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 \texttt{assert(\ldots)} statements.
- **Safety** property: “nothing bad happens”, a superclass of invariance properties.
- Checking a safety property can be reduced to checking an invariance property of \texttt{(program \parallel \text{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.


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: \((\text{Glob, Frame, init, } (\sqcup_t, \sqcup_t, \sqcup_t)_{t<n})\)

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

For each \(t < n\):

- \(\sqcup_t \subseteq (\text{Glob} \times \text{Frame}) \times (\text{Glob} \times \text{Frame} \times \text{Frame})\),
- \(\sqcup_t \subseteq (\text{Glob} \times \text{Frame}) \times (\text{Glob} \times \text{Frame})\),
- \(\sqcup_t \subseteq (\text{Glob} \times \text{Frame} \times \text{Frame}) \times (\text{Glob} \times \text{Frame})\).
As the operational semantics of the whole program we choose the so-called interleaving semantics. It is given by

the concrete domain $D = \mathcal{P}(\text{State}),$

and the successor map

$\text{post} : D \to D,$

$$ Q \mapsto \{(g', \ell') \mid \exists \ t < n, (g, \ell) \in Q : (g, \ell_t) \leadsto_t (g', \ell'_t) $$

$$ \land \forall s < n : s \neq t \Rightarrow \ell_s = \ell'_s \}.$$
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 Bölükbaşı, 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:

\[ S \text{ invariant} \iff \text{lfp}(\lambda Q \in D. \text{init} \cup \text{post}(Q)) \subseteq S. \]

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

\[ S \text{ inductive} \iff \text{init} \subseteq S \land \text{post}(S) \subseteq S. \]

To prove an invariance property \( P \subseteq \text{State} \), 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 |Glob| - 1\}, finite n. A set \( S \subseteq \text{State} \) is in multithreaded-Cartesian form iff there are \( L_{g,t} \subseteq \text{Loc} \) \((g \in \text{Glob}, t < n)\) s. t.

\[
S = \{0\} \times L_{0,0} \times \ldots \times L_{0,n-1} \\
\cup \{1\} \times L_{1,0} \times \ldots \times L_{1,n-1} \\
\vdots \\
\cup \{|Glob| - 1\} \times L_{|Glob|-1,0} \times \ldots \times L_{|Glob|-1,n-1}.
\]
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 $S \subseteq \text{State}$ is in multithreaded-Cartesian form iff
the intersection of each equivalence class with $S$ is a Cartesian
product.
Multithreaded-Cartesian overapproximation

\[ \rho_{mc} \left( \{0\} \times \bigcup \{1\} \times \bigcup \{2\} \right) \]

\[ = \{0\} \times \bigcup \{1\} \times \bigcup \{2\} . \]

For \( S \subseteq \text{State} \),

\[ \rho_{mc}(S) = \{(g, \ell) \mid \forall t < n \exists \tilde{\ell} \in \text{Loc}^n: \tilde{\ell}_t = \ell_t \wedge (g, \tilde{\ell}) \in S\} . \]

\( \rho_{mc} \) is an upper closure operator.
Multithreaded-Cartesian Galois-connection: abstraction

Concrete domain: \( D = \mathcal{P}(\text{State}) = \mathcal{P}(\text{Glob} \times \text{Loc}^n) \).
Abstract domain: \( D^\# = (\mathcal{P}(\text{Glob} \times \text{Loc}))^n \).

\[
\alpha_{mc} \left( \{0\} \times \bigcup \{1\} \times \bigcup \{2\} \right) = \left( \{0\} \times \bigcup \{1\} \times \bigcup \{2\} \right),
\]

Abstraction \( \alpha_{mc}(S) = \left( \{(g, \ell_t) \mid (g, \ell) \in S\} \right)_{t<n} \).
Multithreaded-Cartesian abstraction: concretization

\[ \gamma_{mc}(\{0\} \times \{1\} \cup \{2\}) = \{0\} \times \{1\} \cup \{2\} \]

Concretization \( \gamma_{mc}((A_t)_{t<n}) = \{(g, \ell) | \forall t<n: (g, \ell_t) \in A_t\} \).

\[ \rho_{mc} = \gamma_{mc} \circ \alpha_{mc} \]
Multithreaded-Cartesian abstract interpretation

\[ \text{lfp}(\lambda \ S \in D. \ \rho_{mc}(\text{init} \cup \text{post}(S))) \]

\[ \text{lfp}(\lambda \ A \in D^\#. \ \alpha_{mc}(\text{init} \cup \text{post}(\gamma_{mc}(A)))) \]
Intricate example

Initially $g = 0$

Procedure $f$:
- $g = 0 \land g' = 1$
  - Call $f$
  - Return
- $g = 0 \land g' = 2$
  - Call $f$
  - Return
- $g = 0 \land g' = 3$
  - Call $f$
  - Return

Procedure $h$:
- $g = 1 \land g' = 0$
  - Call $h$
  - Return
- $g = 3 \land g' = 0$
  - Call $h$
  - Return

Postcondition $g = 0 \lor g = 3$

The strongest inductive invariant is not context-free.
Strongest multithreaded-Cartesian inductive invariant for
the example

\[
S = \left( \begin{array}{c}
\{0\} \times (\{Ax, Dx \mid x \in \{B, C\}^*\} \cup \{B, C\}^+) \\
\cup \{1\} \times \{ABx \mid x \in \{B, C\}^*\} \\
\cup \{2\} \times \{ACx \mid x \in \{B, C\}^*\} \\
\cup \{3\} \times (\{Dx \mid x \in \{B, C\}^*\} \cup \{B, C\}^+) \\
\times (\{Ax, Dx \mid x \in \{B, C\}^*\} \cup \{B, C\}^+) \end{array} \right)
\]

Notice:

\[ (|w_1| = |w_2| = 1 \land (g, (w_1, w_2)) \in S) \Rightarrow (g = 0 \lor g = 3). \]

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!
Generating a regular representation of

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

- Construct $n$ NFAs simultaneously, one per thread.
- Each NFA describes a set of thread states ($\in \text{Glob} \times \text{Loc}$).
- Sequentially chain the NFAs.
Generating automata for the left and right threads (1)

Procedure $f$: initially $g = 0$

Procedure $h$: 

```plaintext
A

\[ g = 0 \land g' = 1 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 2 \]
\[ \Rightarrow 3 \]
\[ \text{call } f \]
\[ \text{return} \]
```

```plaintext
B

\[ g = 0 \land g' = 0 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 3 \]
\[ \Rightarrow 2 \]
\[ \Rightarrow 0 \]
\[ \text{call } h \]
\[ \text{return} \]
```

```plaintext
C

\[ g = 1 \land g' = 0 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 3 \]
\[ \Rightarrow 0 \]
\[ \text{call } h \]
\[ \text{return} \]
```

```plaintext
D

\[ g = 0 \land g' = 1 \]
\[ \Rightarrow 3 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 0 \]
\[ \text{call } f \]
\[ \text{return} \]
```

```plaintext
E

\[ g = 0 \land g' = 2 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 0 \]
\[ \Rightarrow 0 \]
\[ \text{call } f \]
\[ \text{return} \]
```

\[ A \]

\[ f \]
Generating automata for the left and right threads (2)

Procedure $f$: initially $g = 0$

Procedure $h$: $g = 0 \land g' = 2$

$G_0 = \{(0,1)\}$
Generating automata for the left and right threads (3)

Procedure $f$: initially $g = 0$

Procedure $h$: 

$G_0 = \{(0, 1), (0, 2)\}$
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 $h$: $g = 0 \land g' = 1$

$g = 0 \land g' = 2$

$g = 0 \land g' = 3$

$\text{call } f$

return

$\text{call } f$

return

$\text{call } h$

return

$\text{call } h$

return

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

$G_0 = \{(0,1), (0,2), (0,3)\}$
Generating automata for the left and right threads (8)

Procedure $f$:  initially $g = 0$

\[

g = 0 \land g' = 1 \\
g = 0 \land g' = 2 \\
g = 0 \\
g = 1 \land g' = 0 \\
g = 1 \land g' = 3 \\
g = 2 \land g' = 0 \\
g = 3 \land g' = 0
\]

\text{call } f \quad \text{return} \quad \text{call } f \quad \text{return}

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

Procedure $h$:

\[

g = 1 \land g' = 0 \\
g = 1 \land g' = 0 \\
g = 3 \land g' = 0 \\
g = 2 \land g' = 0
\]

\text{call } h \quad \text{return} \quad \text{call } h \quad \text{return}

$G_1 = \{(1, 0)\}$
Generating automata for the left and right threads (9)

Procedure $f$: initially $g = 0$

Procedure $h$: $\begin{align*}
g &= 0 \land g' = 1 \\
g &= 0 \land g' = 2 \\
g &= 0 \land g' = 3 \\
call f \\
ret. \\
return \\
call h \\
ret. \\
return
\end{align*}$

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

$G_1 = \{(1, 0), (2, 0)\}$
Generating automata for the left and right threads (10)

Procedure $f$: initially $g = 0$

Procedure $h$: 

\[
\begin{align*}
A & \quad D & \quad B & \quad C \\
\begin{array}{c}
g = 0 \\
g' = 1 \\
g = 0 \\
g' = 2
\end{array} & \quad \begin{array}{c}
g = 0 \\
g' = 0 \\
g = 3 \\
g' = 0
\end{array} & \quad \begin{array}{c}
g = 0 \\
g' = 0 \\
g = 2 \\
g' = 0
\end{array}
\end{align*}
\]

\[\begin{array}{c}
call f \\
return \]

\[\begin{array}{c}
call f \\
return \]

\[\begin{array}{c}
call h \\
return \]

\[\begin{array}{c}
call h \\
return \]

\[G_0 = \{(0, 1), (0, 2), (0, 3)\} \quad G_1 = \{(1, 0), (2, 0), (3, 0)\}\]
Generating automata for the left and right threads (11)

Procedure \( f \): initially \( g = 0 \)

\[
\begin{align*}
A & \quad \text{call } f \\
B & \quad g = 0 \wedge g' = 1 \\
D & \quad \text{return} \\
C & \quad \text{return} \\
E & \quad \text{return} \\
\end{align*}
\]

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

Procedure \( h \):

\[
\begin{align*}
A & \quad \text{call } h \\
B & \quad g = 1 \wedge g' = 0 \\
D & \quad \text{return} \\
C & \quad \text{return} \\
E & \quad \text{return} \\
\end{align*}
\]

\[
G_1 = \{(1, 0), (2, 0), (3, 0)\}
\]
Generating automata for the left and right threads (12)

Procedure $f$: initially $g = 0$

Procedure $h$: 

\[
\begin{align*}
&\text{call } f \\
&\text{return} \\
\end{align*}
\]

\[
\begin{align*}
&\text{call } f \\
&\text{return} \\
\end{align*}
\]

\[
\begin{align*}
&\text{call } h \\
&\text{return} \\
\end{align*}
\]

\[
\begin{align*}
&\text{call } h \\
&\text{return} \\
\end{align*}
\]

\[
\begin{align*}
&g = 0 \land g' = 0 \\
&g = 1 \land g' = 0 \\
&g = 0 \land g' = 2 \\
&g = 0 \land g' = 1 \\
&g = 2 \land g' = 0 \\
&g = 0 \land g' = 3 \\
\end{align*}
\]

\[
\begin{align*}
&g = 0 \land g' = 0 \\
&g = 0 \land g' = 2 \\
&g = 3 \land g' = 0 \\
&g = 0 \land g' = 3 \\
\end{align*}
\]

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

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

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

Procedure $h$:

$G_1 = \{(1, 0), (2, 0), (3, 0)\}$
Generating automata for the left and right threads (15)

Procedure $f$:
- Initially $g = 0$
- Procedure $h$:
- $g = 0 \land g' = 1$
- $g = 0 \land g' = 2$
- $g = 0 \land g' = 3$
- Call $f$
- Call $h$
- Return

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

$G_1 = \{(1, 0), (2, 0), (3, 0)\}$
Generating automata for the left and right threads (16)

Procedure $f$: initially $g = 0$

$\begin{align*}
&g = 0 \land g' = 1 \\
&g = 0 \land g' = 2 \\
&g = 0 \land g' = 3 \\
&\text{call } f
\end{align*}$

Procedure $h$:

$\begin{align*}
&g = 1 \land g' = 0 \\
&g = 3 \land g' = 0 \\
&\text{call } h
\end{align*}$

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

$G_1 = \{(1, 0), (2, 0), (3, 0)\}$
Generating automata for the left and right threads (17)

Procedure $f$: initially $g = 0$

Procedure $h$: $g = 0$ and $g' = 1$

$g = 0$ and $g' = 2$

$g = 0$ and $g' = 3$

call $f$

return

call $f$

return

call $h$

return

call $h$

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

$G_1 = \{(1, 0), (2, 0), (3, 0)\}$
Generating automata for the left and right threads (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)

Procedure $f$: initially $g = 0$

Procedure $h$: $g = 0 \land g' = 1$ 

$g = 0 \land g' = 2$ 

$g = 0 \land g' = 3$ 

$\text{call } f$ 

$\text{return}$

$g = 1 \land g' = 0$ 

$g = 1 \land g' = 3$ 

$g = 2 \land g' = 0$ 

$\text{call } h$ 

$\text{return}$

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

$G_1 = \{(1, 0), (2, 0), (3, 0)\}$
Generating automata for the left and right threads (20)

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 (21)

Procedure $f$: initially $g = 0$

Procedure $h$:

\begin{align*}
G_0 &= \{(0, 1), (0, 2), (0, 3)\} \\
G_1 &= \{(1, 0), (2, 0), (3, 0)\}
\end{align*}
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 (\ldots)
TMR inference system

Couple post* with thread-modular verification.
Let $f$ be a fresh symbol. Let $V = \text{Glob} \cup \text{Glob} \times \text{Loc}$.

\[
\begin{align*}
\text{(TMR INIT)} & \quad \frac{(g, \ell) \in \text{init}}{t \in n} \quad g \xrightarrow{\ell_t} f \\
\text{(TMR STEP)} & \quad \frac{((g, a), (g', b)) \in \sqcup_t \quad g \xrightarrow{a} v}{t \in n} \quad g' \xrightarrow{b} v \quad (g', g) \in G_t \\
\text{(TMR PUSH)} & \quad \frac{g \xrightarrow{a} v \quad ((g, a), (g', b, c)) \in \sqcup_t \quad g' \xrightarrow{b} (g', b) \xrightarrow{c} v \quad (g, g') \in G_t}{g' \xrightarrow{b} v} \\
\text{(TMR POP)} & \quad \frac{g \xrightarrow{a} v \quad ((g, a, b), (g', c)) \in \sqcup_t \quad g' \xrightarrow{c} v \quad (g', g) \in G_t}{g' \xrightarrow{c} v} \\
\text{(TMR ENV)} & \quad \frac{(g, g') \in G_t \quad g \xrightarrow{a} v \quad t \neq s \text{ are in } n}{g' \xrightarrow{a} v} 
\end{align*}
\]
Generated automata for the left and right threads

Procedure $f$: initially $g = 0$

- $g = 0 \land g' = 1$:
  - call $f$
  - return

- $g = 0 \land g' = 2$:
  - $g = 3$:
    - call $f$
    - return
  - $g = 0$:
    - $g = 2$:
      - return
    - $g = 0$:
      - $g = 0$:
        - return

- $g = 1$:
  - $g = 0$:
    - call $h$
    - return

- $g = 2$:
  - $g = 0$:
    - call $h$
    - return

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

Procedure $h$:

- $g = 1 \land g' = 0$:
  - call $h$
  - return

- $g = 2 \land g' = 0$:
  - call $h$
  - return

$G_1 = \{(1,0), (2,0), (3,0)\}$
NFAs for the threads → NFAs for the program

\[ \tilde{A}_{0,0}: \]
- Initial state: 0
- Transitions:
  - 0 \(\xrightarrow{A,B,C,D} (1,A)\)
  - 0 \(\xrightarrow{B} (2,A)\)
  - 0 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{0,1}: \]
- Initial state: 1
- Transitions:
  - 1 \(\xrightarrow{B,C} (0,A)\)
  - 1 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{1,0}: \]
- Initial state: 2
- Transitions:
  - 2 \(\xrightarrow{B} (1,A)\)
  - 2 \(\xrightarrow{C} (1,A)\)
  - 2 \(\xrightarrow{B} (2,A)\)
  - 2 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{1,1}: \]
- Initial state: 3
- Transitions:
  - 3 \(\xrightarrow{B} (0,A)\)
  - 3 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{2,0}: \]
- Initial state: 4
- Transitions:
  - 4 \(\xrightarrow{B,C} (1,A)\)
  - 4 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{2,1}: \]
- Initial state: 5
- Transitions:
  - 5 \(\xrightarrow{B,C} (0,A)\)
  - 5 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{3,0}: \]
- Initial state: 6
- Transitions:
  - 6 \(\xrightarrow{B,C} (1,A)\)
  - 6 \(\xrightarrow{A,B,C,D} f\) (accepting)

\[ \tilde{A}_{3,1}: \]
- Initial state: 7
- Transitions:
  - 7 \(\xrightarrow{B,C} (0,A)\)
  - 7 \(\xrightarrow{A,B,C,D} f\) (accepting)
The strongest multithreaded-Cartesian inductive invariant is a regular language.

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

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:
  $O(n(|\text{init}| + |\text{Glob}|^4|\text{Frame}|^5)(L(|\text{init}|) + L(n) + L(|\text{Glob}|) + L(|\text{Frame}|)))$.

- Constructing the full NFA after constructing the threads’ NFAs is asymptotically negligible.

- In the input length size, it is $O((\text{input length})^2L(\text{input length}))$. 
Questions?