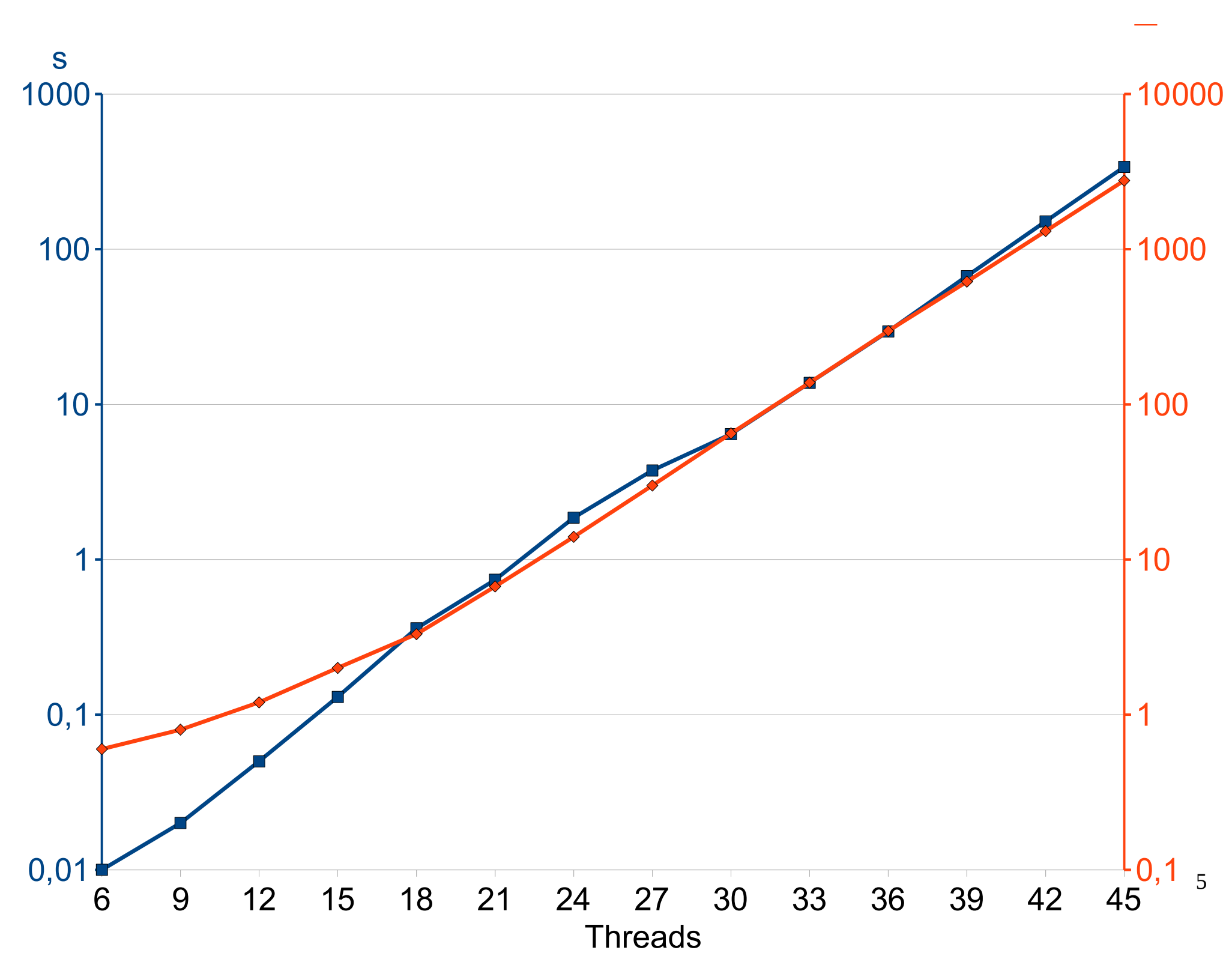
Modular verification of multithreaded shared-memory programs

Alexander Malkis

- Owicki-Gries
- Rely-guarantee reasoning
- Thread-modular model checking
- Cartesian abstraction
- Thread simplification
- Refinement

```
int x=0;

proctype Inc()   { do :: atomic{ x<200 -> x=x+1 } od }

proctype Dec()   { do :: atomic{ x>0   -> x=x-1 } od }

proctype Reset() { do :: x==200 -> x=0   od }

proctype Check() { assert 0<=x && x<=200    }

init { byte k=0; run Check();
  do :: k<3 -> run Inc(); run Dec(); run Reset(); k++ od }


State-vector 60 byte, depth reached 3435, errors: 0
...
pan: elapsed time 0 seconds
```


s
1000
100
10
1
0,1
0,01
10000
1000
100
10
1
0,1
6
9
12
15
18
21
24
27
30
33
36
39
42
45
Threads

Owicki-Gries succeeds

$$\{x=0\}$$
$$x:=x+1 \parallel x:=x+2$$
$$\{x=3\}$$

$$\{x=0\}$$
$$\begin{array}{c} \{\ldots\} \\ x:=x+1 \\ \{\ldots\} \end{array} \parallel \begin{array}{c} \{\ldots\} \\ x:=x+2 \\ \{\ldots\} \end{array}$$
$$\{x=3\}$$

Owicki-Gries succeeds

$$\{x=0\}$$
$$x:=x+1 \;\|\; x:=x+2$$
$$\{x=3\}$$

$$\{x=0\}$$
$$\begin{array}{c|c} \{x=0 \vee x=2\} & \{x=0 \vee x=1\} \\ x:=x+1 & x:=x+2 \\ \{x=1 \vee x=3\} & \{x=2 \vee x=3\} \end{array}$$
$$\{x=3\}$$

Owicki-Gries fails

$$\{x=0\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{x=2\}$$

$$\{x=0\}$$
$$\{\ x=0 \vee \ldots\} \;\|\; \{\ x=0 \vee \ldots\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{\ x=1 \vee \ldots\} \;\|\; \{\ x=1 \vee \ldots\}$$
$$\{\ x=1 \vee \ldots\}$$

Owicki-Gries fails

$$\{x=0\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{x=2\}$$

$$\{x=0\}$$
$$\{\ x=0 \vee x=1 \vee \ldots\} \;\|\; \{\ x=0 \vee x=1 \vee \ldots\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{\ x=1 \vee x=2 \vee \ldots\} \;\|\; \{\ x=1 \vee x=2 \vee \ldots\}$$
$$\{\ x=1 \vee x=2 \vee \ldots\}$$

Owicki-Gries fails

$$\{x=0\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{x=2\}$$

$$\{x=0\}$$
$$\{\ x=0 \vee x=1 \vee x=2 \vee \ldots\} \;\|\; \{\ x=0 \vee x=1 \vee x=2 \vee \ldots\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{\ x=1 \vee x=2 \vee x=3 \vee \ldots\} \;\|\; \{\ x=1 \vee x=2 \vee x=3 \vee \ldots\}$$
$$\{\ x=1 \vee x=2 \vee x=3 \vee \ldots\}$$

Owicki-Gries: auxiliary variables

$$\{x=0\}$$
$$x:=x+1 \parallel x:=x+1$$
$$\{x=2\}$$

$$\{x=0\}$$
$$a:=0$$
$$\{x=0 \wedge a=0\}$$
$$(x,a):=(x+1,1) \parallel (x,a):=(x+1,2)$$
$$\{?\}$$

Owicki-Gries: auxiliary variables

$$\{x=0\}$$
$$x:=x+1 \;\|\; x:=x+1$$
$$\{x=2\}$$

$$\{x=0\}$$
$$a:=0$$
$$\{x=0 \wedge a=0\}$$
$$\begin{array}{l|l} \{(x=0 \wedge a=0) \vee (x=1 \wedge a=2)\} & \{(x=0 \wedge a=0) \vee (x=1 \wedge a=1)\} \\ (x,a):=(x+1,1) & (x,a):=(x+1,2) \\ \{(x=1 \wedge a=1) \vee (x=2 \wedge a=1) \vee & \{(x=1 \wedge a=2) \vee (x=2 \wedge a=2) \vee \\ \vee (x=2 \wedge a=2)\} & \vee (x=2 \wedge a=1)\} \end{array}$$
$$\{x=2 \wedge (a=1 \vee a=2)\}$$
$$\{x=2\}$$

Auxiliary assignments must terminate

```
{true}
aux:=1/0;
skip;
{false}
```

```
{true}
aux:=array[index_out_of_bounds];
skip;
{false}
```

```
{true}
skip;
{false}
```

Owicki-Gries: completeness

State = Mem × PC1 × ... × PCn

NewState = NewMem × PC1 × ... × PCn

{x=0}

p1:=A; p2:=C

{x=0 ∧ p1=A ∧ p2=C}

A: {p1=A∧ ((x=0∧p2=C)v(x=1∧p2=D))}
(x,p1):=(x+1,B)

B: {p1=B∧ ((x=1∧p2=C)v(x=2∧p2=D))}

||

C: {p2=C∧ ((x=0∧p1=A)v(x=1∧p1=B))}
(x,p2):=(x+1,D)

D: {p2=D∧ ((x=1∧p1=A)v(x=2∧p1=B))}

{x=2 ∧ p1=B ∧ p2=D}

{x=2}

Rely-Guarantee

$$T_1 \| guar_2 \leq guar_1$$
$$guar_1 \| T_2 \leq guar_2$$

$$\frac{\begin{array}{l} \forall i: \; post_i \quad post \\ \quad \wedge \quad (rely \vee \exists k \neq i: guar_k) \quad rely_i \\ \quad \wedge \quad guar_i \quad guar \\ \quad \wedge \quad T_i \;\; \underline{sat} \;\; (pre, rely_i, guar_i, post_i) \end{array}}{T_1 \| \ldots \| T_n \;\; \underline{sat} \;\; (pre, rely, guar, post)}$$

Rely-Guarantee succeeds

"A: x:=x+1; B:" sat
(x=0, x'=x v (x=0∧x'=2) v (x=1∧x'=3),
x'=x v (x=0∧x'=1) v (x=2∧x'=3), x=1 v x=3)

"C: x:=x+2; D:" sat
(x=0, x'=x v (x=0∧x'=1) v (x=2∧x'=3),
x'=x v (x=0∧x'=2) v (x=1∧x'=3), x=2 v x=3)

"A: x:=x+1; B: || C: x:=x+2; D:" sat
(x=0, x'=x,
x'=x v x'=x+1 v x'=x+2, x=3)

Rely-guarantee fails

“A: x:=x+1; B:” sat
(x=0, x'=x,
        x'=x v (x=0∧x'=1), x=1)

“C: x:=x+1; D:” sat
(x=0, x'=x,
        x'=x v (x=0∧x'=1), x=1)

Rely-guarantee fails

“A: x:=x+1; B:” sat
(x=0, x'=x v (x=0∧x'=1),
        x'=x v (x=0∧x'=1) v (x=1∧x'=2), x=1 v x=2)

“C: x:=x+1; D:” sat
(x=0, x'=x v (x=0∧x'=1),
        x'=x v (x=0∧x'=1) v (x=1∧x'=2), x=1 v x=2)

Rely-guarantee fails

“A: x:=x+1; B:” sat
(x=0, x'=x v (x=0∧x'=1) v (x=1∧x'=2),
x'=x v (x=0∧x'=1) v (x=1∧x'=2) v (x=2∧x'=3),
x∈{1,2,3})

“C: x:=x+1; D:” sat
(x=0, x'=x v (x=0∧x'=1) v (x=1∧x'=2),
x'=x v (x=0∧x'=1) v (x=1∧x'=2) v (x=2∧x'=3),
x∈{1,2,3})

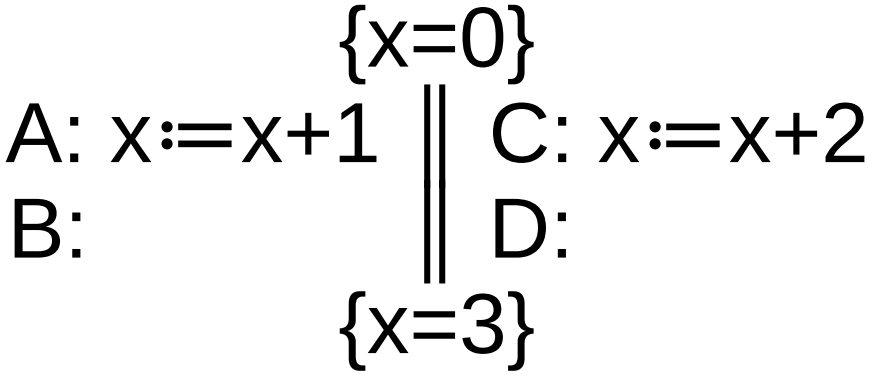
Rely-guarantee fails

"A: x:=x+1; B:" sat
(x=0, x'=x v 1≤x'=x+1,
x'=x v 1≤x'=x+1, x≥1)

"C: x:=x+1; D:" sat
(x=0, x'=x v 1≤x'=x+1,
x'=x v 1≤x'=x+1, x≥1)

"A: x:=x+1; B: ‖ C: x:=x+1; D:" sat
(x=0, x'=x v 1≤x'=x+1,
x'=x v 1≤x'=x+1, x≥1)

Thread-Modular model checking succeeds



R1={...}, R2={...}
G1={...}, G2={...}

Thread-Modular model checking succeeds

{x=0}

A: x:=x+1 ‖ C: x:=x+2

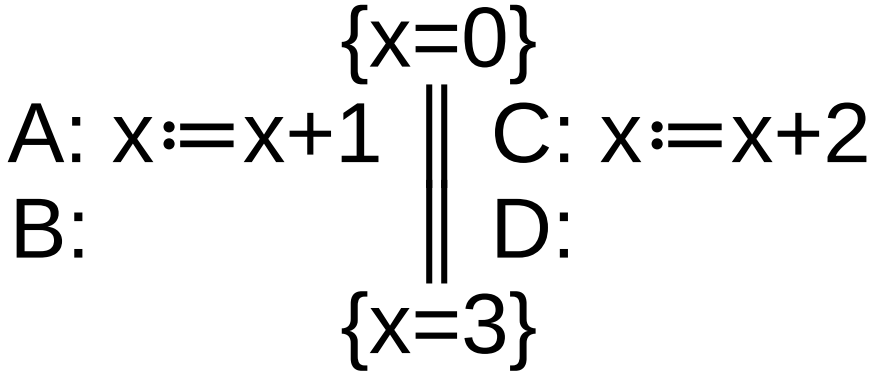
B: ‖ D:

{x=3}

R1={0A, ...}, R2={0C, ...}

G1={...}, G2={...}

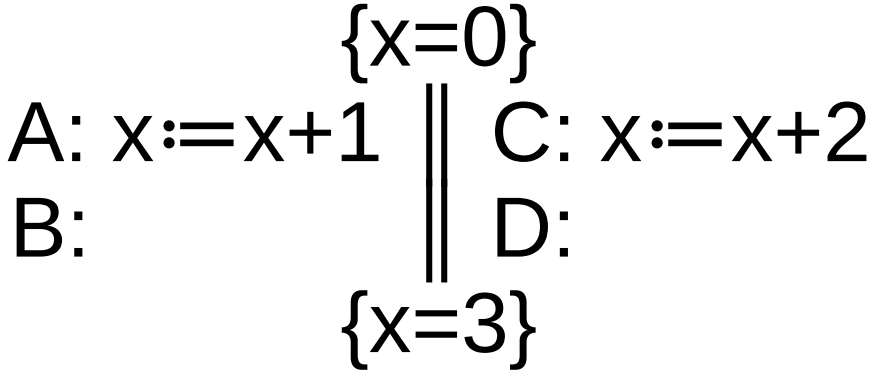
Thread-Modular model checking succeeds



R1={0A, 1B, ...}, R2={0C, 2D, ...}

G1={0▶1, ...}, G2={0▶2, ...}

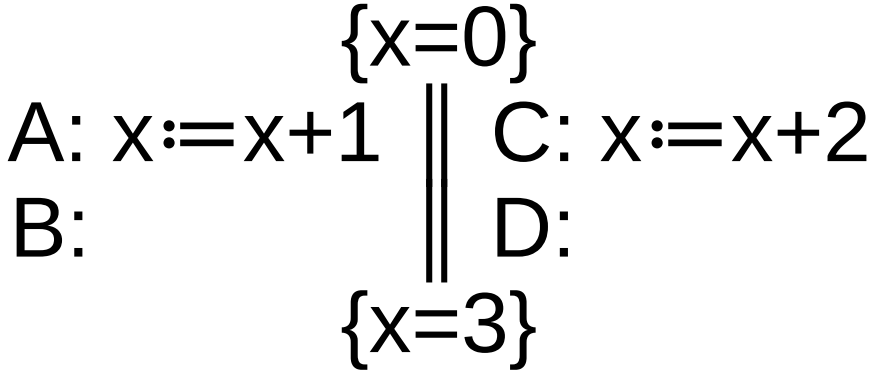
Thread-Modular model checking succeeds



R1={0A, 1B, 2A, ...}, R2={0C, 2D, 1C, ...}

G1={0▶1, ...}, G2={0▶2, ...}

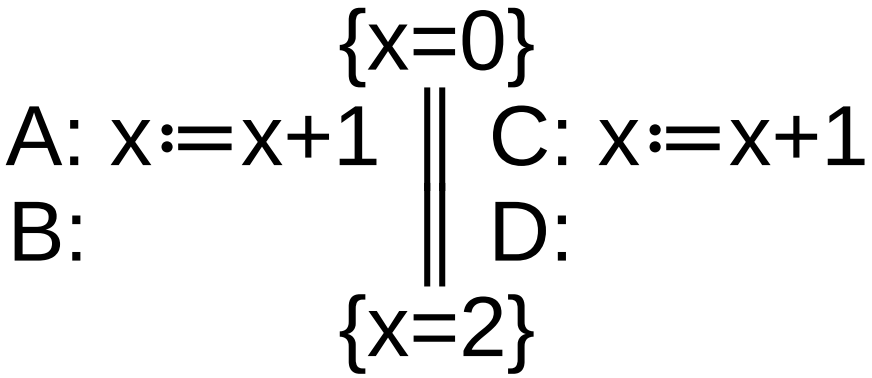
Thread-Modular model checking succeeds



R1={0A, 1B, 2A, 3B}, R2={0C, 2D, 1C, 3D}

G1={0▶1, 2▶3}, G2={0▶2, 1▶3}

Thread-Modular model checking fails

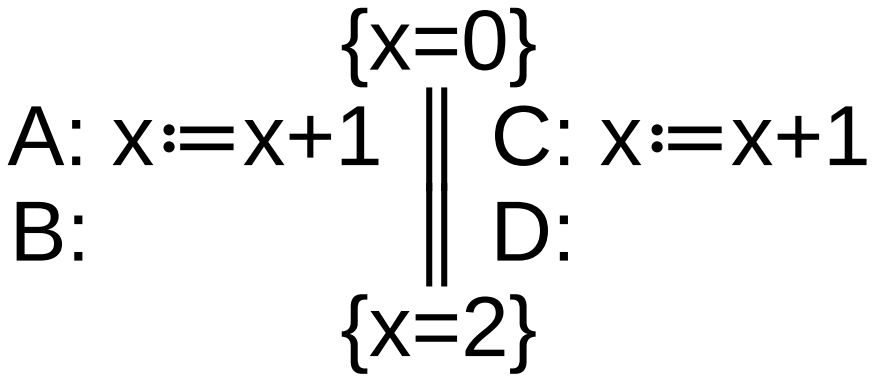


R1=
{0A, ...},

R2=
{0C,...}

G1={...}, G2={...}

Thread-Modular model checking fails



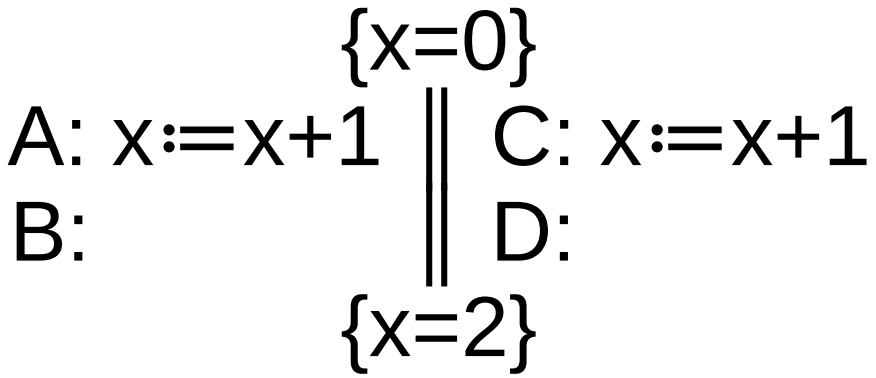
R1= {0A,1B, ...},

R2= {0C,1D, ...}

G1={0▶1, ...},

G2={0▶1, ...}

Thread-Modular model checking fails



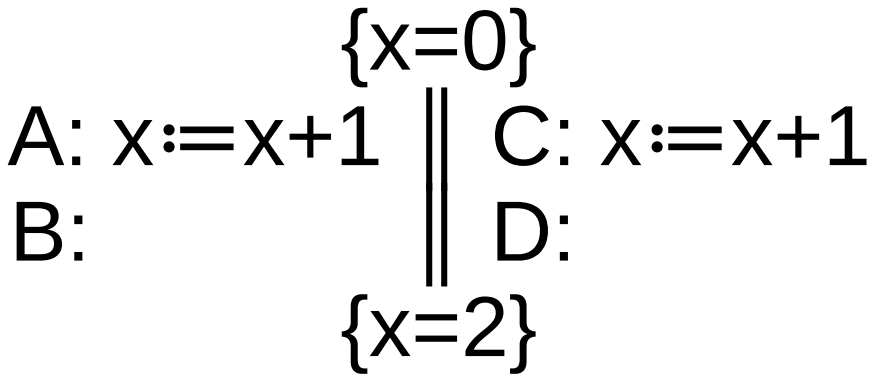
R1= {0A,1B,1A, ...},

R2= {0C,1D,1C, ...}

G1={0▶1, ...},

G2={0▶1, ...}

Thread-Modular model checking fails



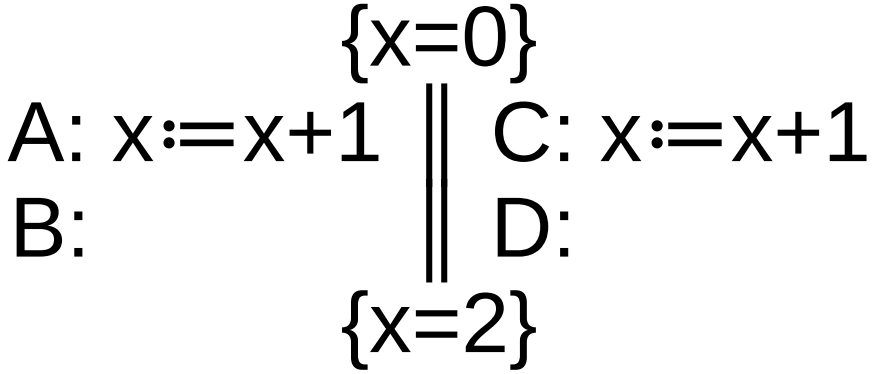
R1= {0A,1B,1A,2B, ...},

R2= {0C,1D,1C,2D, ...}

G1={0▶1, 1▶2, ...},

G2={0▶1, 1▶2, ...}

Thread-Modular model checking fails



R1= {0A,1B,1A,2B,2A,3B,...},

R2= {0C,1D,1C,2D,2C,3D,...}

G1={0▶1, 1▶2, 2▶3,...},

G2={0▶1, 1▶2, 2▶3,...}

Multithreaded Cartesian abstraction succeeds

{x=0}

A: x:=x+1 ‖ C: x:=x+2

B: ‖ D:

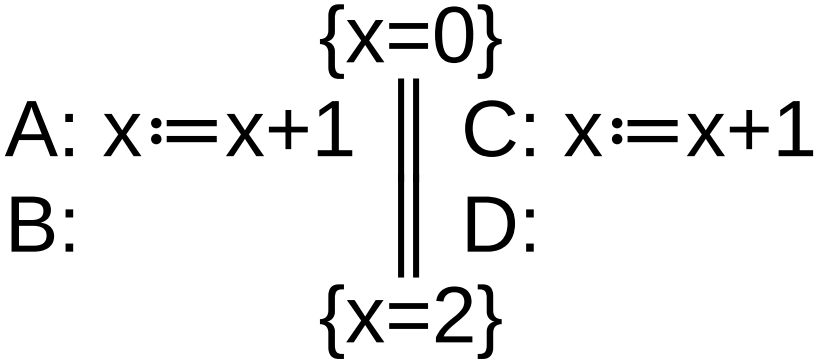
{x=3}

{0}x{A}x{C}

{0}x{A}x{C} U {1}x{B}x{C} U {2}x{A}x{D}

{0}x{A}x{C} U {1}x{B}x{C} U {2}x{A}x{D} U {3}x{B}x{D}

Multithreaded Cartesian abstraction fails



$$\{0\} \times \{A\} \times \{C\}$$

$$\{0\} \times \{A\} \times \{C\} \;\cup\; \{1\} \times \rho(\{B\} \times \{C\} \cup \{A\} \times \{D\})$$

Multithreaded Cartesian abstraction fails

{x=0}

A: x:=x+1 ‖ C: x:=x+1

B: ‖ D:

{x=2}

{0}x{A}x{C}

{0}x{A}x{C} U {1}x{A,B}x{C,D}

{0}x{A}x{C} U {1}x{A,B}x{C,D} U
{2} x ρ({B}x{C,D} U {A,B}x{D})

Multithreaded Cartesian abstraction fails

{x=0}

A: x:=x+1 ‖ C: x:=x+1

B: ‖ D:

{x=2}

{0}x{A}x{C}

{0}x{A}x{C} U {1}x{A,B}x{C,D}

{0}x{A}x{C} U {1}x{A,B}x{C,D} U {2}x{A,B}x{C,D}

{0}x{A}x{C} U {1}x{A,B}x{C,D} U {2}x{A,B}x{C,D} U {3} x ρ({B}x{C,D} U {A,B}x{D})

Multithreaded Cartesian abstraction fails

{x=0}

A: x:=x+1 ‖ C: x:=x+1

B: ‖ D:

{x=2}

{0}x{A}x{C}

{0}x{A}x{C} U {1}x{A,B}x{C,D}

{0}x{A}x{C} U {1}x{A,B}x{C,D} U {2}x{A,B}x{C,D}

{0}x{A}x{C} U {1}x{A,B}x{C,D} U {2}x{A,B}x{C,D} U {3}x{A,B}x{C,D}

Without auxiliary / rigid variables: common precision

∃ Owicki-Gries proof

⇔

∃ rely-guarantee proof

⇔

∃ thread-modular proof

⇔

∃ multithreaded Cartesian proof

Finite-state: polynomial time

Abstract Threads

$$T_1 \,||\, \ldots \,||\, T_n$$

$$\sqsupseteq$$

$$A_1 \,||\, \ldots \,||\, A_n$$

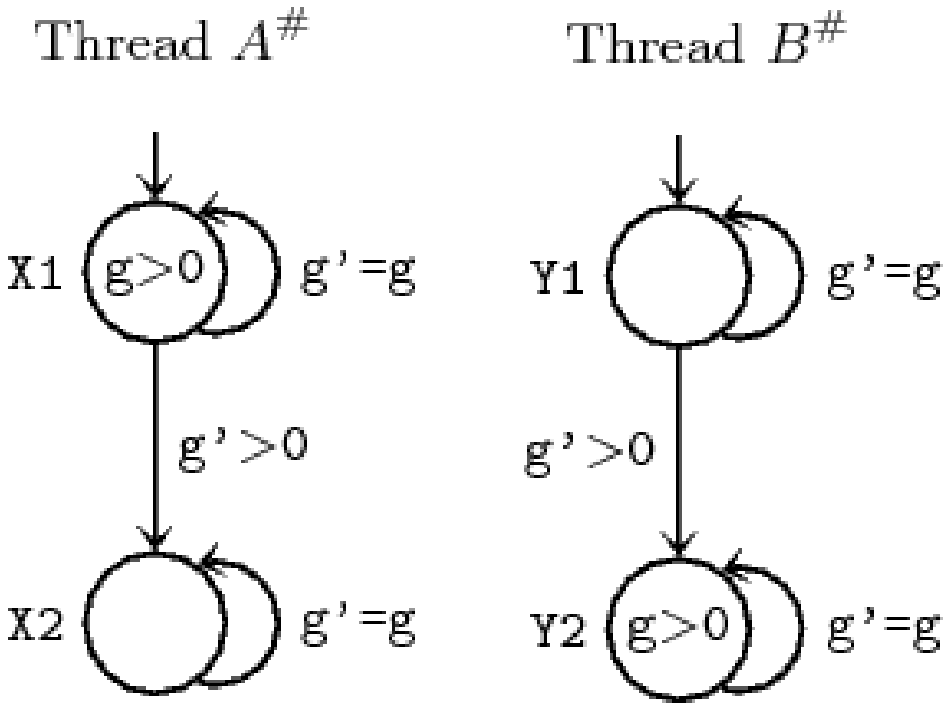
$$\begin{array}{ccc} T_1 & \ldots & T_n \\ \sqcap & \ldots & \sqcap \\ A_1 & \ldots & A_n \end{array}$$

Abstract Threads example

Thread A

```
int x;
x := 1;
while (*) {
    if (*) {
      x := g;
    } else {
      x := x+1;
    }
}
g := x;
```

g>0



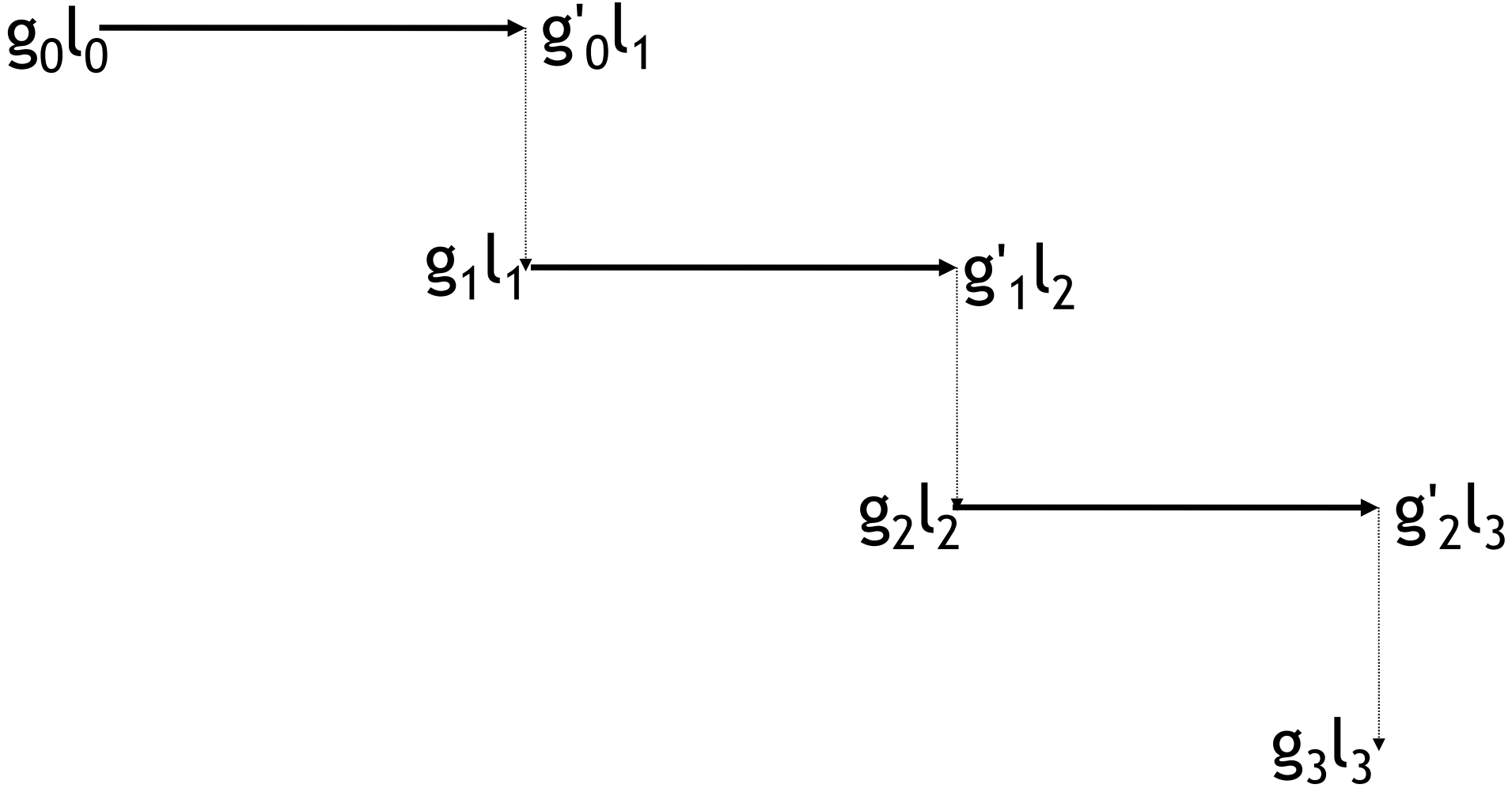
Thread B

```
int y;
y := 1;
while (*) {
    y := y+1;
}
g := y;
assert g>0;
```

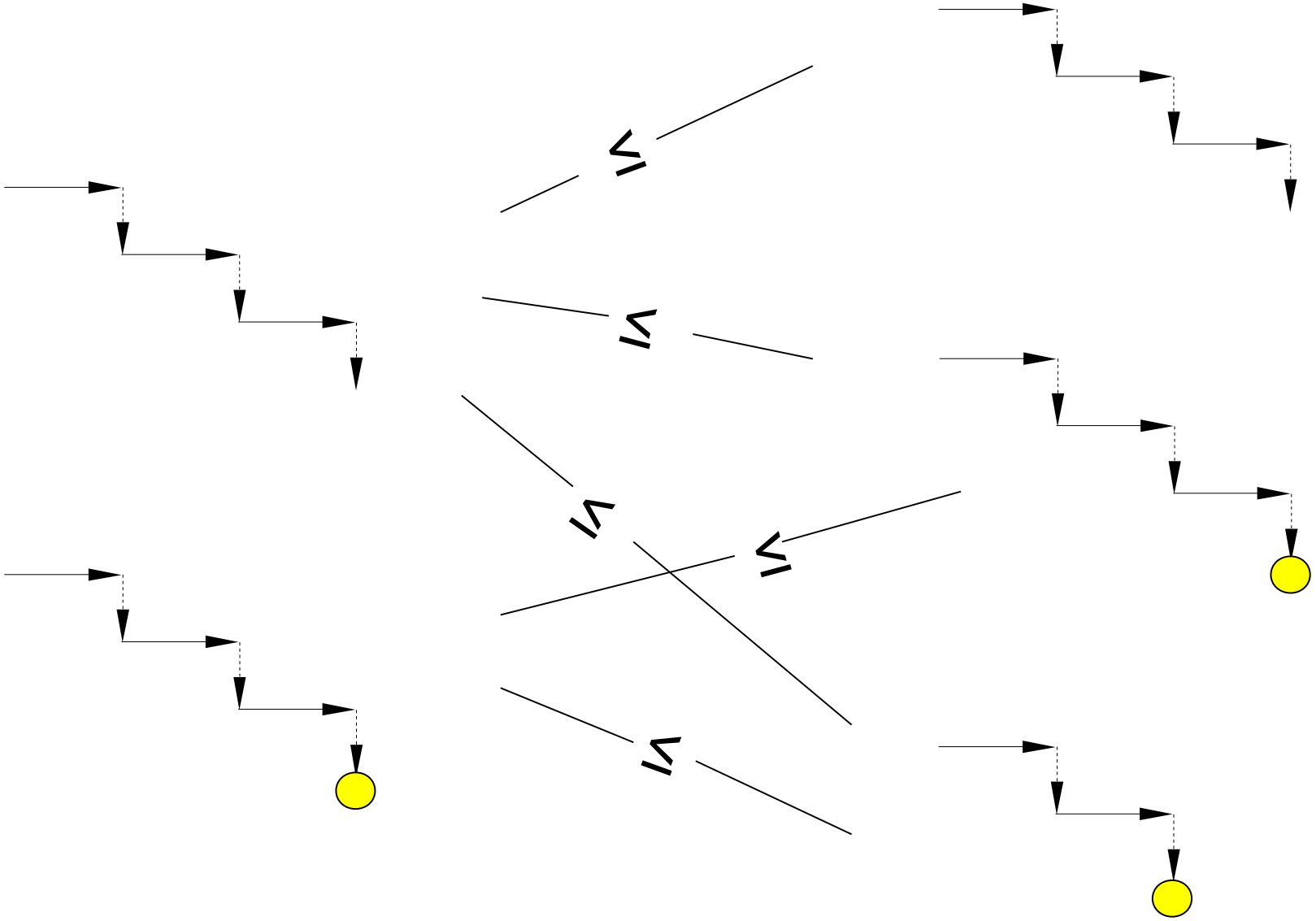
Infinite local state removed!

Abstract Threads: Phased Execution

$((g_0,g_1,g_2,g_3), (g'_0,g'_1,g'_2), (l_0,l_1,l_2,l_3))$



Abstract Threads: Abstraction



Abstract threads: Conformance Checker

- State of conformance checker: (l,F)
- *States* = *Local* × $\wp$(*Local*$^{\#}$)
- *Start* = *Init* × {*Init*$^{\#}$}
- *Error* = *Local* × {Ø} ∪ {(l,F) | $W(l) \not\in W^{\#}(F)$}

$$\frac{\begin{array}{c} l, l' \in Local \quad F, F' \subseteq Local^{\#} \\ \exists\, g, g' \in Global : g \notin W^{\#}(F) \text{ and } (g, l) \rightarrow (g', l') \text{ and} \\ F' = \{m' \mid \exists\, m \in F : (g, m) \rightarrow^{\#} (g', m')\} \end{array}}{(l, F) \rightsquigarrow (l', F')}$$

Abstract Threads: Conformance Checker Coding

- Replace non-assert *st* by

  assert $F \neq \emptyset$;
  assume $\forall\, l^{\#} \in F\colon (g,l^{\#}) \notin Wrong^{\#}$;
  st;
  $F := \{m' \mid \exists\, m \in F\colon (stored\_g,m) \rightarrow^{\#} (g,m')\}$;
  havoc *g*; $stored\_g := g$;

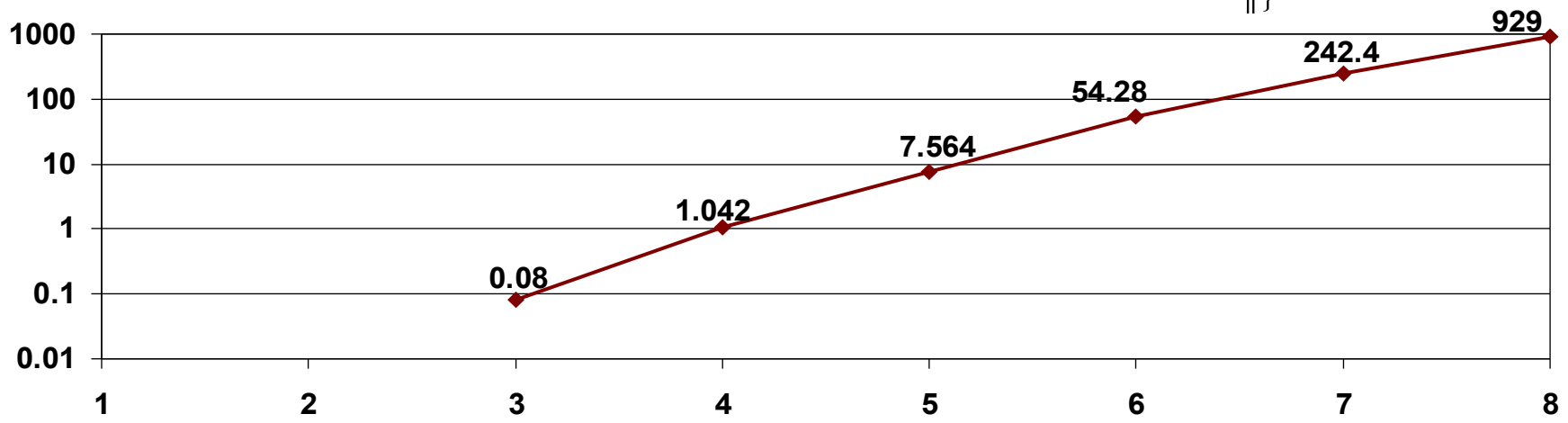
- Replace "assert φ" by
  assert $\neg\varphi \Longrightarrow \exists\, l^{\#} \in F\colon (g,l^{\#}) \in Wrong^{\#}$

Battery Class Driver

- 2 code files, 1 header, 3292 C code lines
- Promela: 141 lines, 24 states / thread

```
WantToRemove=TRUE;
if(1 == ++WorkerActive) {
   if(*) {
     if(0 == --InUseCount)
       ReadyToRemove=TRUE;
   } else {
     ++NumQueuedWorkers;
     // Work to do,
     // start working thread.
   }
}
if(0 < --InUseCount)
   await(&ReadyToRemove);
stopped=TRUE;
```

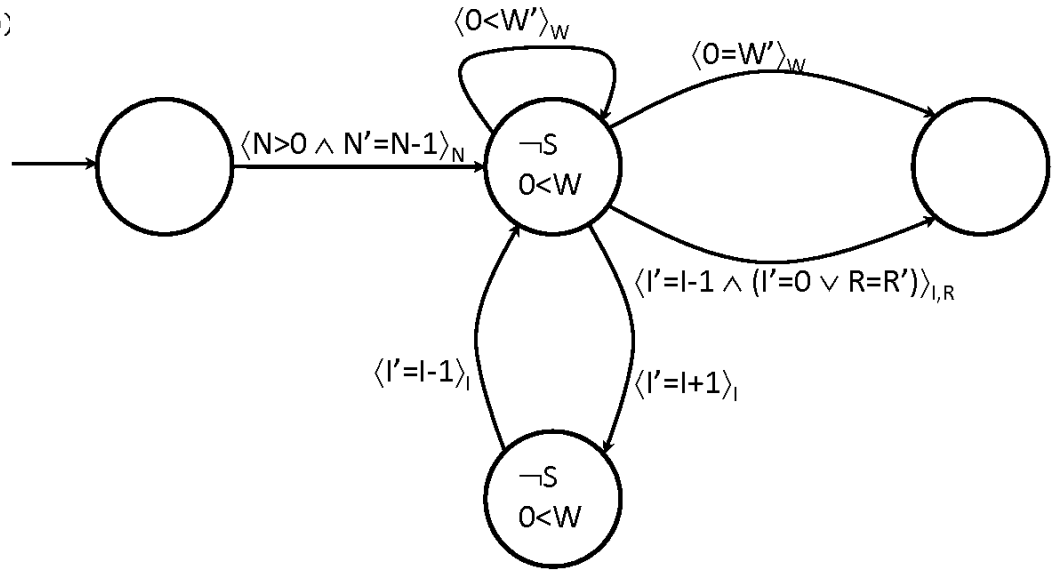
```
atomic {
   await(NumQueuedWorkers>0);
   NumQueuedWorkers--;
}
unsigned long i;
while(TRUE) {
   if(WantToRemove) {
     if(0 == --InUseCount)
       ReadyToRemove=TRUE;
     break;
   }
   if(*) {
     ++InUseCount;
     if(WantToRemove)
       --InUseCount;
     else
       --InUseCount;
   }
   i = --WorkerActive;
   if(0==i) break;
   if(1!=i) WorkerActive=1;
}
```



Abstract Threads: Battery Class Driver

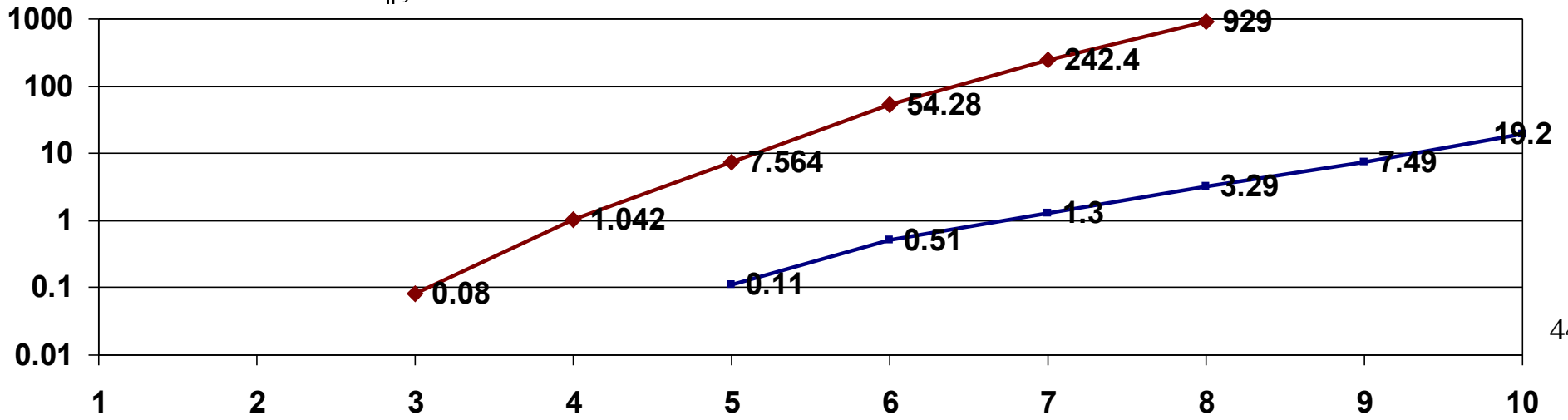
```
WantToRemove=TRUE;
if(1 == ++WorkerActive) {
   if(*) {
     if(0 == --InUseCount)
       ReadyToRemove=TRUE;
   } else {
     ++NumQueuedWorkers;
     // Work to do,
     // start working thread.
   }
}
if(0 < --InUseCount)
   await(&ReadyToRemove);
stopped=TRUE;
```

```
atomic {
   await(NumQueuedWorkers>0)
   NumQueuedWorkers--;
}
unsigned long i;
while(TRUE) {
   if(WantToRemove) {
     if(0 == --InUseCount)
       ReadyToRemove=TRUE;
     break;
   }
   if(*) {
     ++InUseCount;
     if(WantToRemove)
       --InUseCount;
     else
       --InUseCount;
   }
   i = --WorkerActive;
   if(0==i) break;
   if(1!=i) WorkerActive=1;
}
```



Boogie: 30 s.

SPIN:



- Owicki-Gries
- Rely-guarantee reasoning
- Thread-modular model checking
- Cartesian abstraction
- Thread simplification
- Refinement