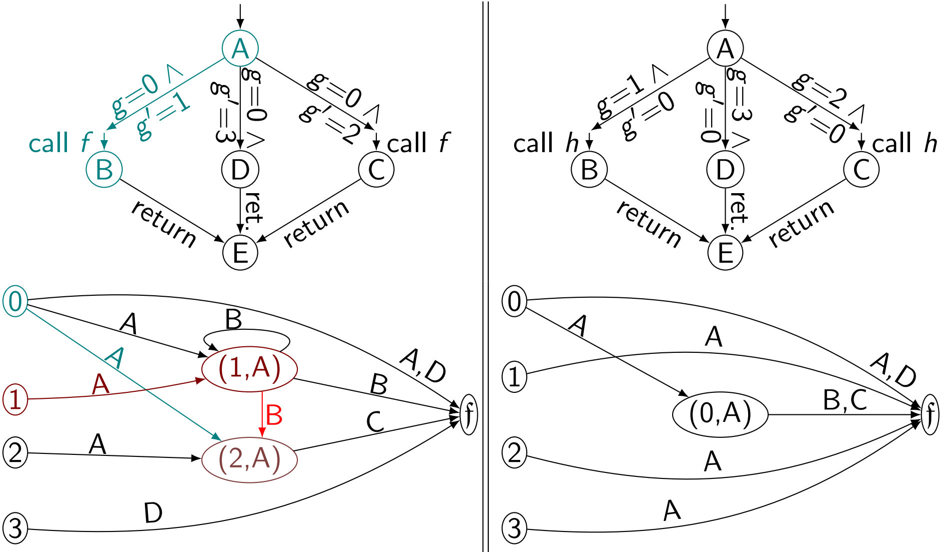
Procedure f: initially $g = 0$ Procedure h:



$G_0 = \{(0,1),(0,2),(0,3)\}$ $G_1 = \{(1,0),(2,0),(3,0)\}$

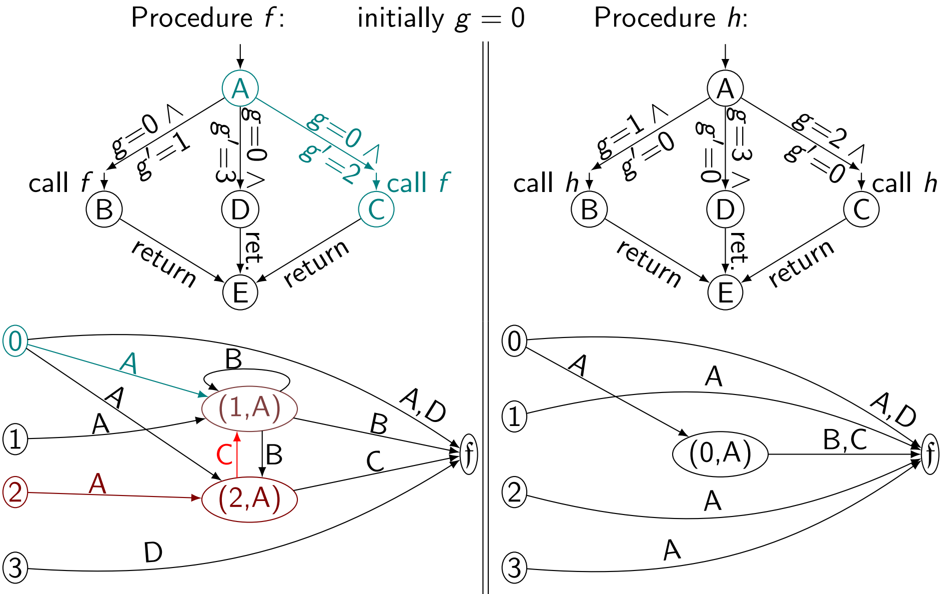
Generating automata for the left and right threads (15)

Procedure f: initially $g = 0$ Procedure h:



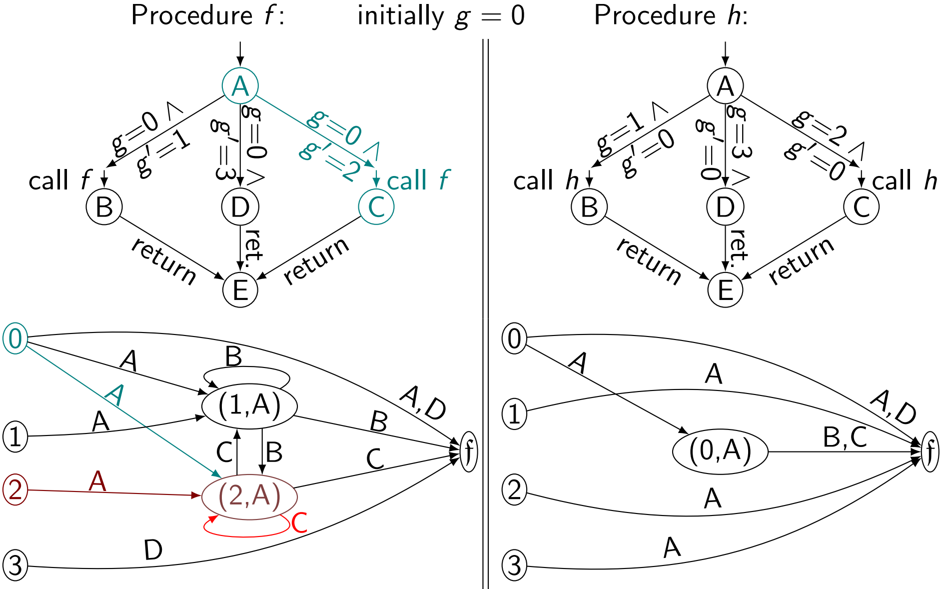
$G_0 = \{(0,1),(0,2),(0,3)\}$ $G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (16)



$G_0 = \{(0,1), (0,2), (0,3)\}$ $G_1 = \{(1,0), (2,0), (3,0)\}$

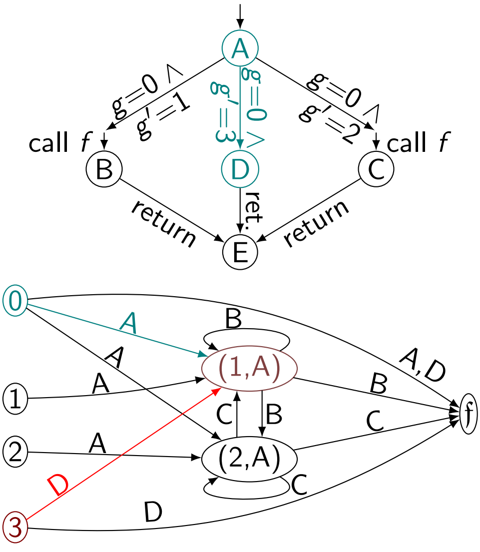
Generating automata for the left and right threads (17)



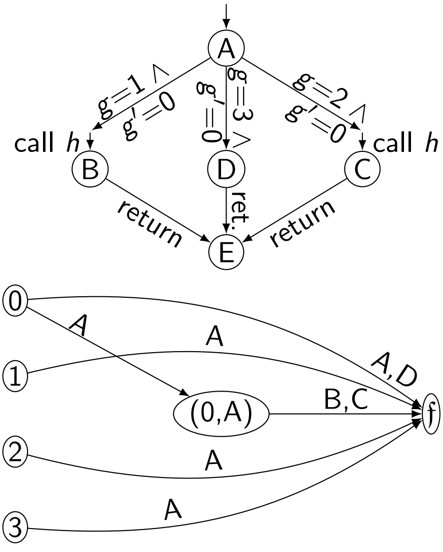
$G_0 = \{(0,1), (0,2), (0,3)\}$ $G_1 = \{(1,0), (2,0), (3,0)\}$

Generating automata for the left and right threads (18)

Procedure f: initially $g = 0$ Procedure h:

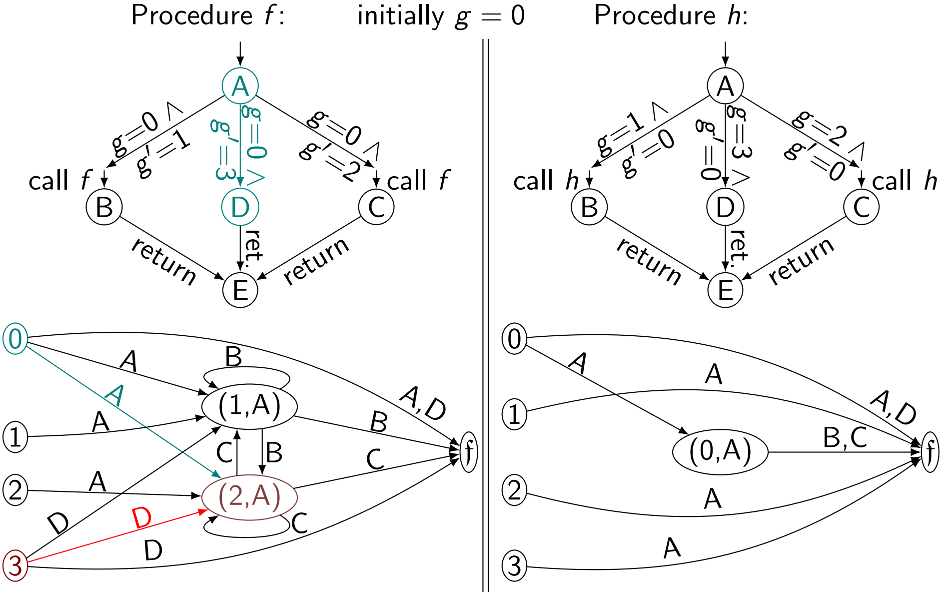


$$G_0 = \{(0,1),(0,2),(0,3)\}$$



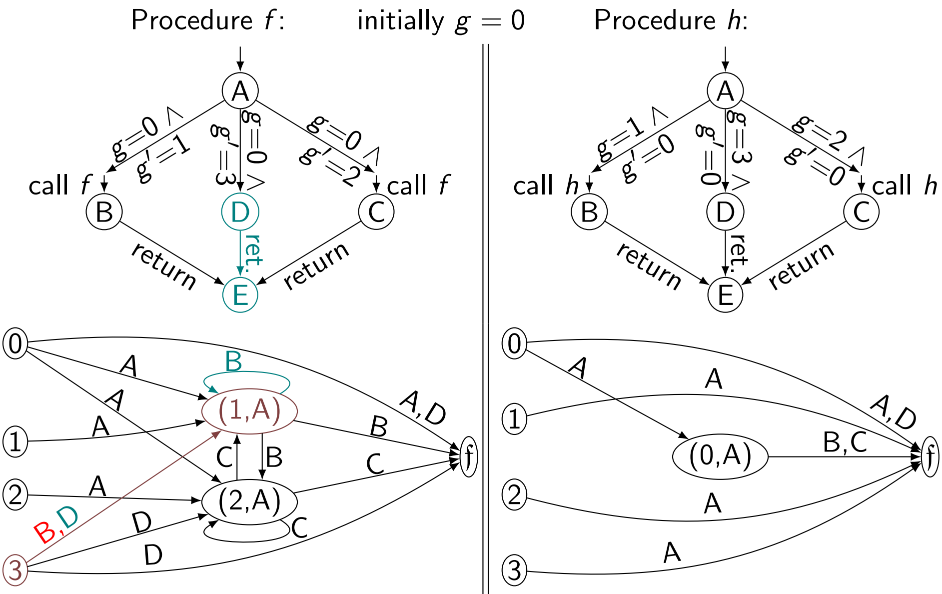
$$G_1 = \{(1,0),(2,0),(3,0)\}$$

Generating automata for the left and right threads (19)



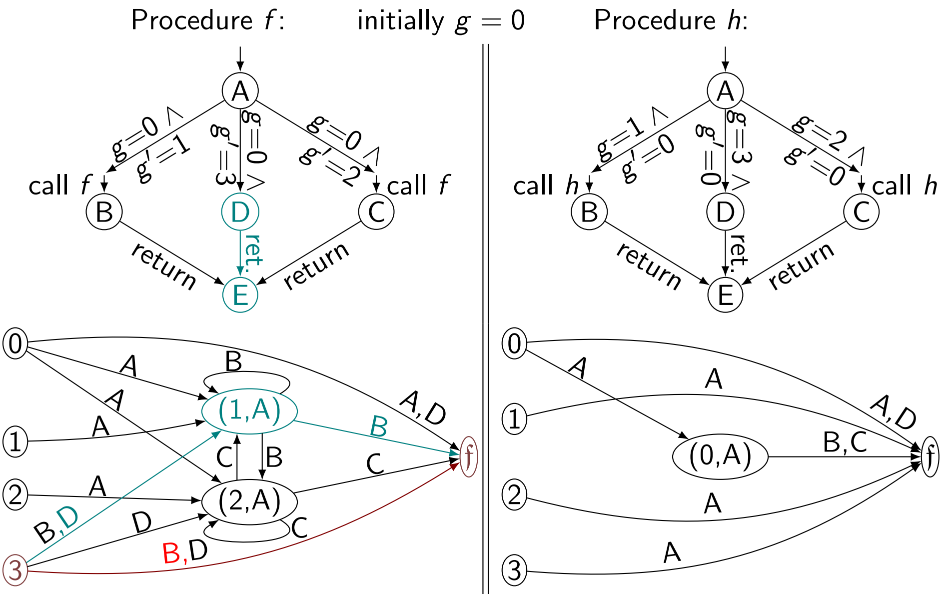
$G_0 = \{(0,1),(0,2),(0,3)\}$ $\qquad$ $G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (20)



$G_0 = \{(0,1),(0,2),(0,3)\}$ $G_1 = \{(1,0),(2,0),(3,0)\}$

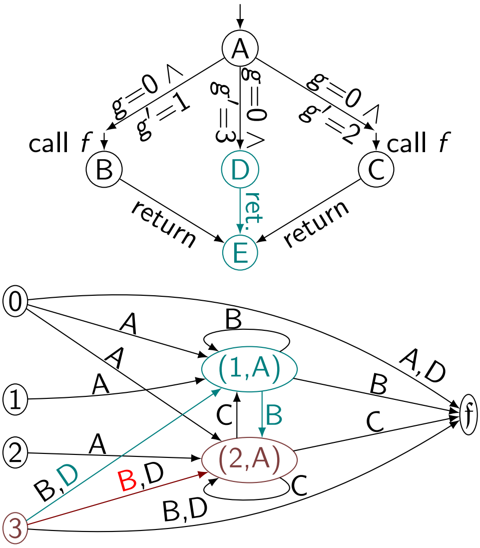
Generating automata for the left and right threads (21)



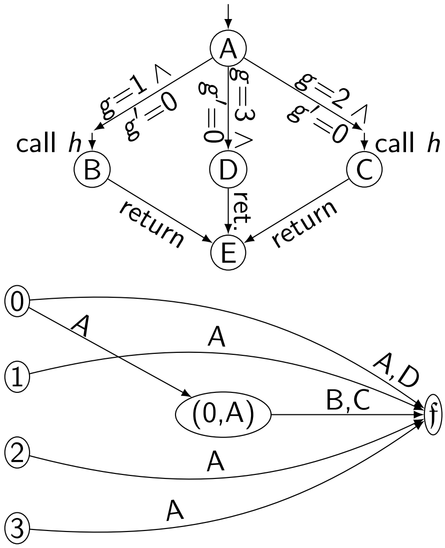
$G_0 = \{(0,1),(0,2),(0,3)\}$ $\qquad$ $G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (22)

Procedure f: initially $g = 0$ Procedure h:



$G_0 = \{(0,1),(0,2),(0,3)\}$



$G_1 = \{(1,0),(2,0),(3,0)\}$

Generating automata for the left and right threads (. . .)

. . .

TMR inference system

Couple post* with thread-modular verification.

Let $\mathfrak{f}$ be a fresh symbol. Let $V = \mathrm{Glob} \cup \mathrm{Glob} \times \mathrm{Loc}$.

$$(\textsc{tmr init})\ \frac{(g, \ell) \in \mathrm{init}}{g \xrightarrow[t]{\ell_t} \mathfrak{f}}\ t \in n$$

$$(\textsc{tmr step})\ \frac{((g, a), (g', b)) \in \sqcup\!\!\!\!\!-_t \qquad g \xrightarrow[t]{a} v}{g' \xrightarrow[t]{b} v \qquad (g, g') \in G_t}\ t \in n$$

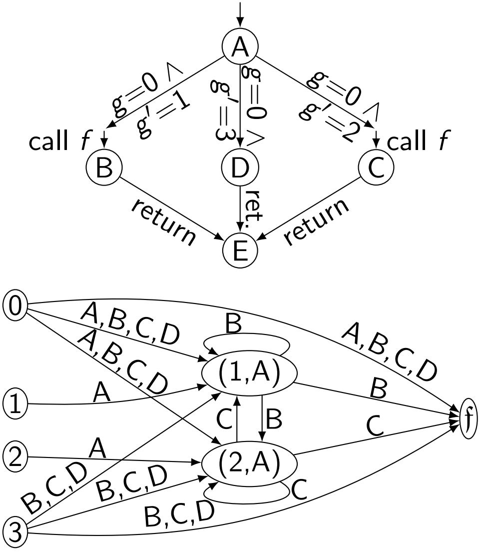
$$(\textsc{tmr push})\ \frac{g \xrightarrow[t]{a} v \qquad ((g, a), (g', b, c)) \in \sqcup\!\!\!\!\!\downarrow_t}{g' \xrightarrow[t]{b} (g', b) \xrightarrow[t]{c} v \qquad (g, g') \in G_t}$$

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

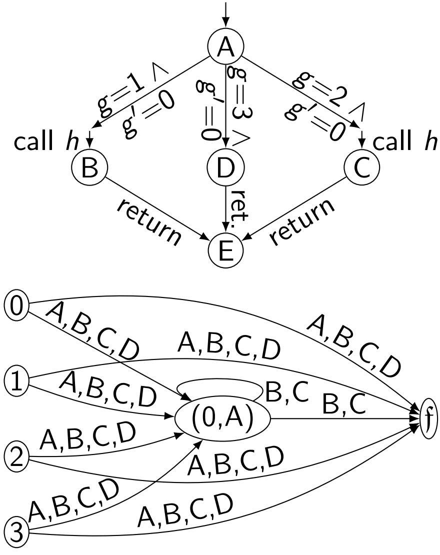
$$(\textsc{tmr env})\ \frac{(g, g') \in G_t \qquad g \xrightarrow[s]{a} v}{g' \xrightarrow[s]{a} v}\ t \neq s \text{ are in } n$$

Generated automata for the left and right threads

Procedure f: initially $g = 0$ Procedure h:

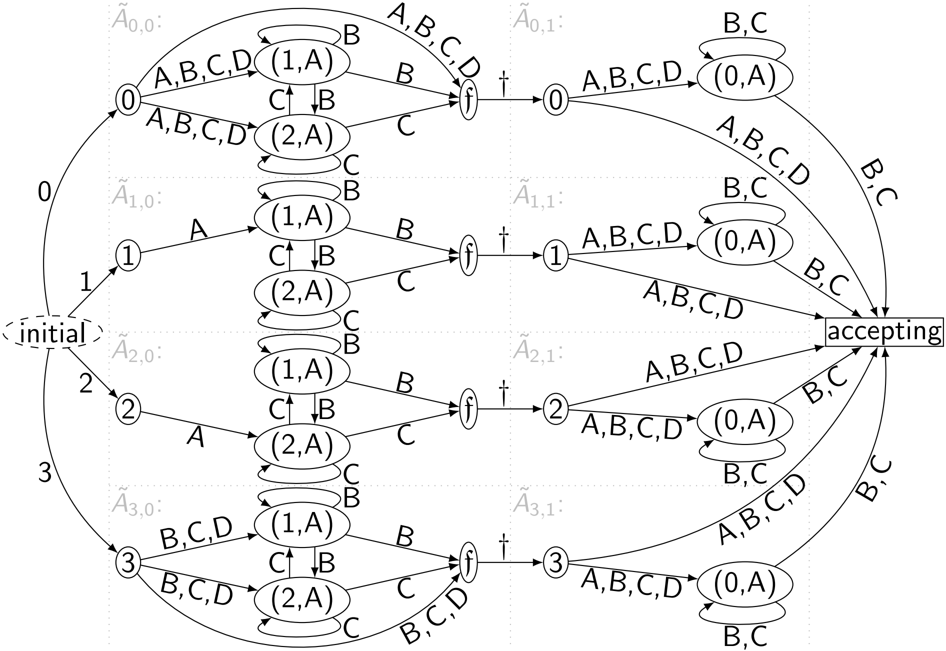


$G_0 = \{(0,1),(0,2),(0,3)\}$



$G_1 = \{(1,0),(2,0),(3,0)\}$

NFAs for the threads → NFAs for the program



Summary

The strongest multithreaded-Cartesian inductive invariant is a regular language.

Multithreaded-Cartesian abstract interpretation can be implemented in $\mathcal{O}(n \log n)$ time on a RAM under log-cost measure (and in polynomial time in other quantities).

(Proof: see *Multithreaded-Cartesian Abstract Interpretation of Multithreaded Recursive Programs Is Polynomial*, Alexander Malkis; RP 2015, Warsaw, Springer, `https://www.sec.in.tum.de/~malkis/Malkis-MultCartAbstIntOfMultRecProgIsPoly_techrep.pdf`.)

Precise running time on RAM under log-cost

- Representation of all data: lists in tables.
- Let $L(x)$ be the length of the binary representation of $x \in \mathbb{N}_0$.
- Time of a single access $= L(\text{address}) + L(\text{data}) + 1$.
- Running time of TMR: $\mathcal{O}\big(n(|\text{init}| + |\text{Glob}|^4|\text{Frame}|^5)(L(|\text{init}|) + L(n) + L(|\text{Glob}|) + L(|\text{Frame}|))\big)$.
- Constructing the full NFA after constructing the threads' NFAs is asymptotically negligible.
- In the input length size, it is $\mathcal{O}((\text{input length})^2 L(\text{input length}))$.

Questions?