

Abstraction-Based Techniques for the Verification of Concurrent Software

Csanád Telbisz



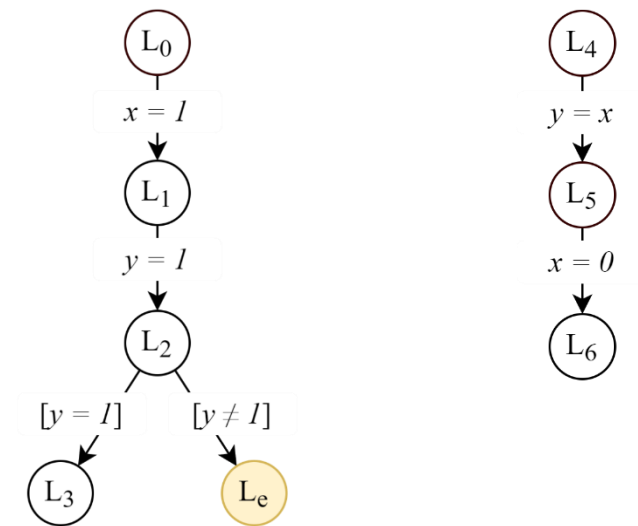
**Critical Systems
Research Group**

Outline

- Setup, formal representation
- Abstraction-based partial order reduction
- Abstract Data-Flow-Based Statement Reduction
"concurrent cone-of-influence"

Concurrent Software Verification

Csanád Telbisz



Concurrent Control-Flow Automata

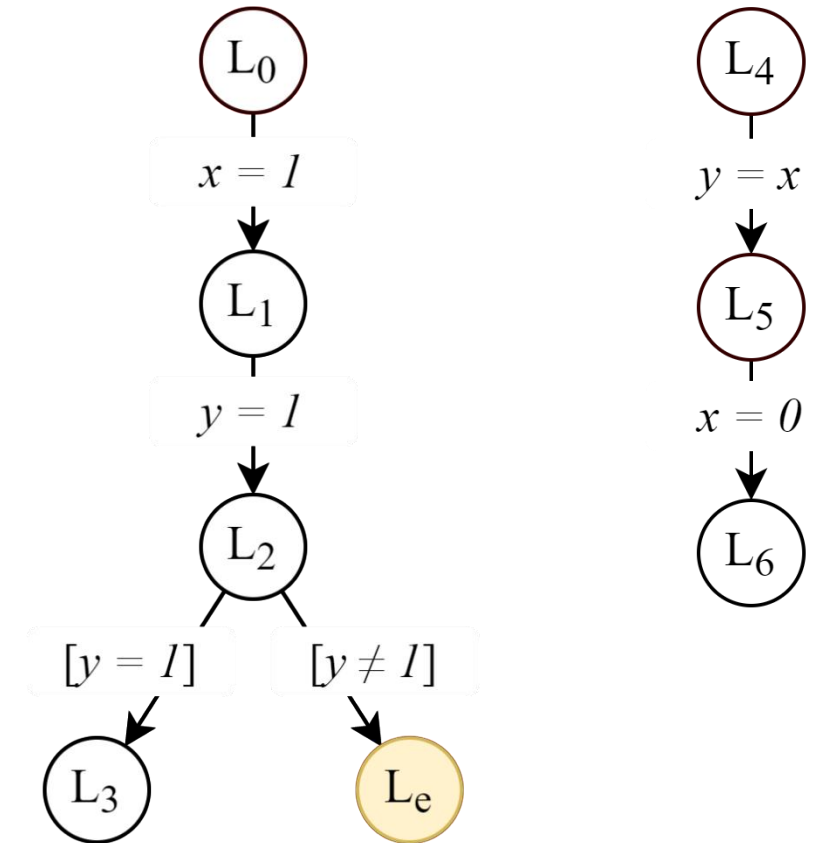
```
int x, y;

void thread1() {
    x = 1;
    y = 1;
    if (y != 1) {
        reach_error();
    }
}

void thread2() {
    y = x;
    x = 0;
}

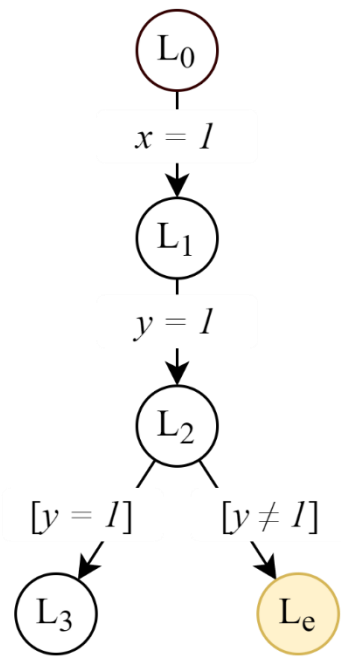
...
```

Source code (C)

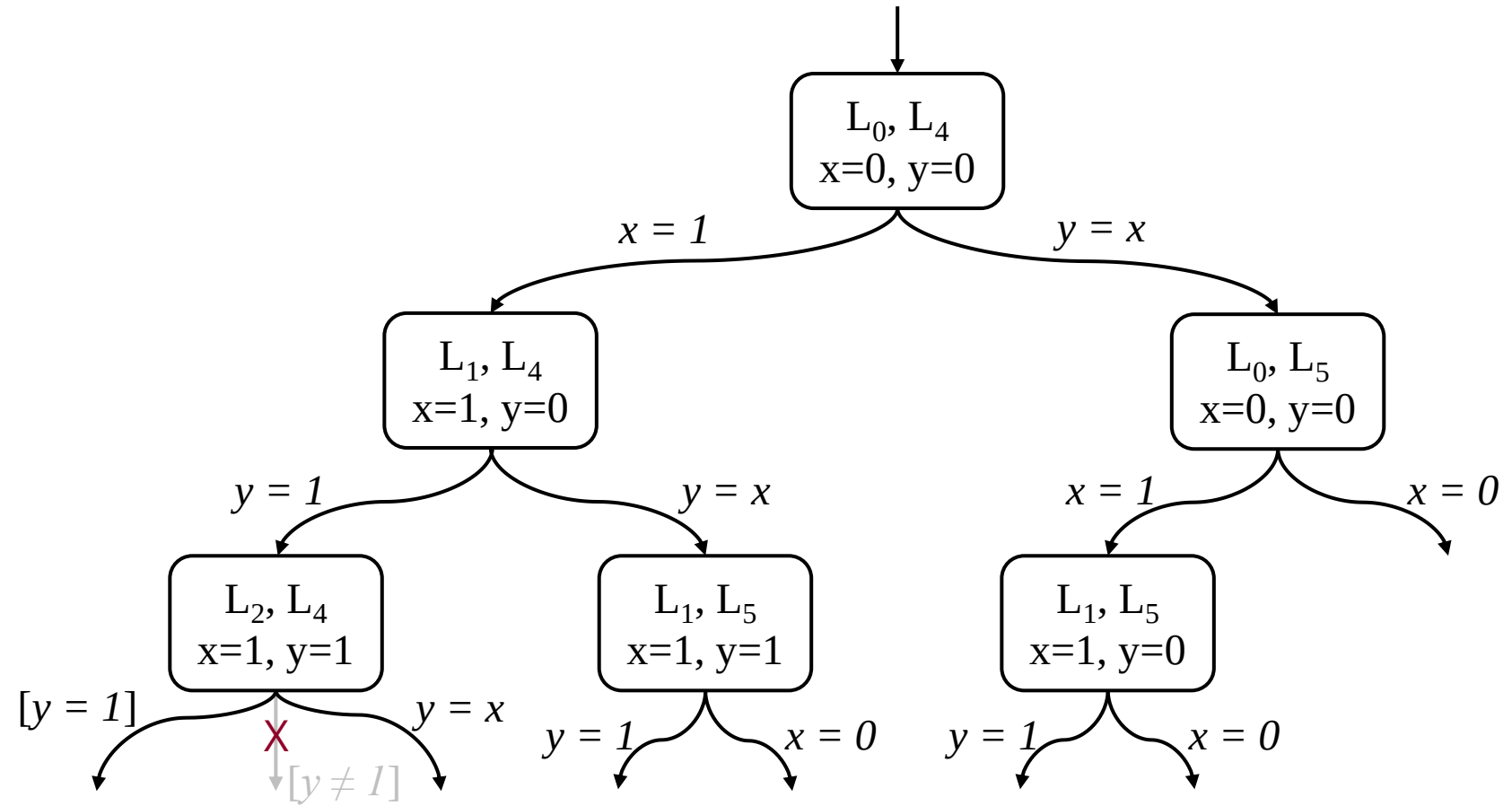
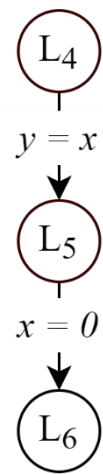


CFA (Control-Flow Automata)

Model Checking with State Space Exploration

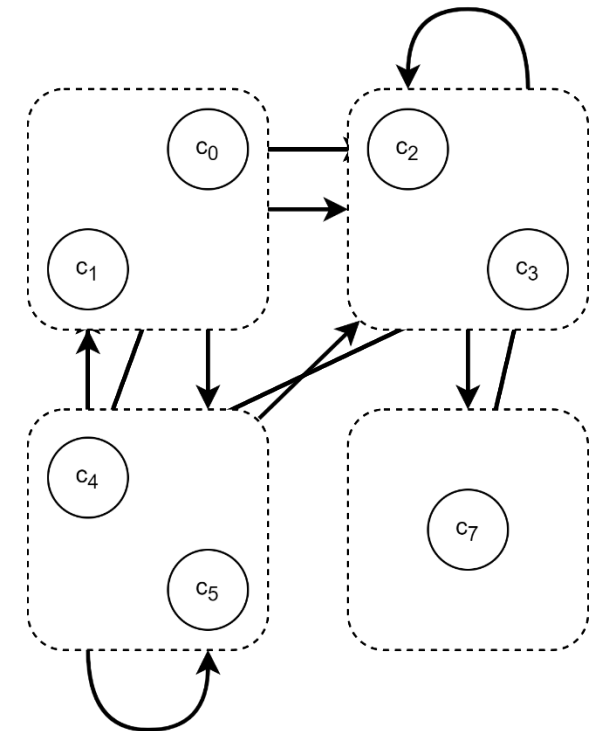


CFA



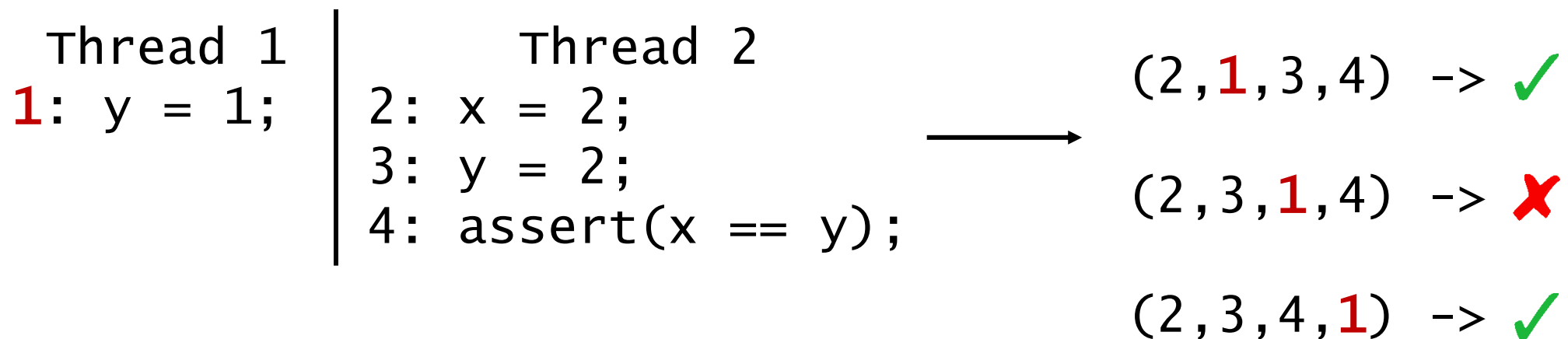
Abstraction

- Some variables are represented by fewer values
→ by abstract variables
- Precision (Π): set of abstract variables (tracked pieces of information)
- **Explicit-value abstraction:**
a subset of variables are tracked
Abstract variable: a tracked variable
E.g. $\Pi = \{x, y\}$
- **Predicate abstraction:**
predicates are tracked about variables
Abstract variable: a tracked predicate
E.g. $\Pi = \{x > 0, y = z - 2\}$



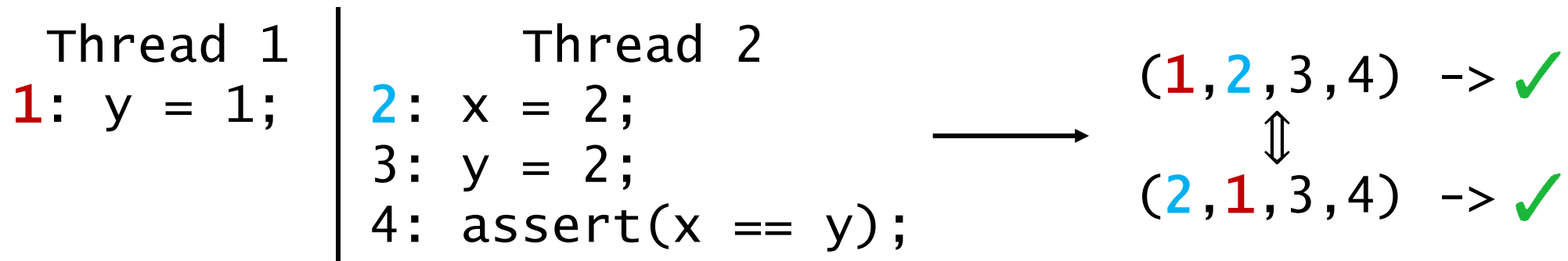
Problem with Concurrency

- Concurrent software
 - ⇒ many possible thread interleavings
 - ⇒ state space grows exponentially with the number of operations and threads



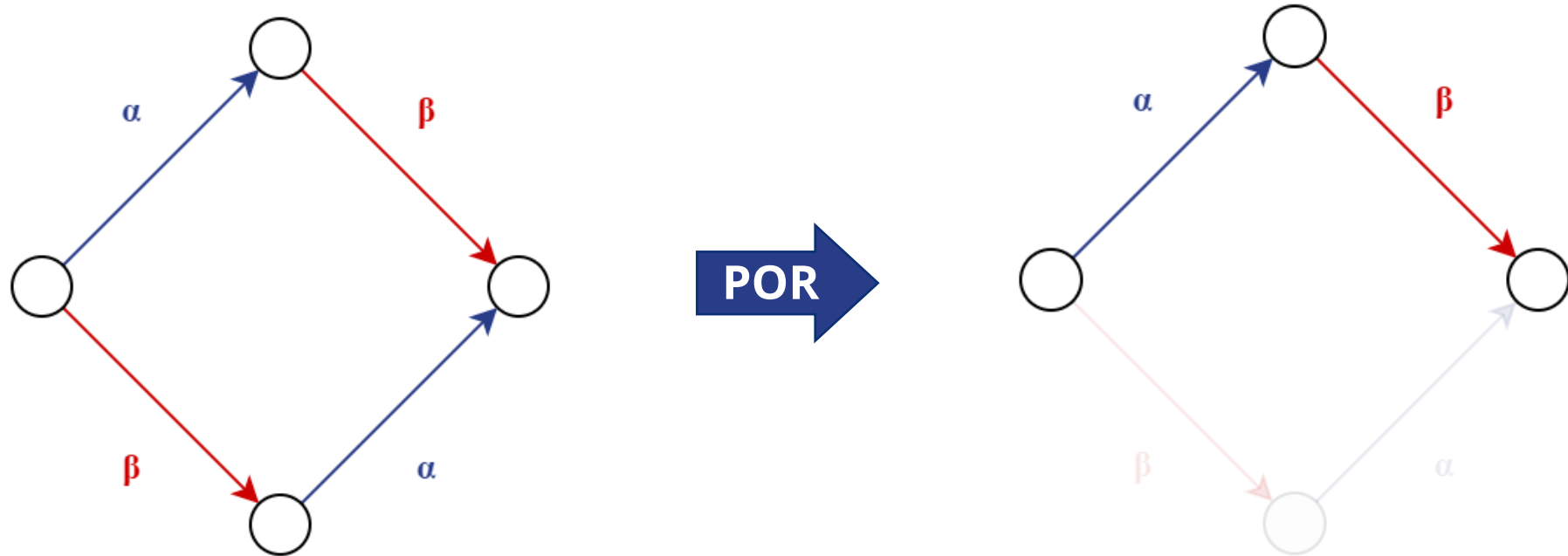
Partial Order Reduction (POR)

- Basic concept: let's identify equivalent executions
- Swapping **independent** operations does not change the result

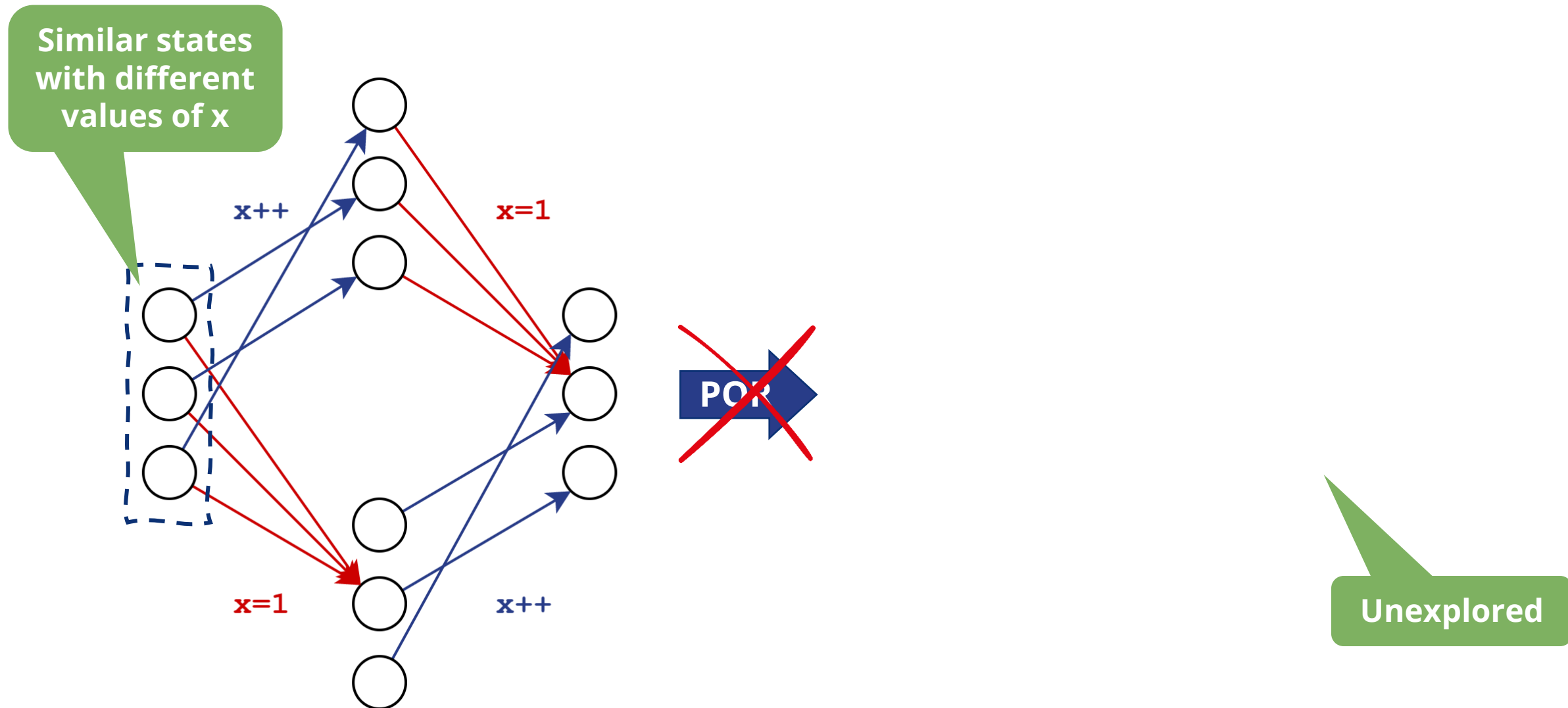


Partial Order Reduction (POR)

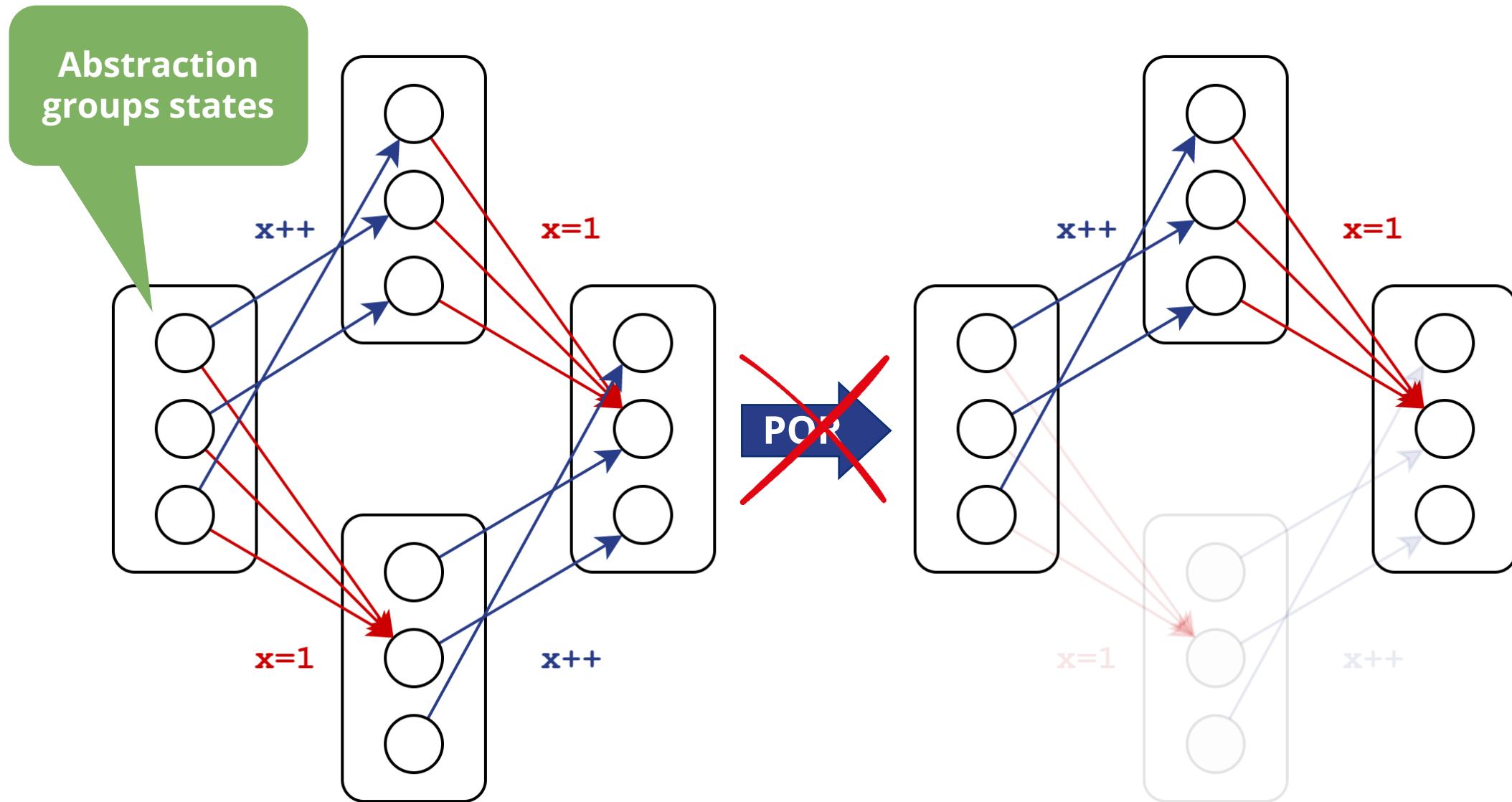
If α and β are commutative, it is enough to check one of the two paths:



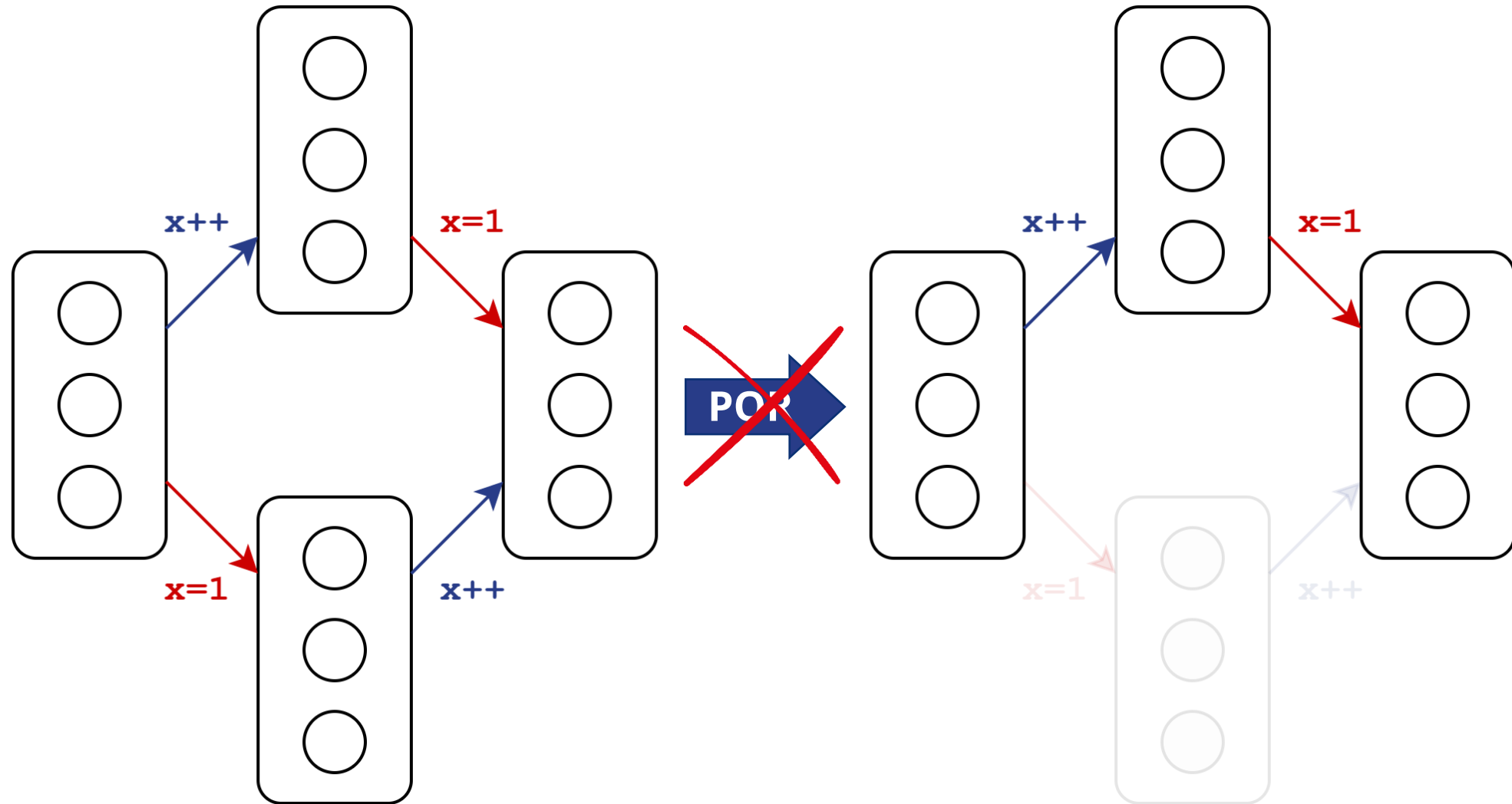
Partial Order Reduction (POR)



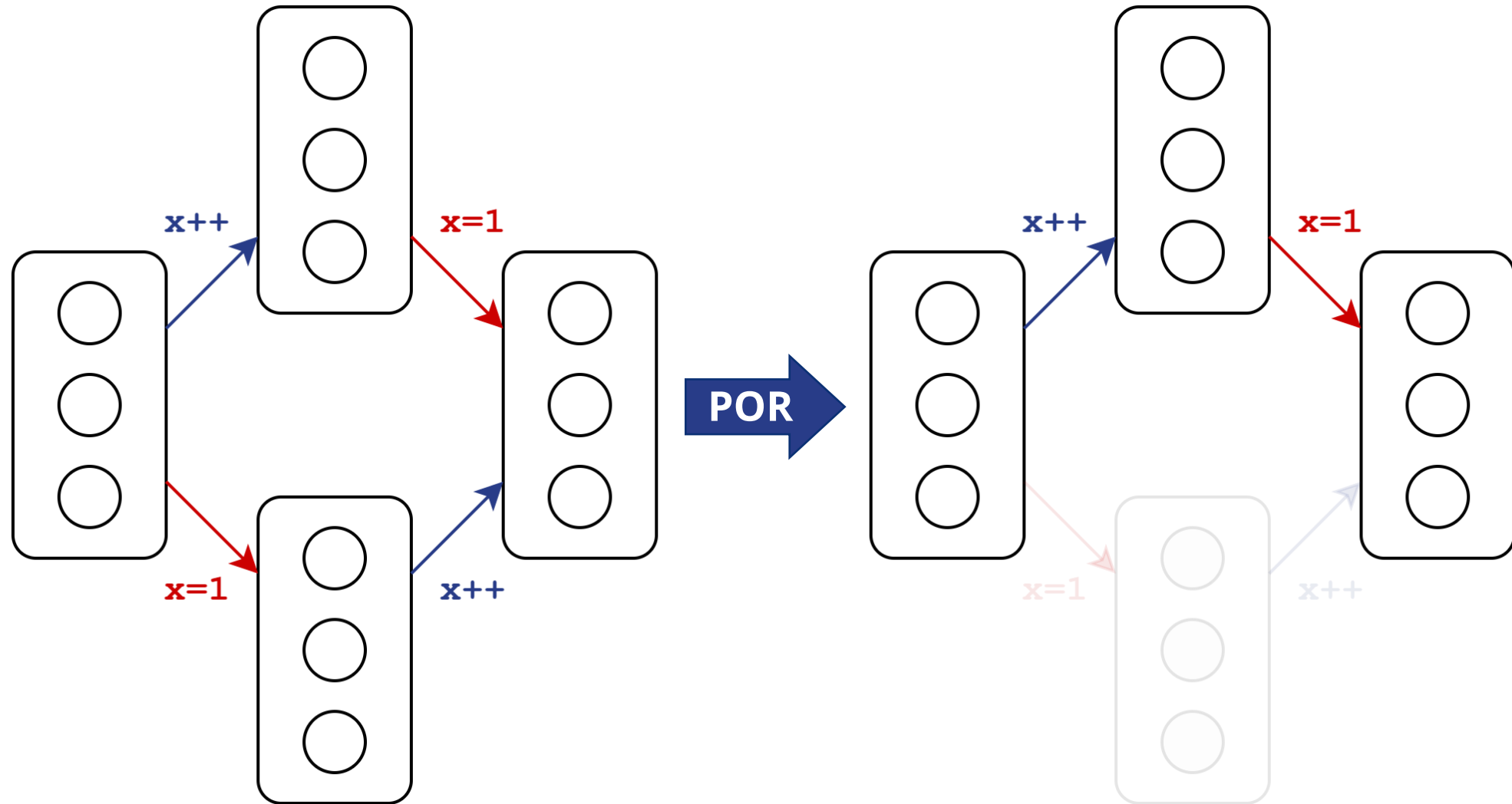
Partial Order Reduction (POR)



Partial Order Reduction (POR)



Partial Order Reduction (POR)

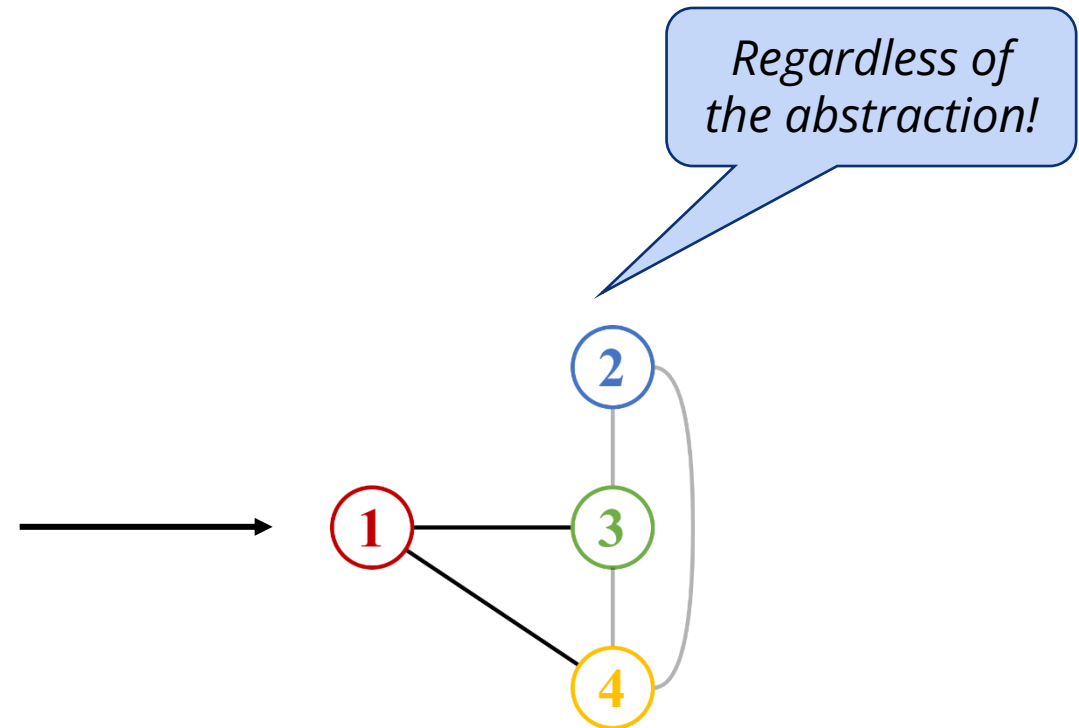


Dependency Relation

Two operations α and β are independent if:

- α and β are in different processes
- α and β access different variables

Thread 1	Thread 2
1 : $y = 1;$	2 : $x = 2;$
	3 : $y = 2;$
	4 : $\text{assert}(x == y);$

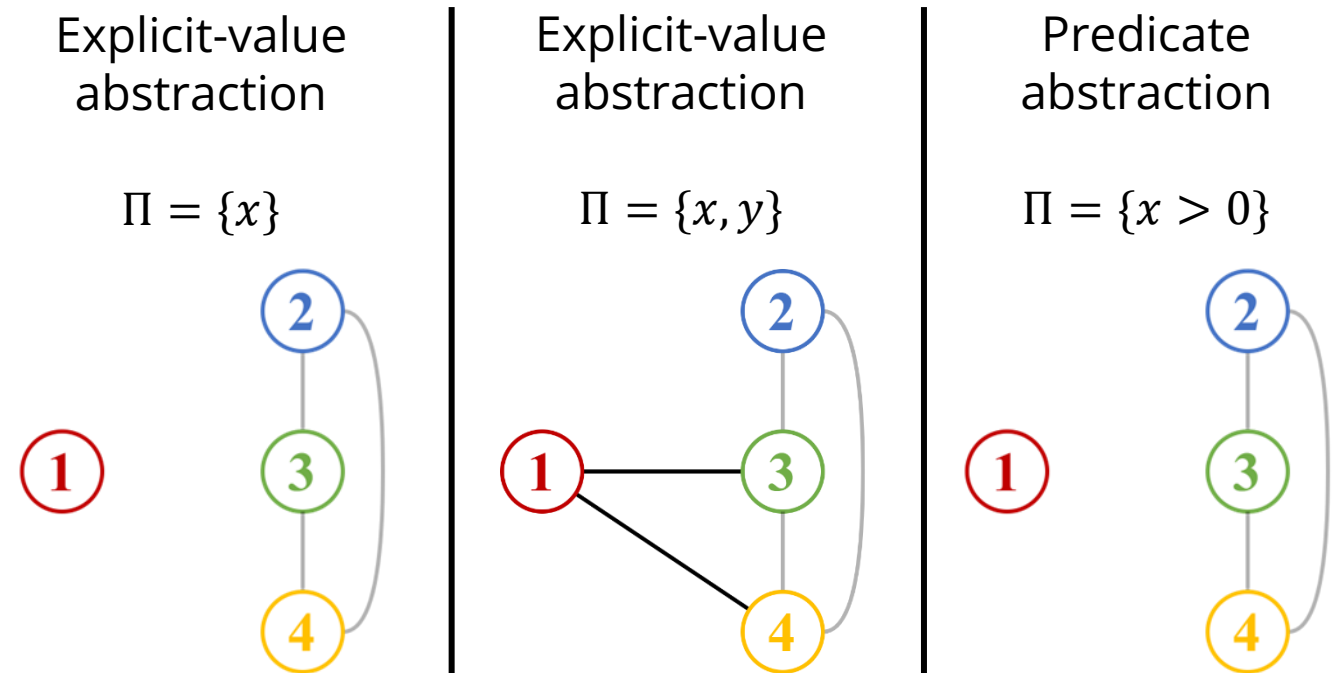


Abstraction-Aware Dependency Relation

Two operations α and β are independent if:

- α and β are in different processes
- α and β do not both access the same variable if we track any information about this variable in the current abstraction

Thread 1	Thread 2
1 : $y = 1;$	2 : $x = 2;$
	3 : $y = 2;$
	4 : $\text{assert}(x == y);$



Abstraction-Aware Dependency Relation

Two operations α and β are independent if:

- α and β are in different processes
- α and β do not both access the same variable if we track any information about this variable in the current abstraction

Fewer dependencies

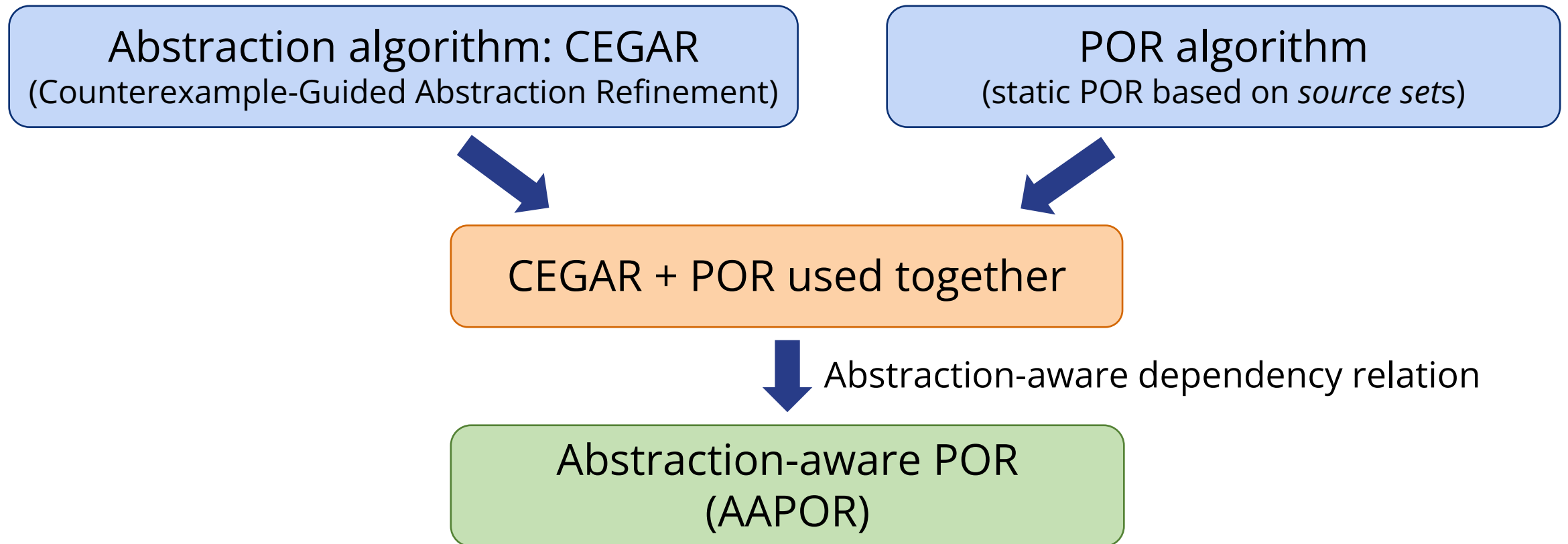


More reduction



Better performance

Abstraction-Aware Partial Order Reduction



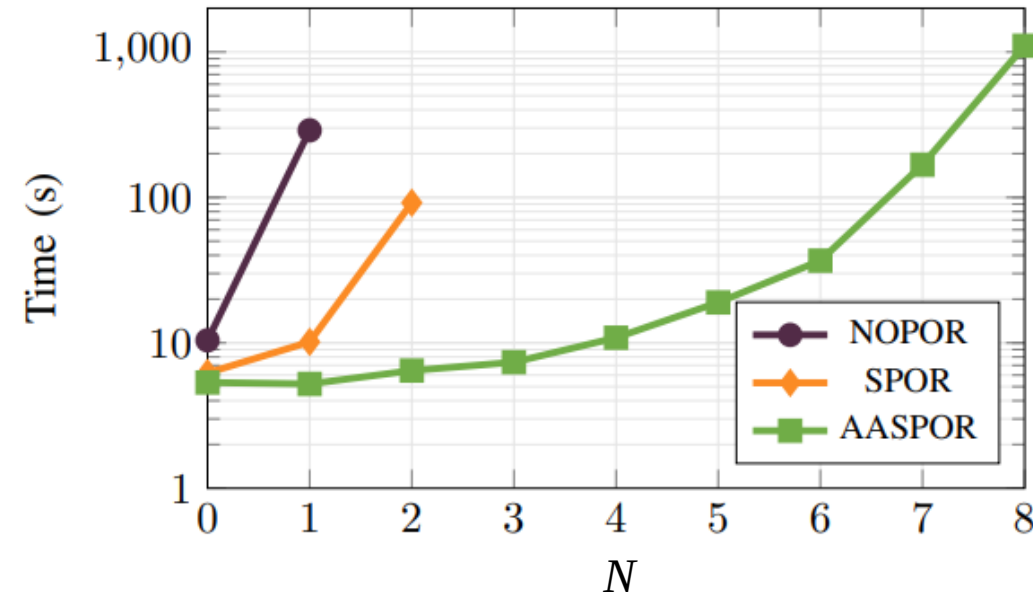
Optimizations and Limitations

- Compatibility with **lazy pruning** strategy in CEGAR:
 - CEGAR: **iterative** abstraction-based model checking algorithm
 - Lazy pruning: **reusing parts of the state-space** calculated in previous iterations
 - Cannot be trivially combined with AAPOR
 - **Some states** in the preserved parts **must be visited again**
- AAPOR **cannot be used for data race detection**
 - Data race can happen on untracked variables
 - AAPOR may evade the states where the race can be detected

Evaluation

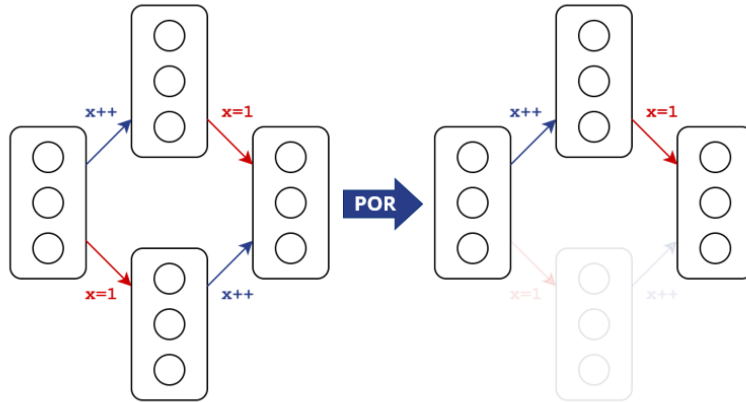
- Implemented in THETA 
- SV-COMP benchmarks (THETA can parse 598 of 714 reachability tasks):
 - Explicit-value abstraction: 18% improvement in CPU time
 - Predicate abstraction: 5% improvement
- Exponential gain in some cases:

```
initially:  $x := y := z := 0$   
  
Process  $p_0$  | Processes  $p_1 - p_N$   
repeat N times:  
   $z := z + 2 * y$   
  if  $z \bmod 2 = 0$ :  
     $x := 0$   
  else:  
     $x := 1$   
    |  
    Processes  $p_{N+1} - p_{2N}$   
     $y := y * y$   
  
finally: assert ( $x * y = 0$ )
```



Source: github.com/ftsrg/theta
gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c

Partial Order Reduction (POR)



Csanád Telbisz

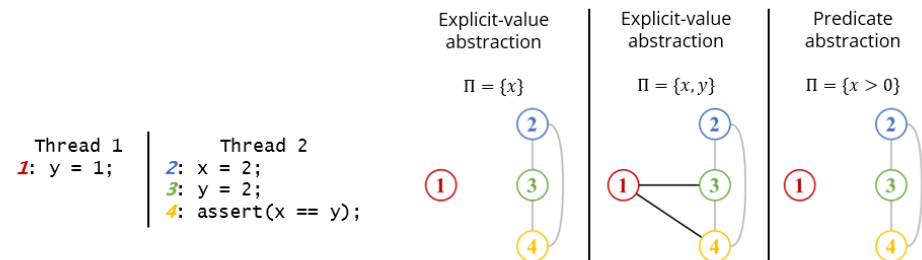
13



Abstraction-Aware Dependency Relation

Two operations α and β are independent if:

- α and β are in different processes
- α and β do not both access the same variable if we track any information about this variable in the current abstraction

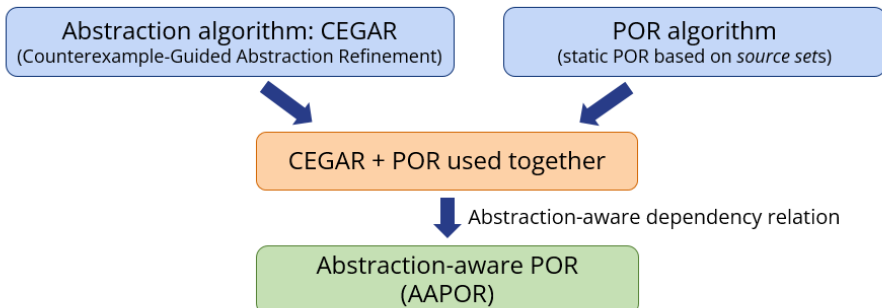


Csanád Telbisz

15



Abstraction-Aware Partial Order Reduction



Csanád Telbisz

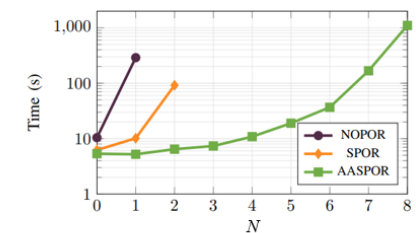
17



Evaluation

- Implemented in THETA
- SV-COMP benchmarks (THETA can parse 598 of 714 reachability tasks):
 - Explicit-value abstraction: 18% improvement in CPU time
 - Predicate abstraction: 5% improvement
- Exponential gain in some cases:

initially: $x=y=z=0$
 Process p_0 :
 repeat N times:
 $z := z+2*y$
 if $z \bmod 2 = 0$:
 $x := 0$
 else:
 $x := 1$
 finally: assert ($x*y=0$)
 Processes $p_1 - p_N$:
 $y := y+1$
 Processes $p_{N+1} - p_{2N}$:
 $y := y*y$



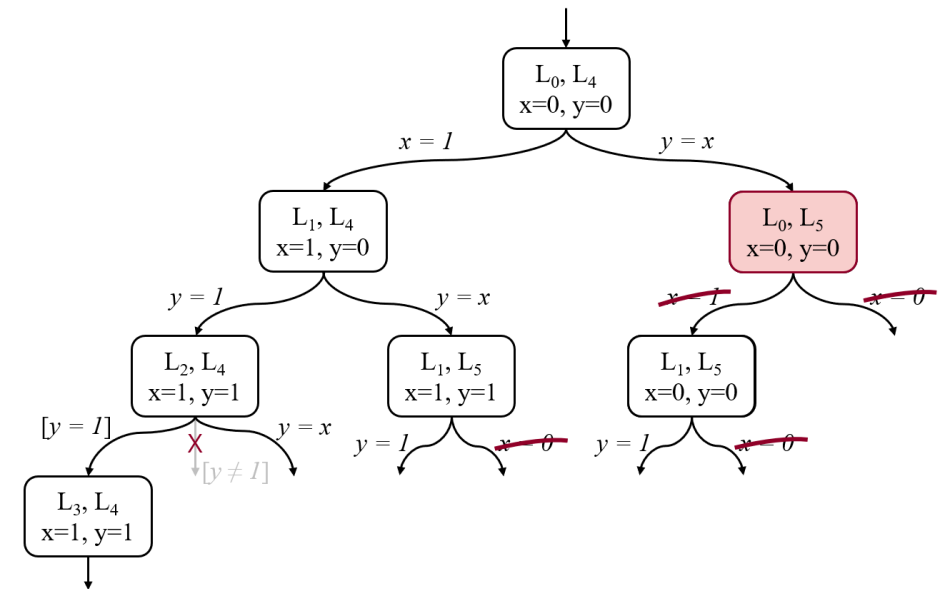
Source: github.com/ftsrg/theta
gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c

BSc Thesis Defence - 10/01/2023
 Csanád Telbisz

19



Abstract Data-Flow-Based Statement Reduction



Cone-of-Influence Techniques

- Erase *irrelevant* variables from the model

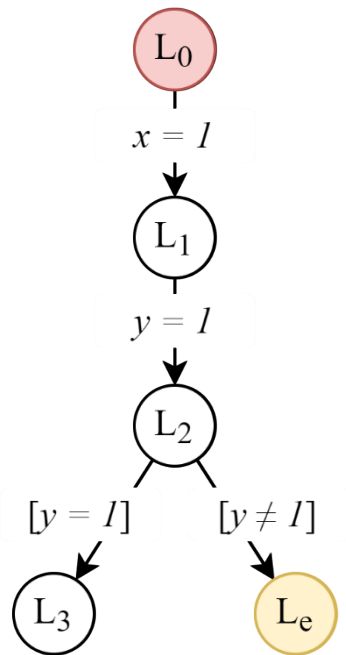
**Irrelevant variables for
checking the assertion:**
a, sum, i

```
void mean() {  
    int a, n;  
    int sum = 0;  
    scanf("%d", n);           // number of elements  
  
    for(int i = 0; i < n; ++i) {  
        scanf("%d", &a); // read next element  
        sum += a;  
    }  
  
    assert(n > 0);  
    printf("Mean: %f", (float) sum / n);  
}
```



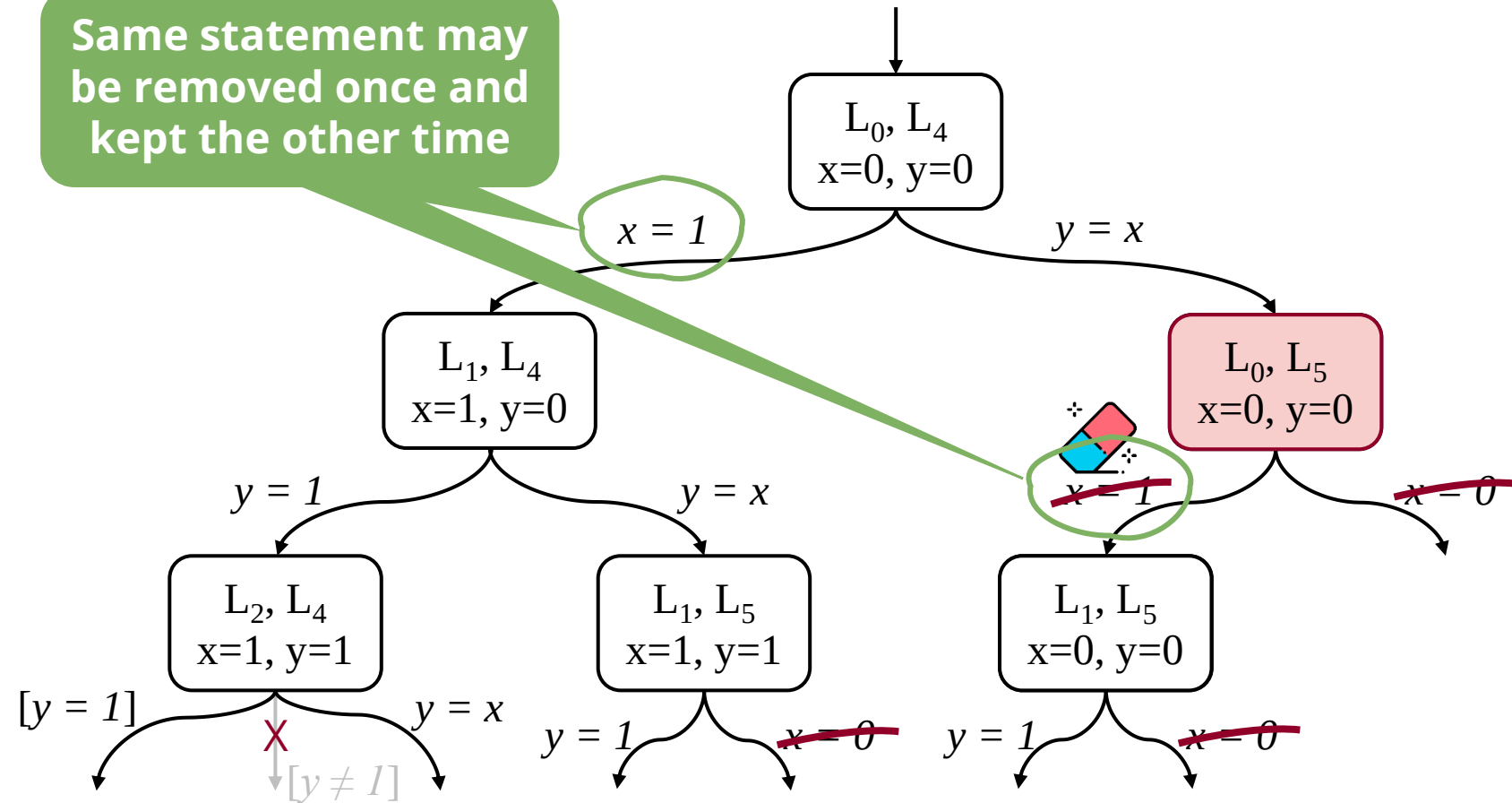
```
void mean() {  
    int n;  
  
    scanf("%d", n);  
  
  
  
    assert(n > 0);  
}
```

Fine-grained Statement Simplification



The value of x is not used after this state!

Same statement may be removed once and kept the other time

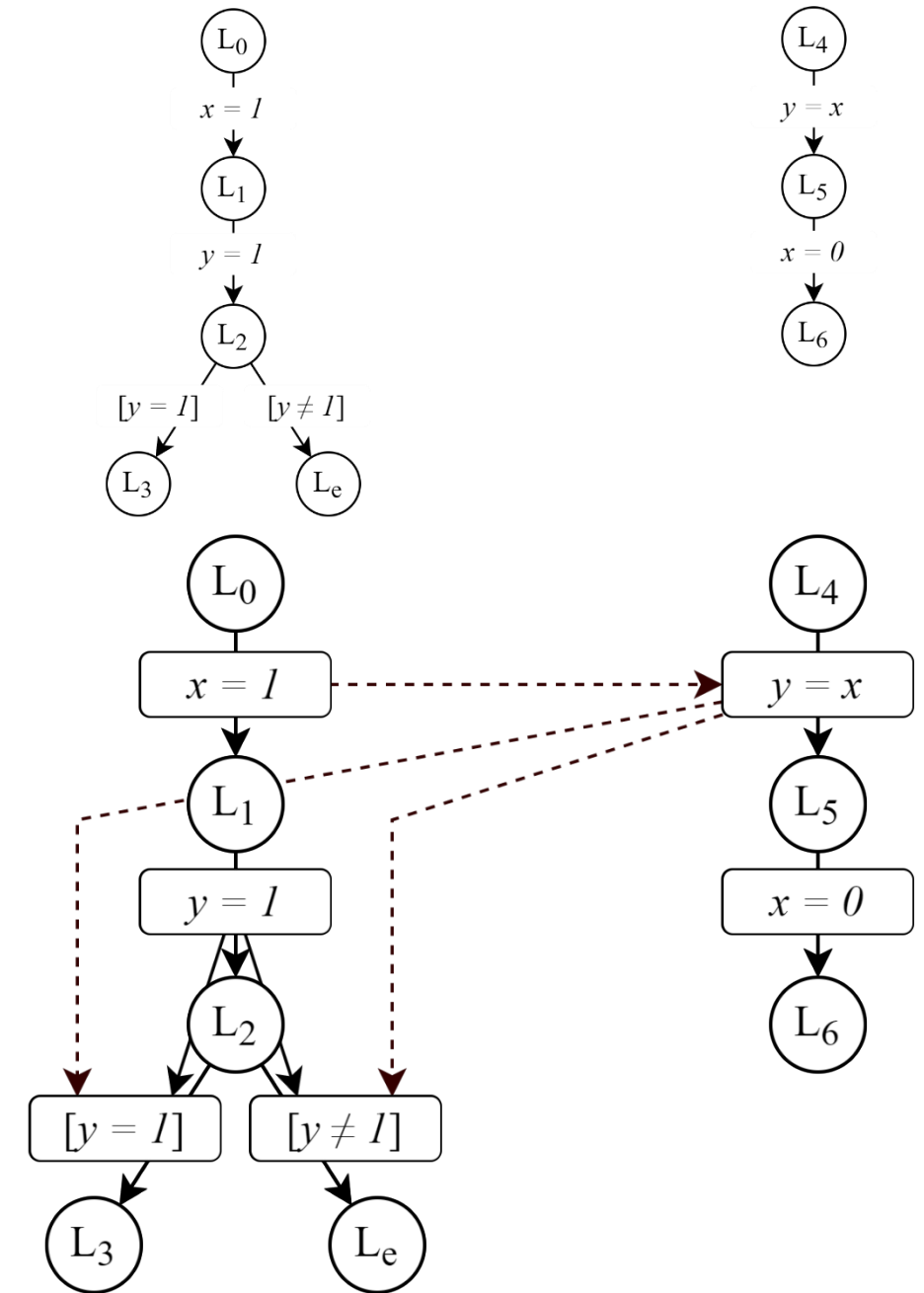


Formalization

Question: is the *current* statement *relevant* for proving the property? Remove, if not.

Abstract data-flow graph:

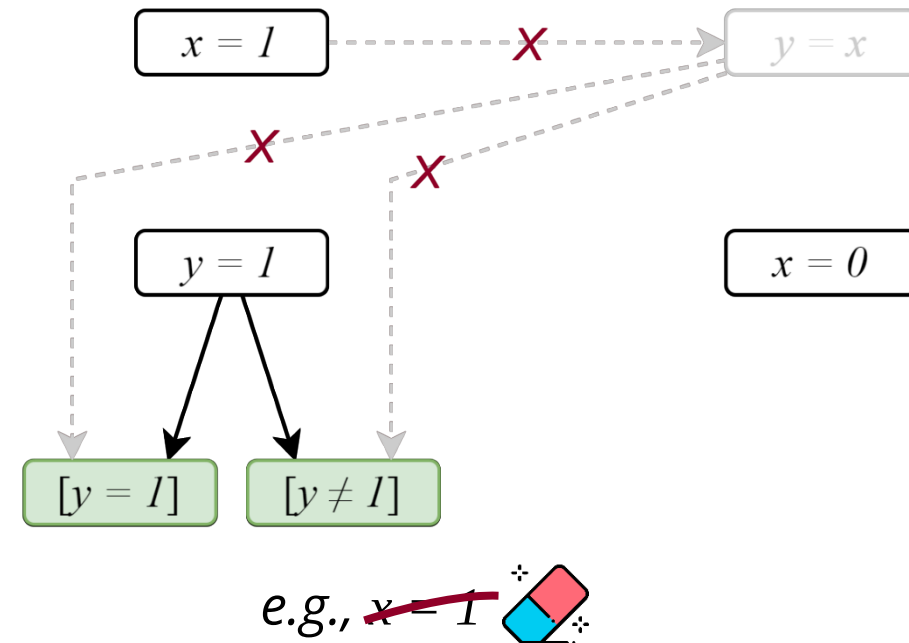
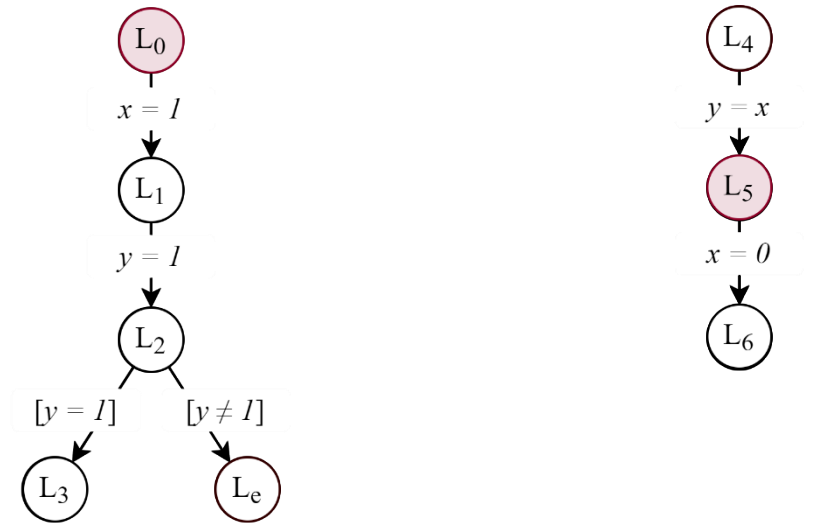
- **nodes**: program statements
- (directed) **edges**: if a statement may use the result of another one „observation relation”
 - **direct** („in-process”) edge $\longrightarrow$
 - **indirect** („inter-process”) edge $\dashrightarrow$



Formalization

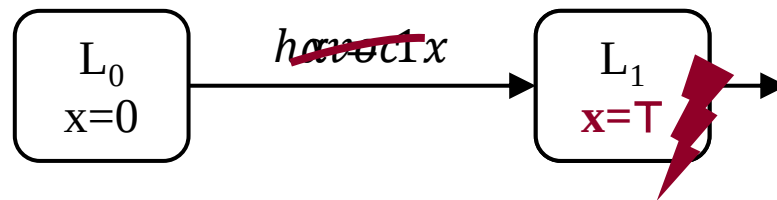
Question: is the *current* statement *relevant* for proving the property? Remove, if not.

- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process
- **Real observers**: conditional statements
 - reachability of error locations depends on them
 - **transitive observation** by a real observer:
path to a real observer in the data-flow graph
- If **no real observer** of the current statement:
it **can be reduced**



Algorithmic Integration – CEGAR

- **Found counterexamples must be checked** in CEGAR
- With the statement reduction algorithm:
 - if the statements are eliminated from a counterexample: contradiction may not be found
 - **original statements must be used** for checking counterexamples
- Another problem: **contradicting state after removed statement**
→ solution: **non-deterministic assignment** instead of complete erasure



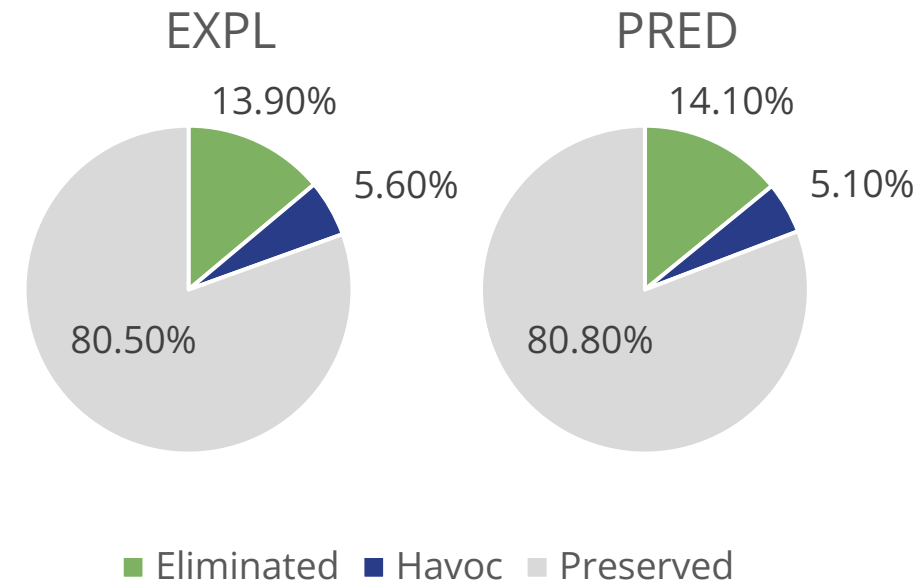
Checking counterexample
with original statement

Algorithmic Integration

- Partial-order reduction (POR):
 - **based on statement dependence**
 - **the new statement simplification algorithm erases variable accesses**
→ less dependent statements
- Algorithmic improvement: **strongly connected components with Tarjan's algorithm** for quick CFA reachability information

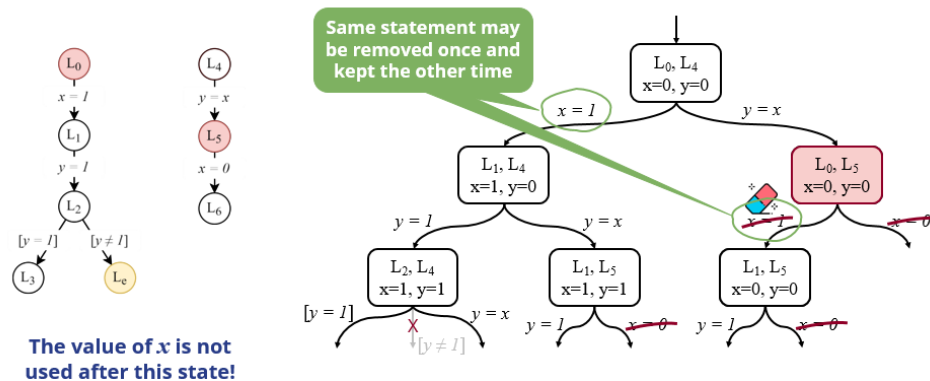
Evaluation

- Proportion of simplified statements:
~20% removed
- Successor state calculation time:
 - EXPL: 33% improvement
 - PRED: 35% improvement
- Overall CPU time:
 - EXPL: 1-2% improvement
 - PRED: 12% improvement



**Predicate abstraction is more costly to compute:
more improvement expected**

Fine-grained Statement Simplification



Csanád Telbisz

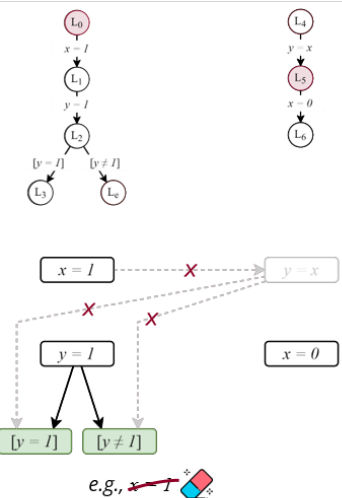
23



Formalization

Question: is the *current* statement *relevant* for proving the property? Remove, if not.

- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process
- **Real observers**: conditional statements
 - reachability of error locations depends on them
 - **transitive observation** by a real observer:
path to a real observer in the data-flow graph
- If **no real observer** of the current statement:
it **can be reduced**



Csanád Telbisz

2



Algorithmic Integration – CEGAR

- **Found counterexamples must be checked** in CEGAR
- With the statement reduction algorithm:
 - if the statements are eliminated from a counterexample: contradiction may not be found
 - **original statements must be used** for checking counterexamples
- Another problem: **contradicting state after removed statement**
 → solution: **non-deterministic assignment** instead of complete erasure



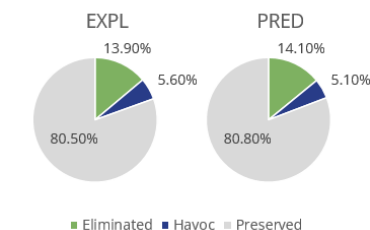
Csanád Telbisz

26



Evaluation

- Proportion of simplified statements:
 - ~20% removed
- Successor state calculation time:
 - EXPL: 33% improvement
 - PRED: 35% improvement
- Overall CPU time:
 - EXPL: 1-2% improvement
 - PRED: 12% improvement



**Predicate abstraction is more costly to compute:
more improvement expected**

Students' Scientific Conference (TDK) – 2023
Csanád Telbisz

2

