

On-the-fly Cone-of-Influence Reduction for Model Checking Concurrent Software

Csanád Telbisz, Levente Bajczi, Dániel Szekeres,
András Vörös



**Critical Systems
Research Group**

Context: Problem & Representation

```
int x, y;

void thread1() {
    x = 1;
    y = 1;
    assert(y == 1);
}

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

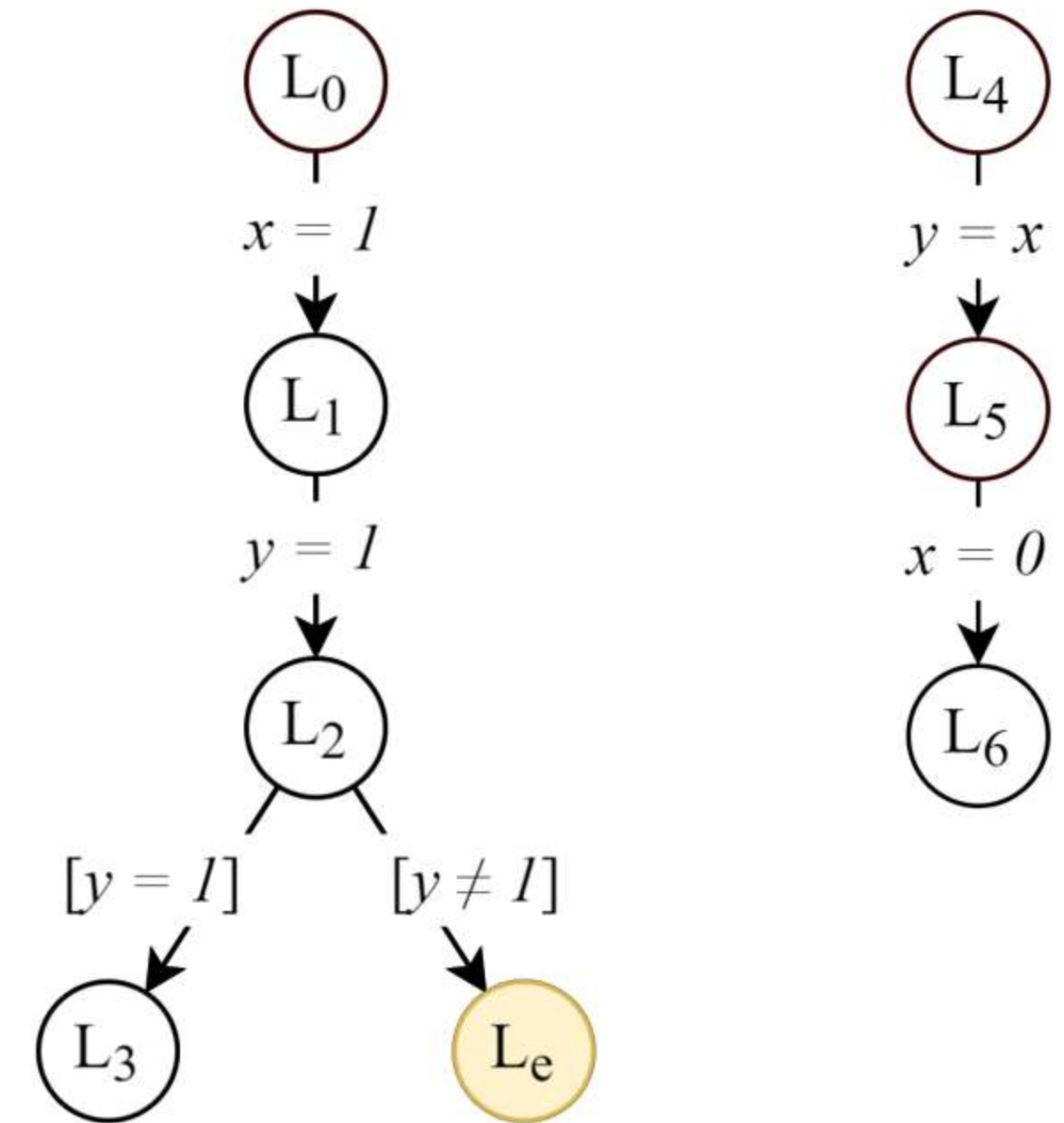
...
```

Source code (C)

Context: Problem & Representation

```
int x, y;  
  
void thread1() {  
    x = 1;  
    y = 1;  
    assert(y == 1);  
}  
  
void thread2() {  
    y = x;  
    x = 0;  
}  
  
...
```

Source code (C)



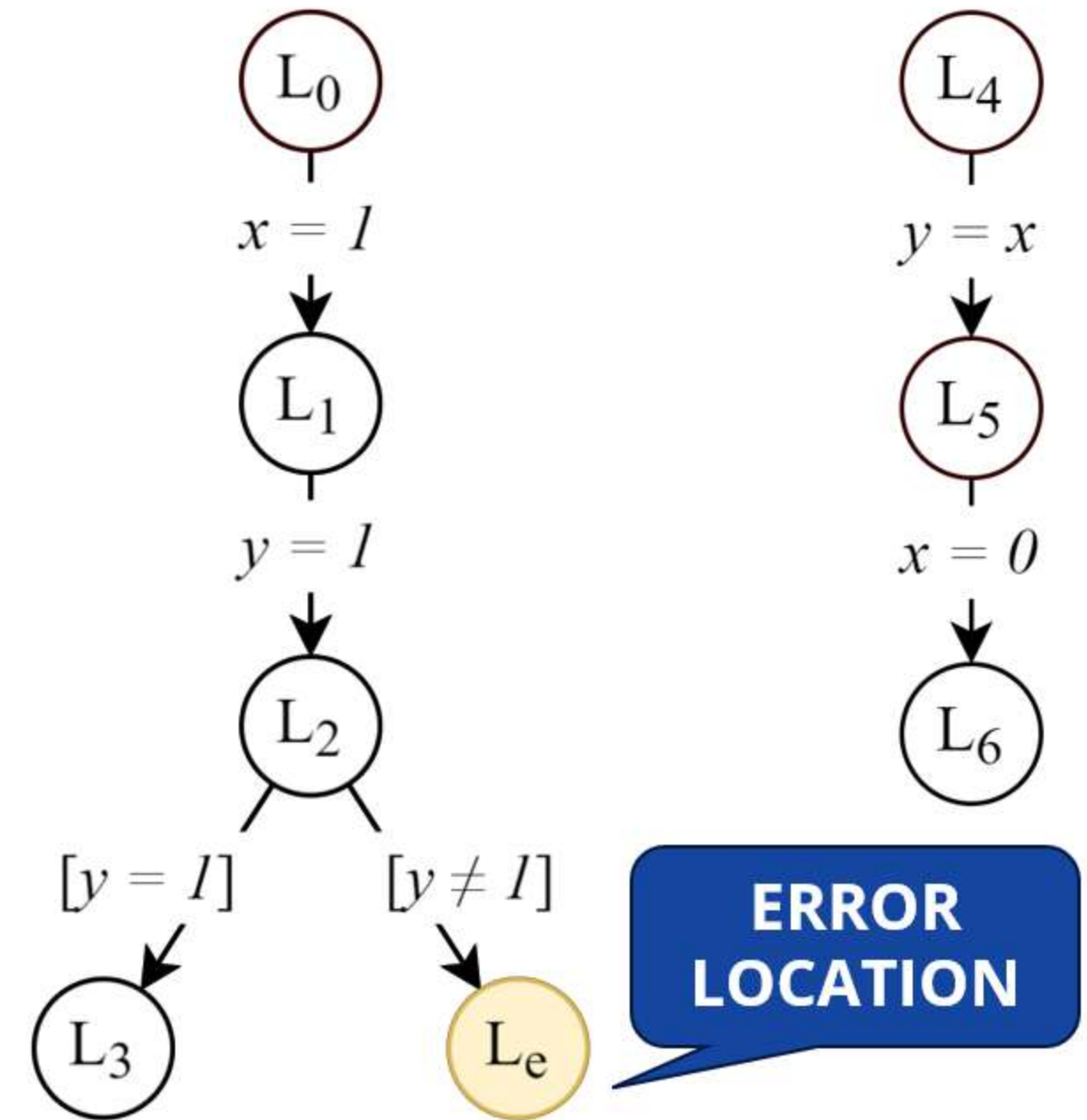
CFA (Control-Flow Automata)

Context: Problem & Representation

```
int x, y;  
  
void thread1() {  
    x = 1;  
    y = 1;  
    assert(y == 1);  
}  
  
void thread2() {  
    y = x;  
    x = 0;  
}  
  
...
```

SPECIFICATION

Source code (C)



**ERROR
LOCATION**

CFA (Control-Flow Automata)

Cone-of-Influence Techniques

- Erase **irrelevant** model elements from the model

Cone-of-Influence Techniques

- Erase **irrelevant** model elements from the model

```
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);  
}
```

Cone-of-Influence Techniques

- Erase **irrelevant** model elements 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);  
}
```

Cone-of-Influence Techniques

- Erase **irrelevant** model elements from the model

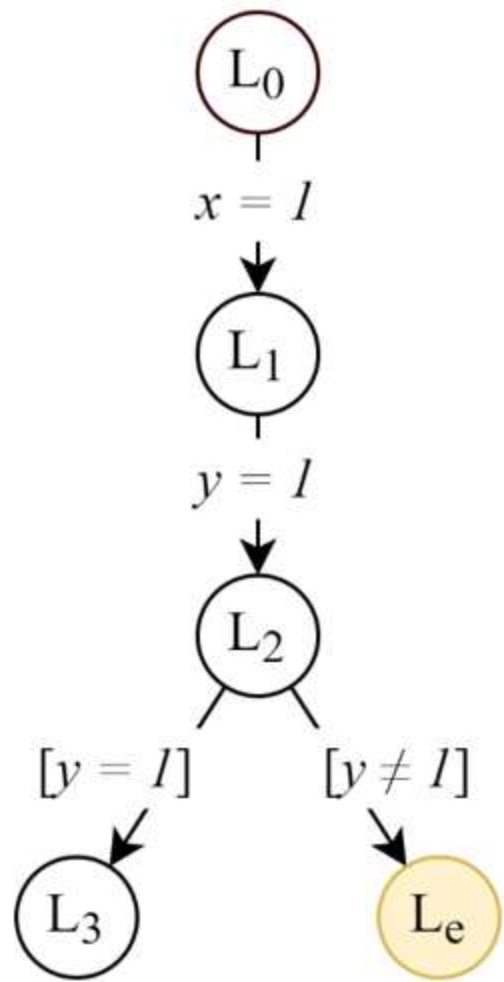
Irrelevant variables for
checking the assertion:
a, sum, i

```
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);  
}
```

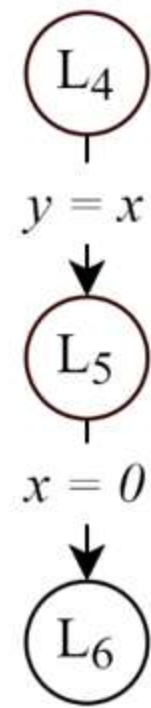


```
void mean() {  
    int n;  
  
    scanf("%d", n);  
  
  
  
    assert(n > 0);  
}
```

Model Checking with State Space Exploration

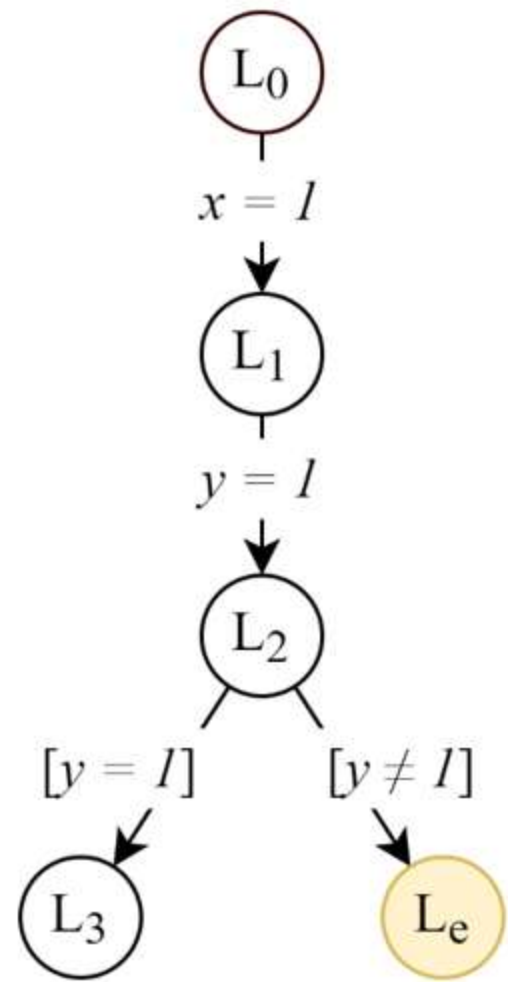


CFA

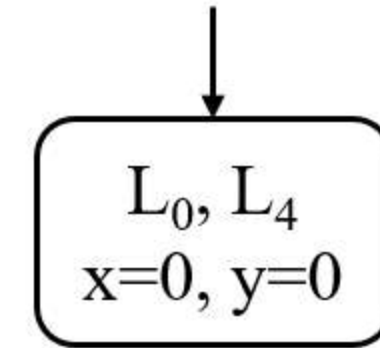


State space

Model Checking with State Space Exploration

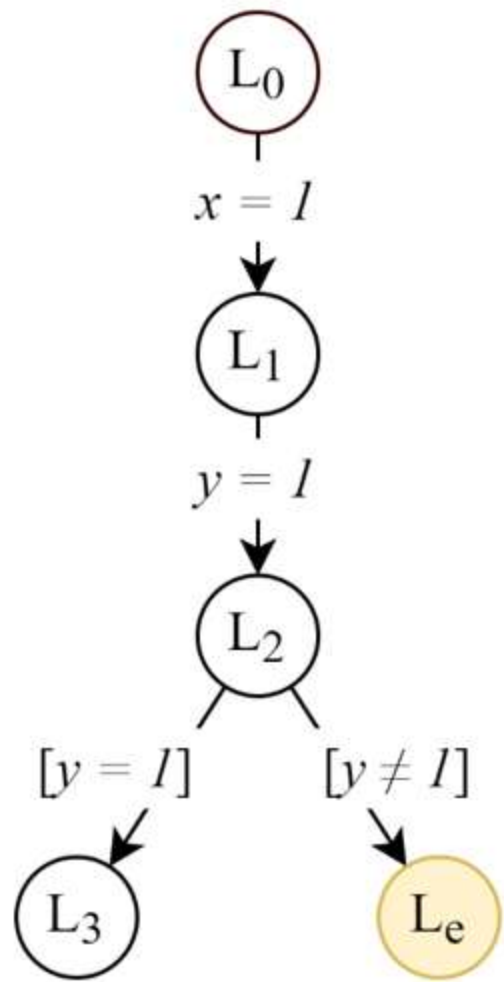


CFA

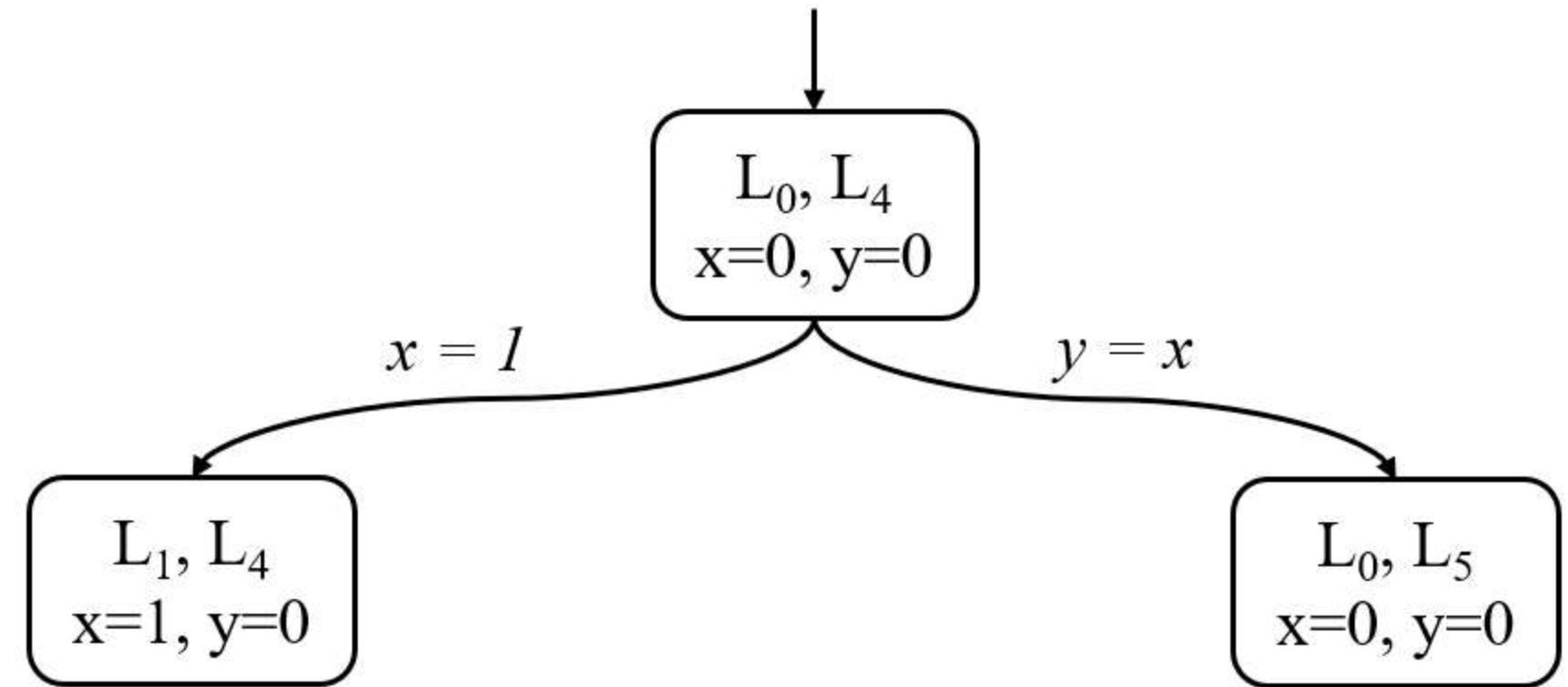
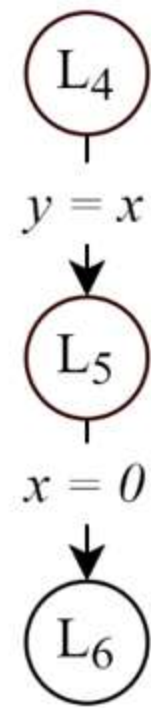


State space

Model Checking with State Space Exploration

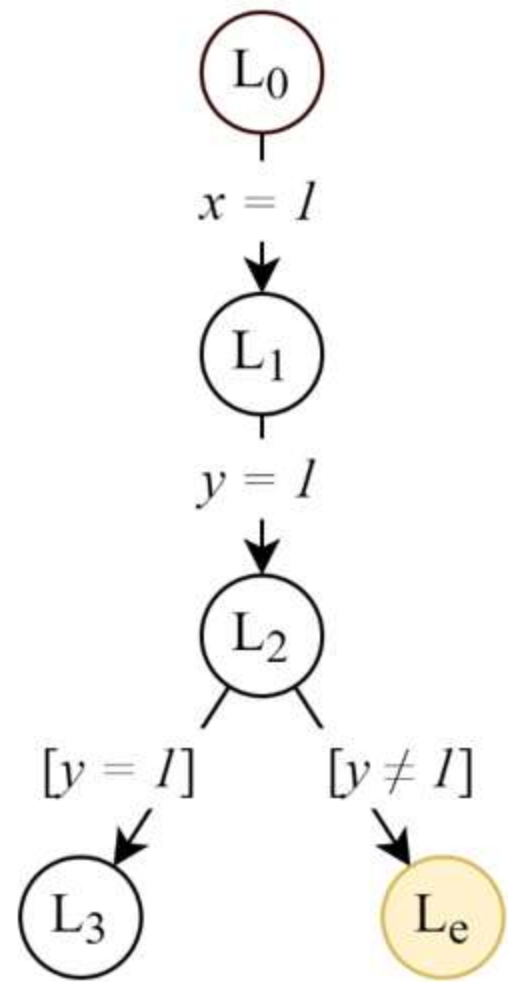


CFA

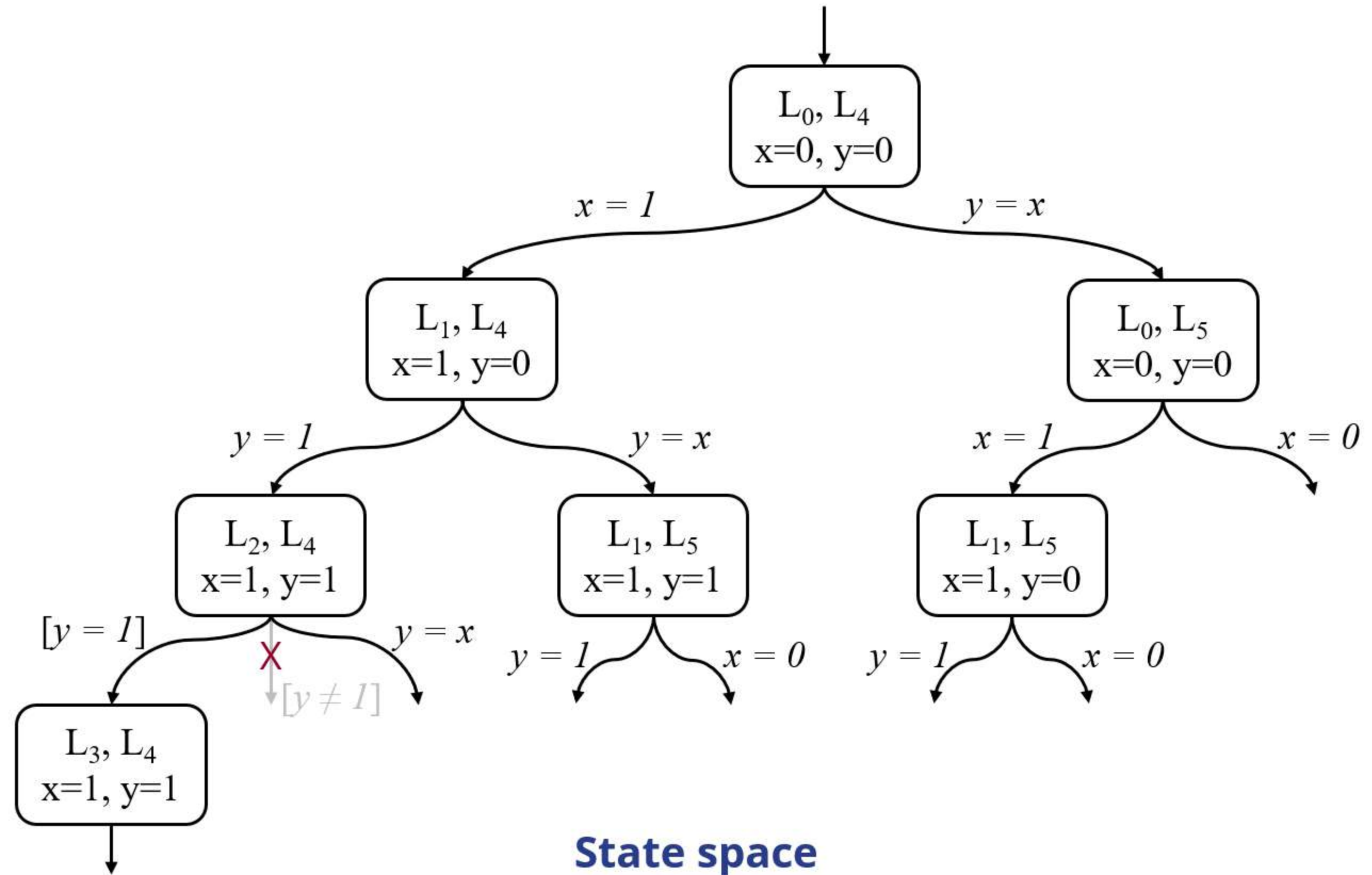
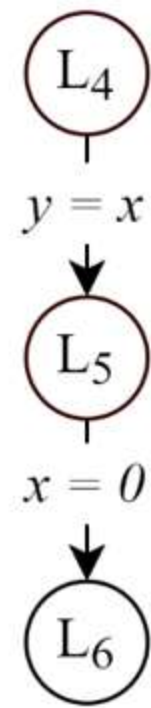


State space

Model Checking with State Space Exploration

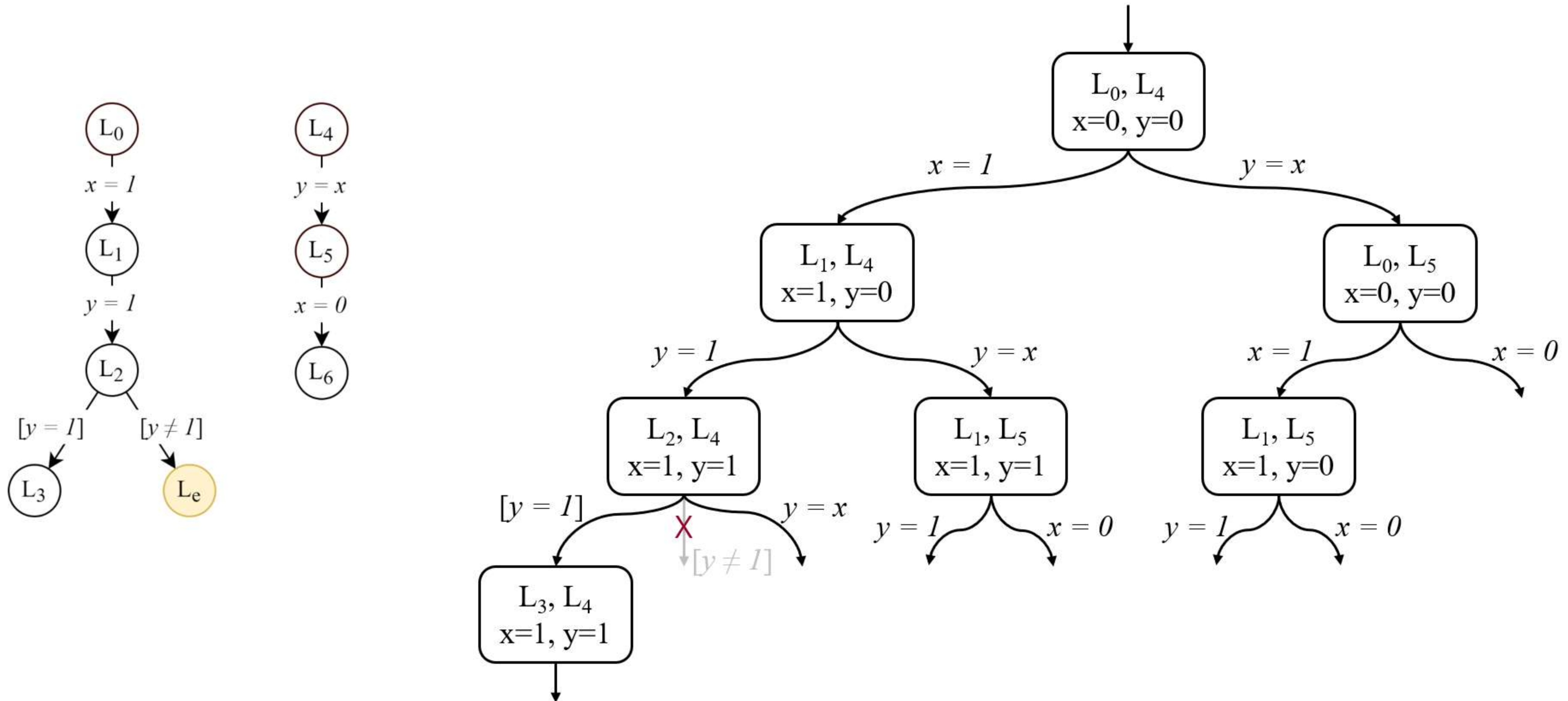


CFA

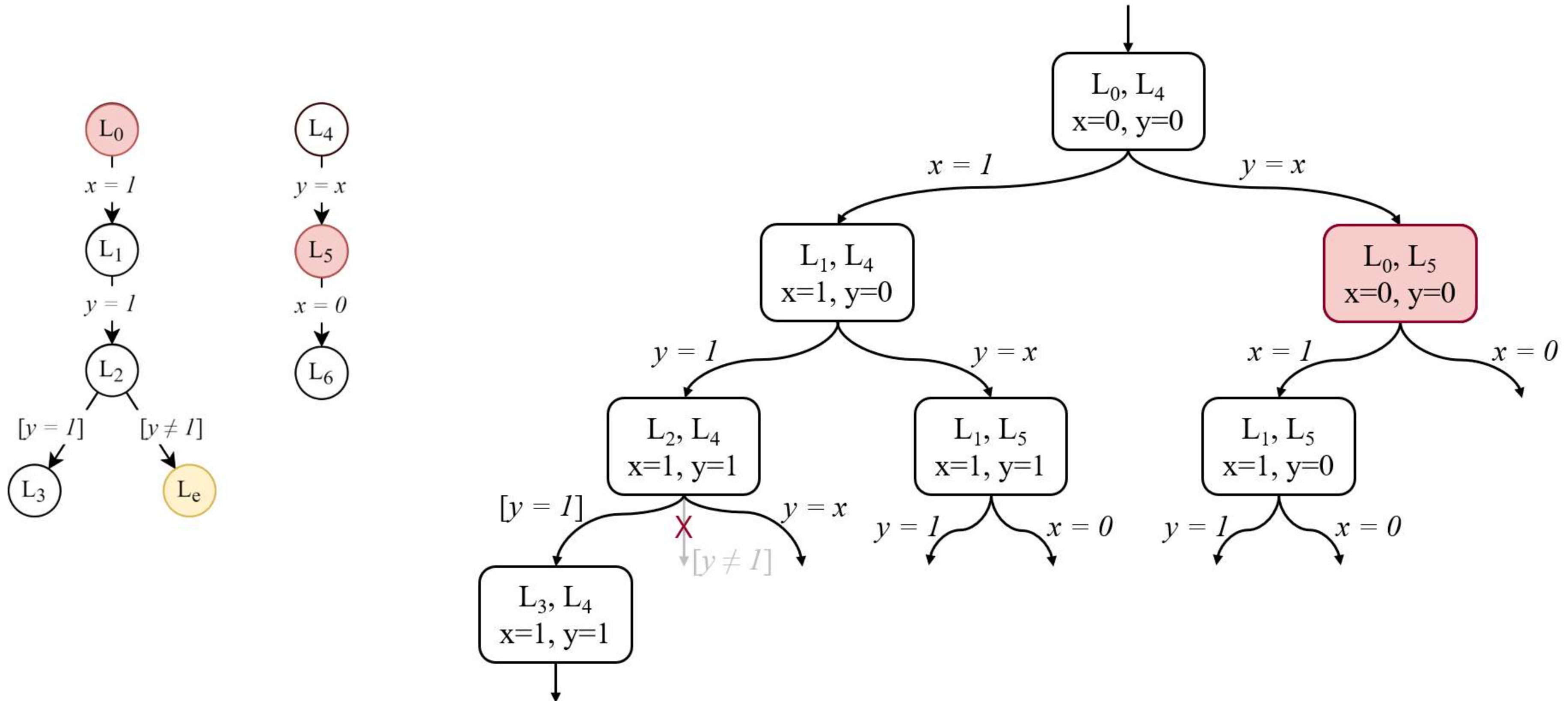


State space

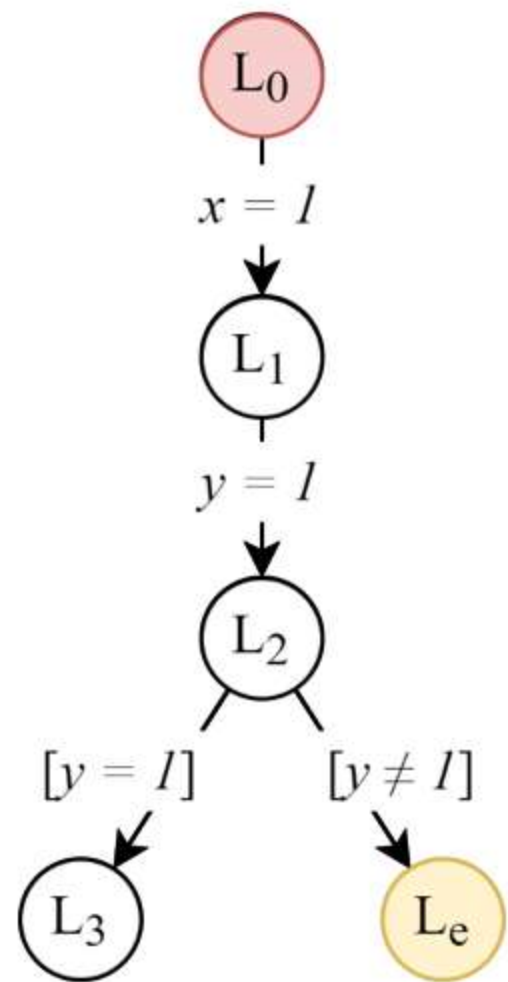
Fine-grained Statement Simplification



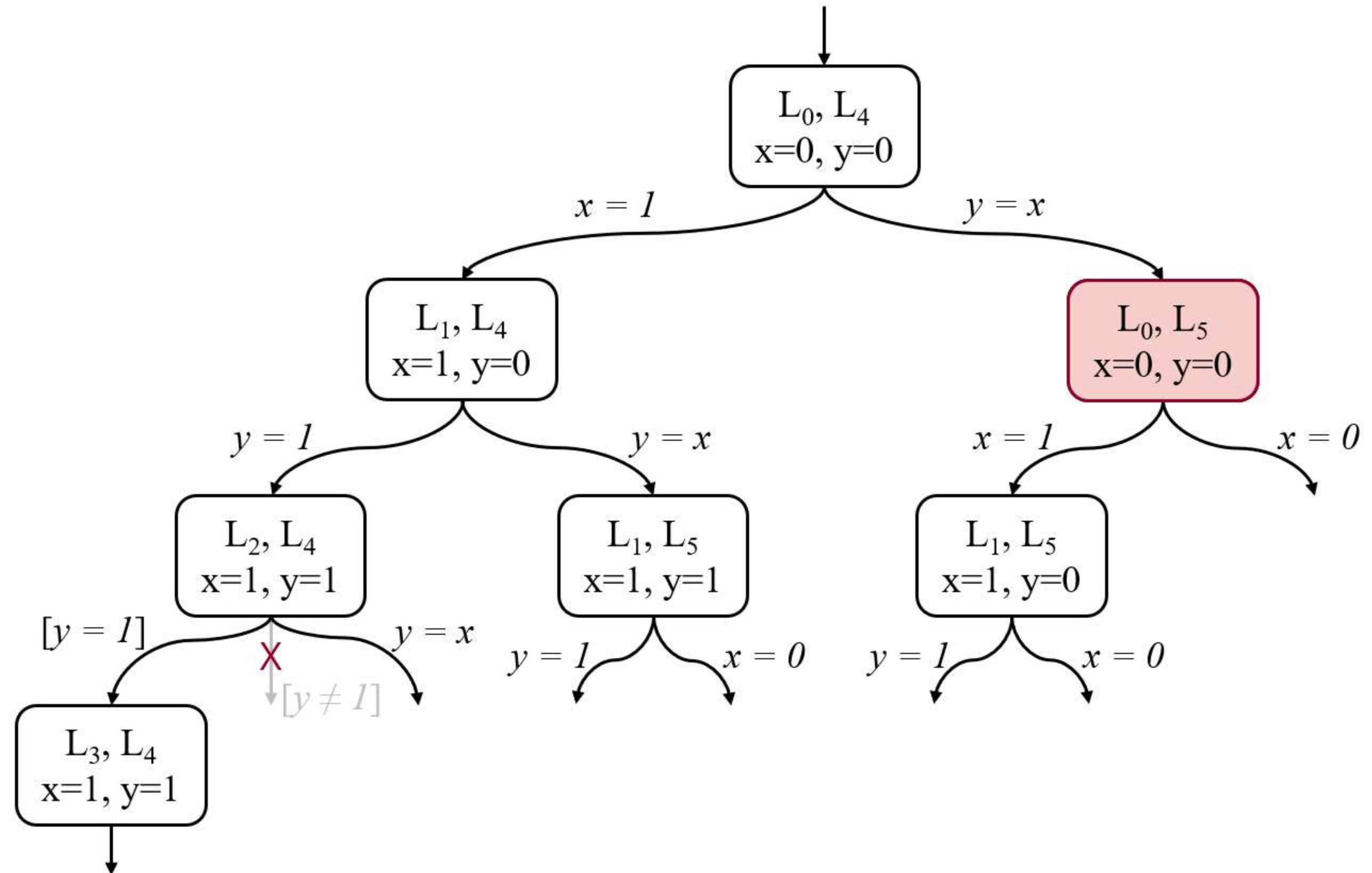
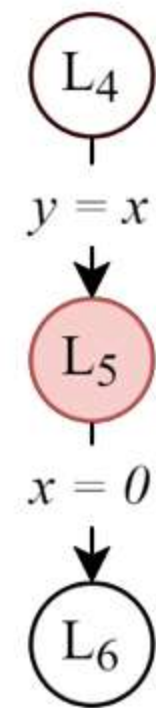
Fine-grained Statement Simplification



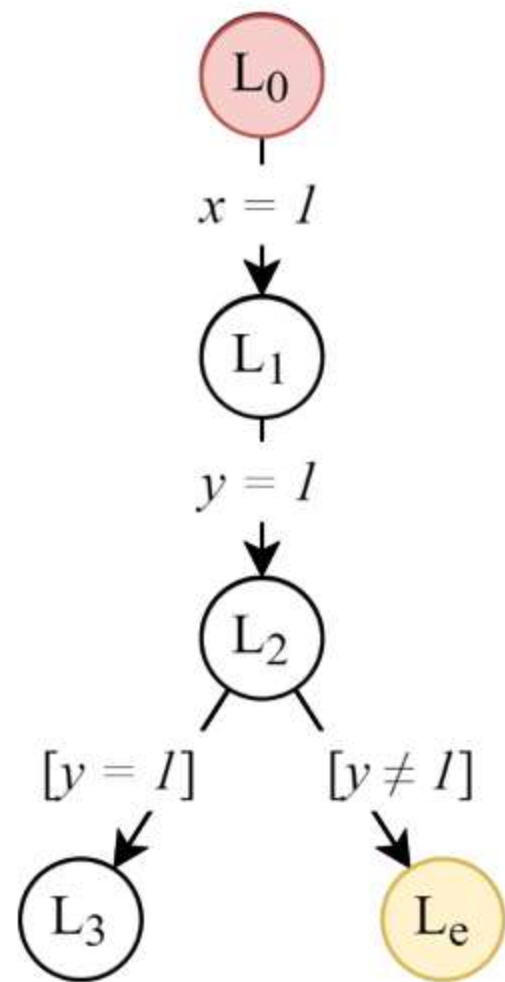
Fine-grained Statement Simplification



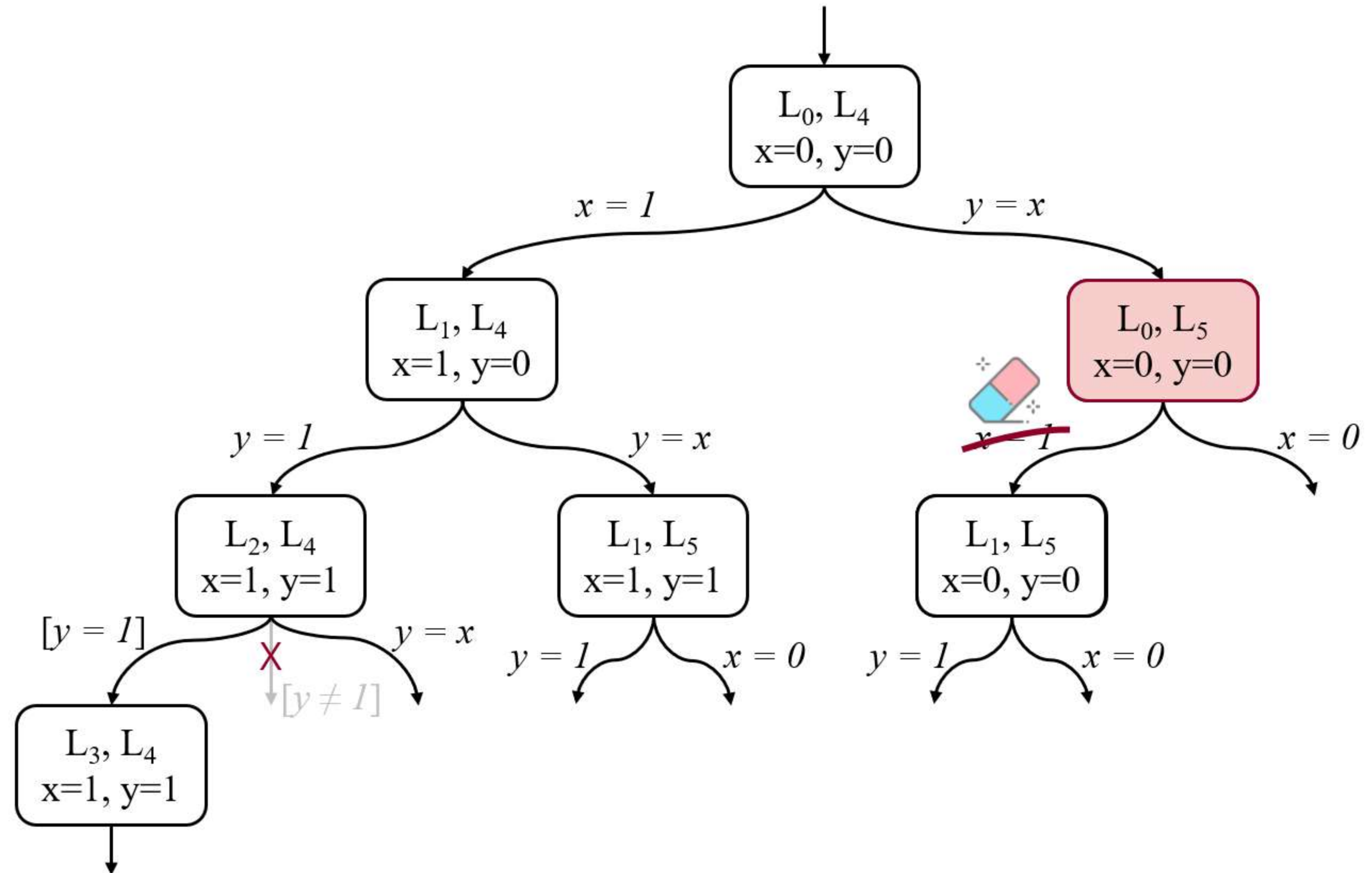
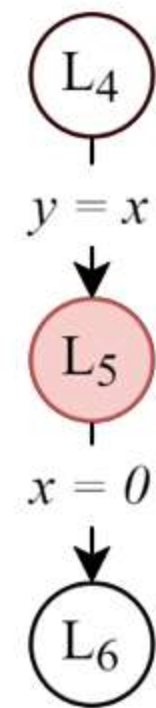
The value of x is not used after this state!



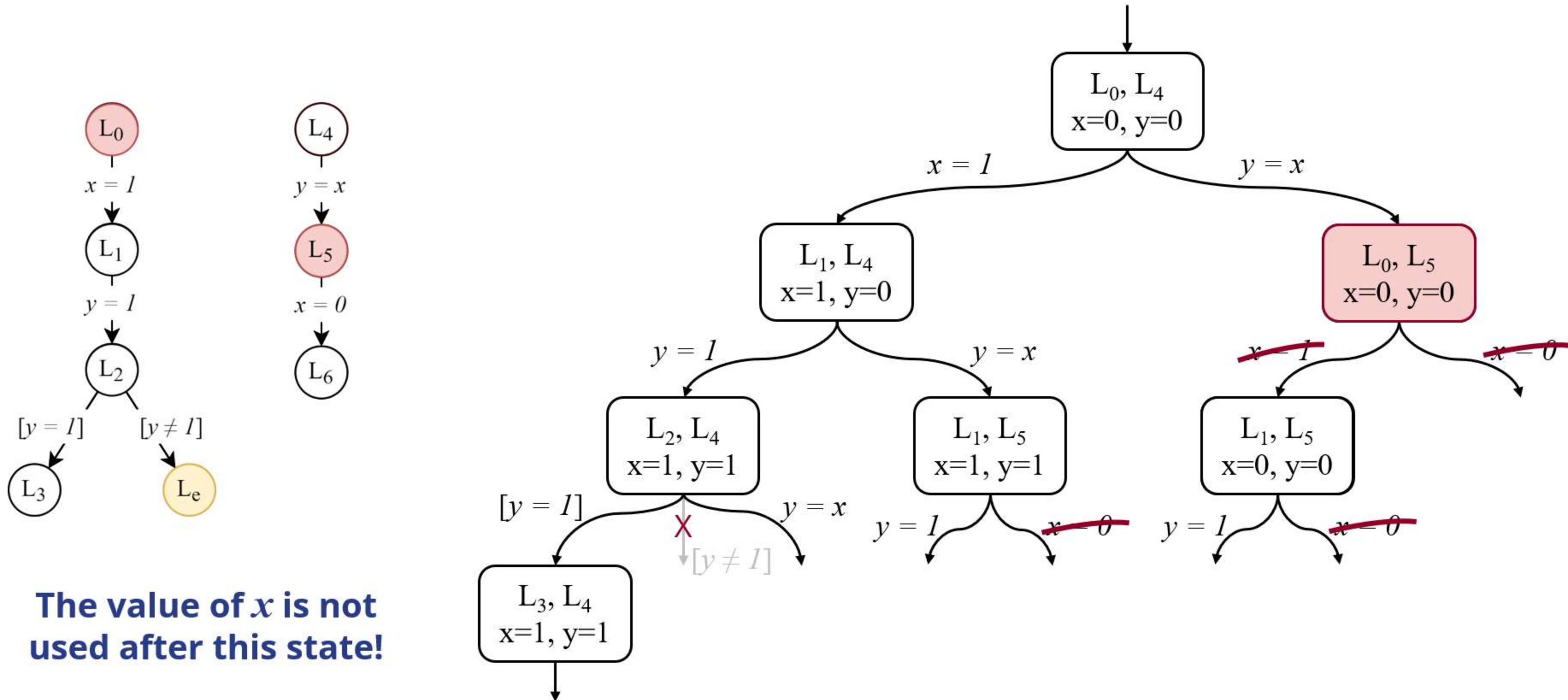
Fine-grained Statement Simplification



The value of x is not used after this state!

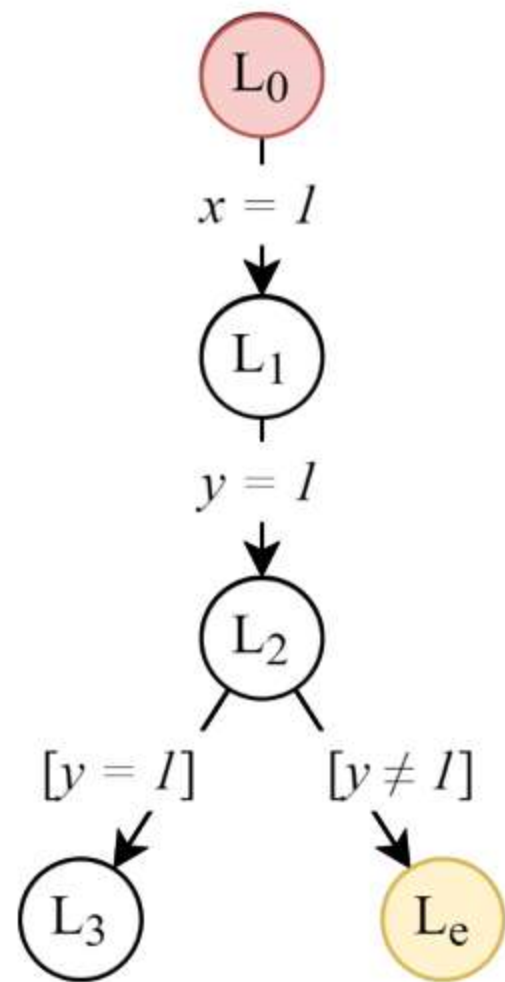


Fine-grained Statement Simplification

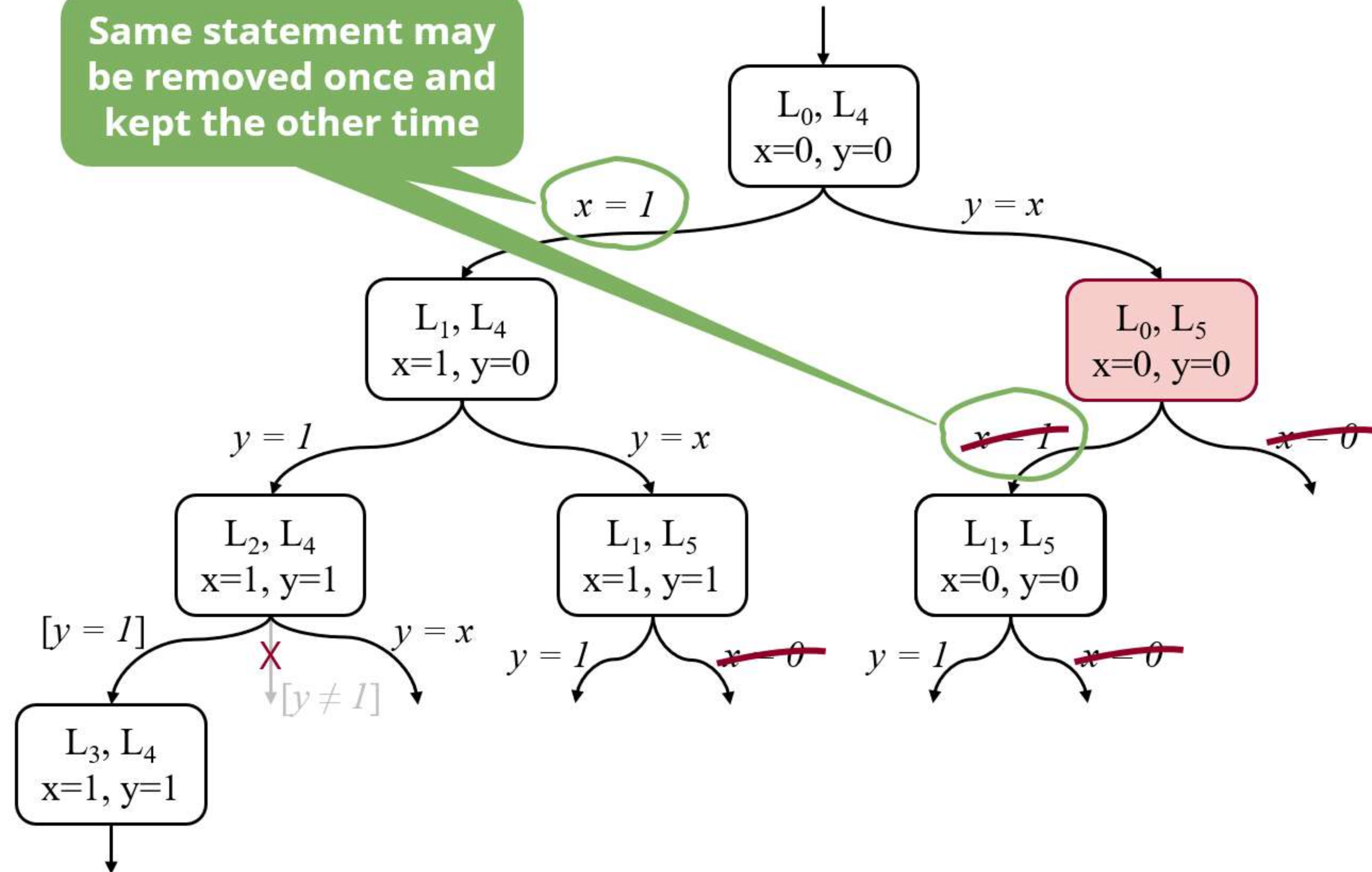


Fine-grained Statement Simplification

Same statement may
be removed once and
kept the other time



**The value of x is not
used after this state!**



Formalization

Formalization

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

Formalization

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

Abstract data-flow graph:

Formalization

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

Abstract data-flow graph:

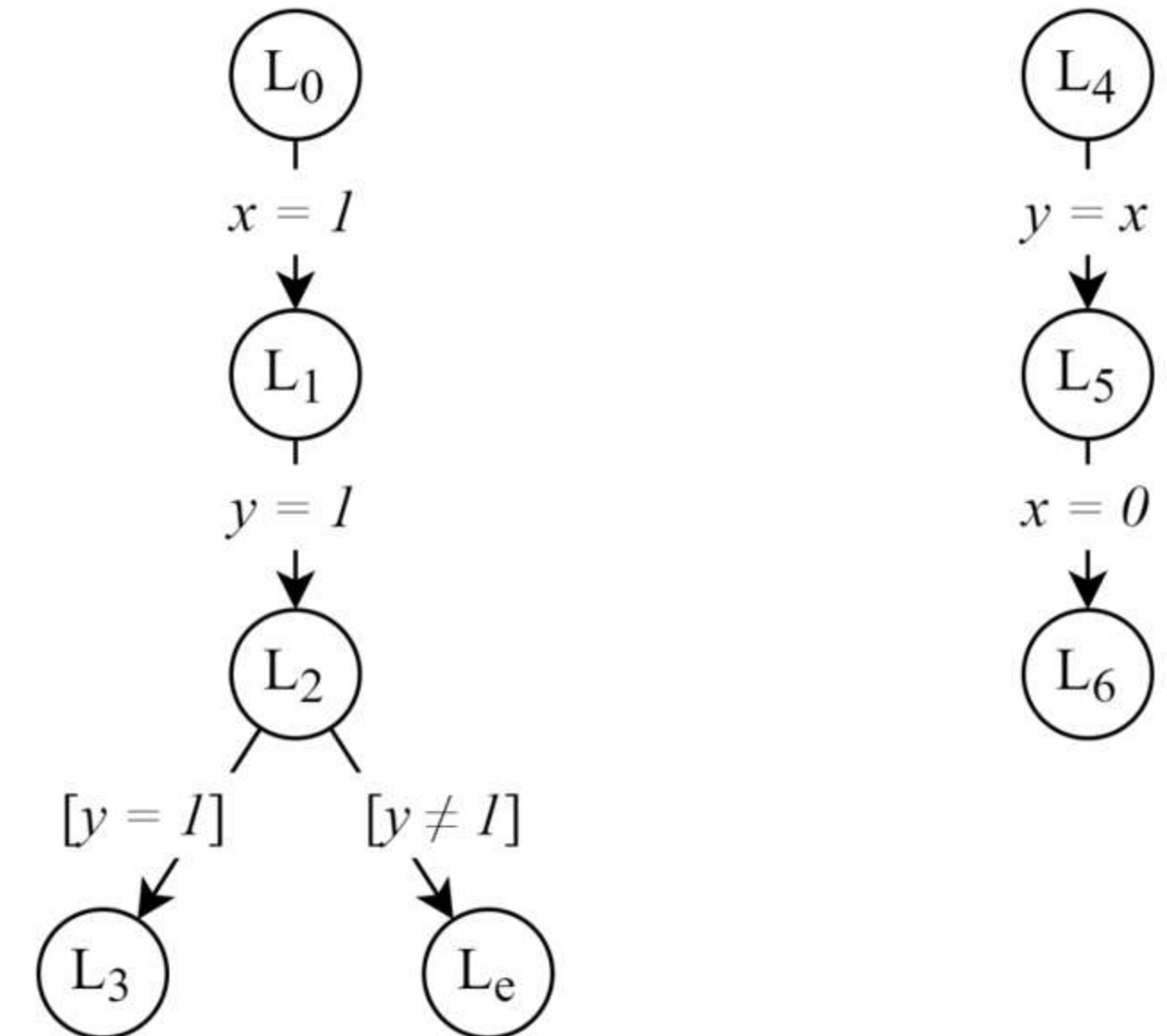
- **nodes**: program statements

Formalization

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

Abstract data-flow graph:

- **nodes**: program statements

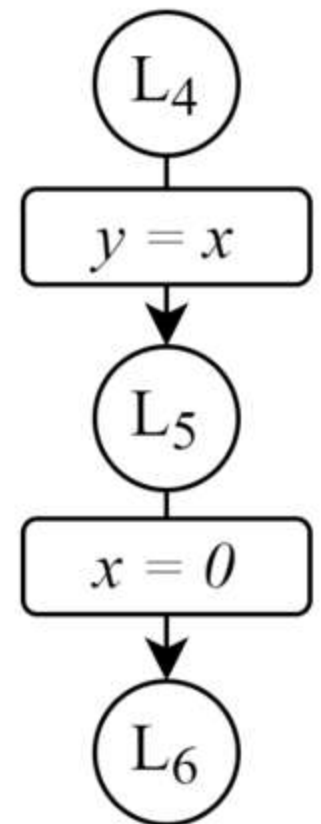
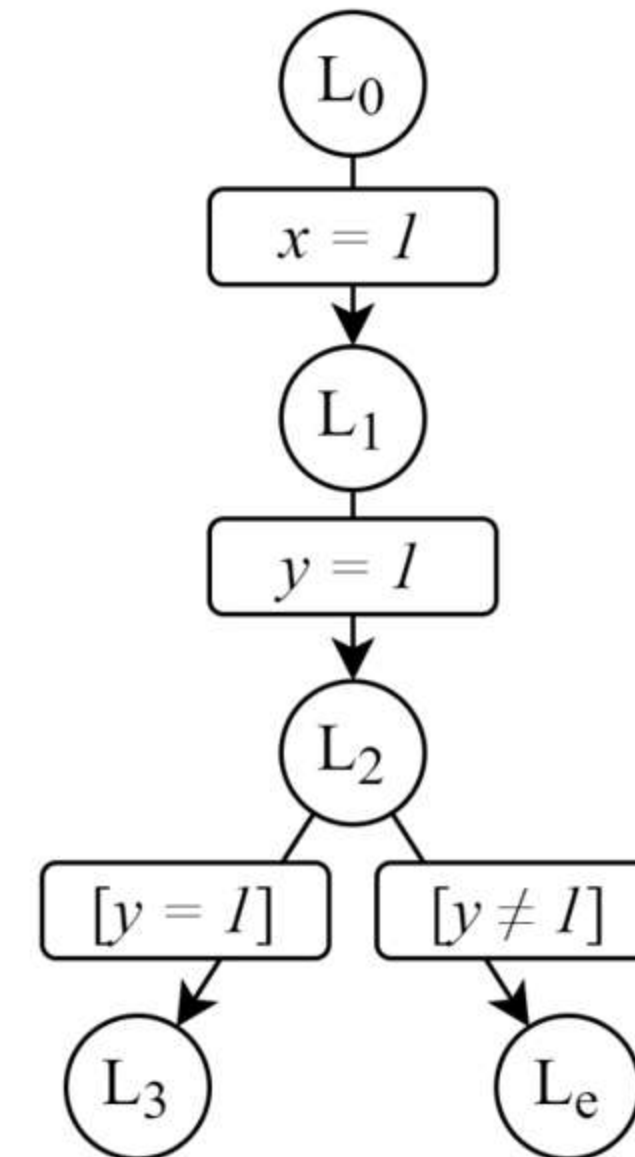


Formalization

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

Abstract data-flow graph:

- **nodes**: program statements

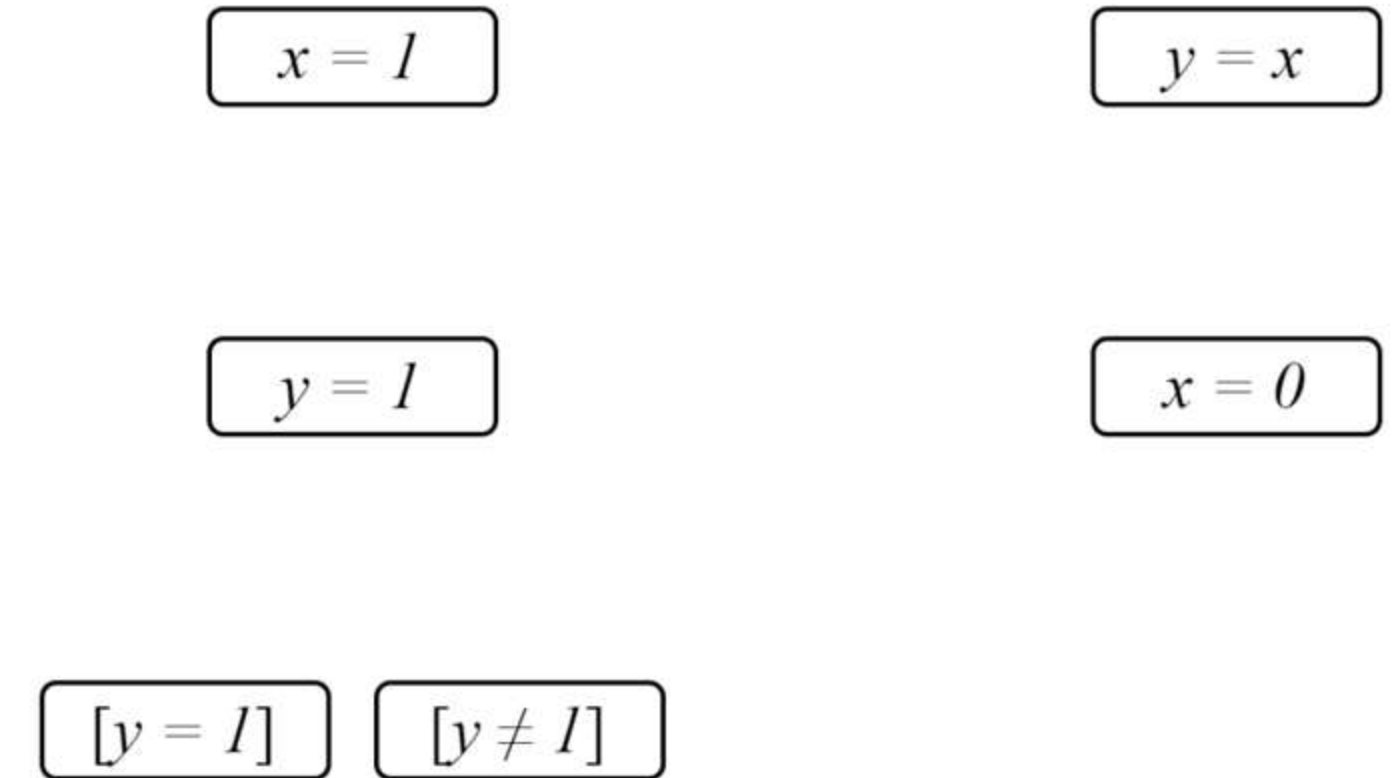


Formalization

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

Abstract data-flow graph:

- **nodes**: program statements

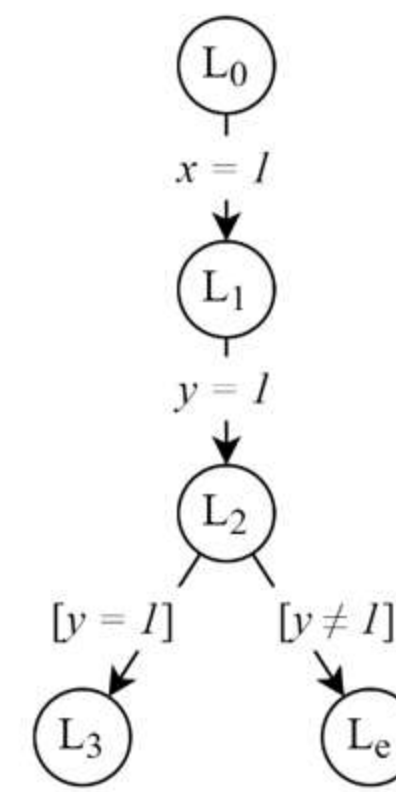


Formalization

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

Abstract data-flow graph:

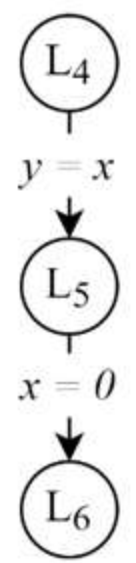
- **nodes**: program statements



$x = 1$

$y = 1$

$[y = 1]$ $[y \neq 1]$



$y = x$

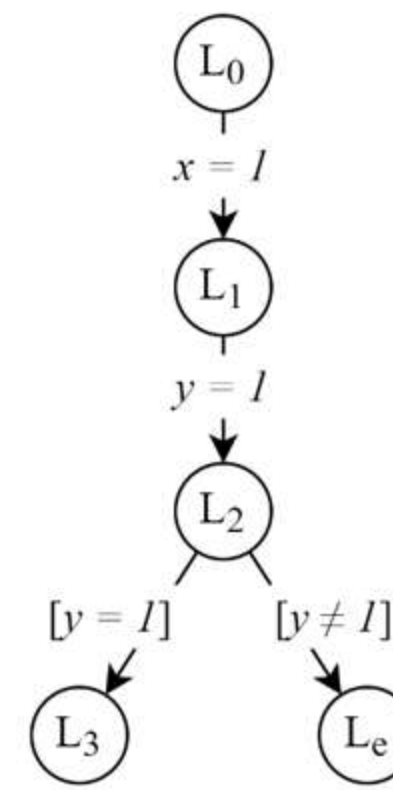
$x = 0$

Formalization

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

Abstract data-flow graph:

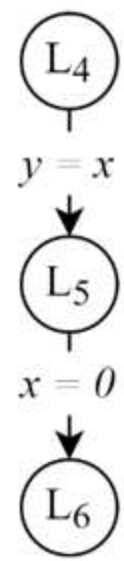
- **nodes**: program statements
- (directed) **edges**: if a statement may use the result of another one
„observation relation”



$x = 1$

$y = 1$

$[y = 1]$ $[y \neq 1]$



$y = x$

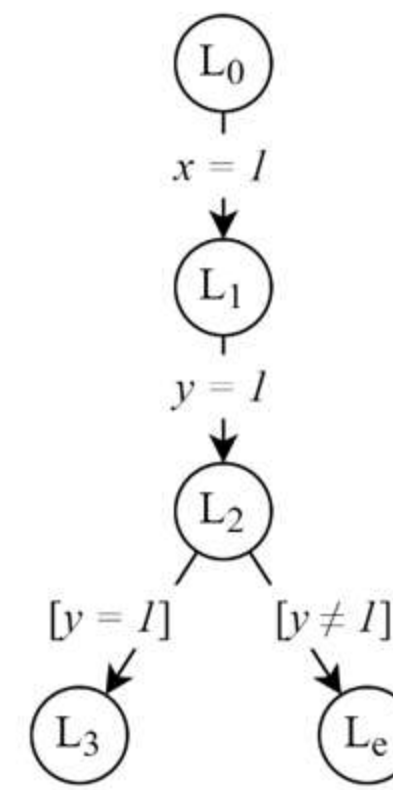
$x = 0$

Formalization

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

Abstract data-flow graph:

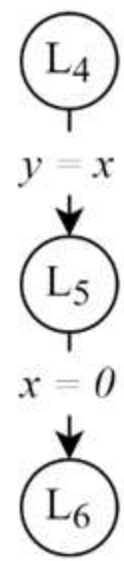
- **nodes**: program statements
- (directed) **edges**: if a statement may use the result of another one
„observation relation”
 - **direct** („in-process”) edge
 - **indirect** („inter-process”) edge



$x = l$

$y = l$

$[y = l]$ $[y \neq l]$



$y = x$

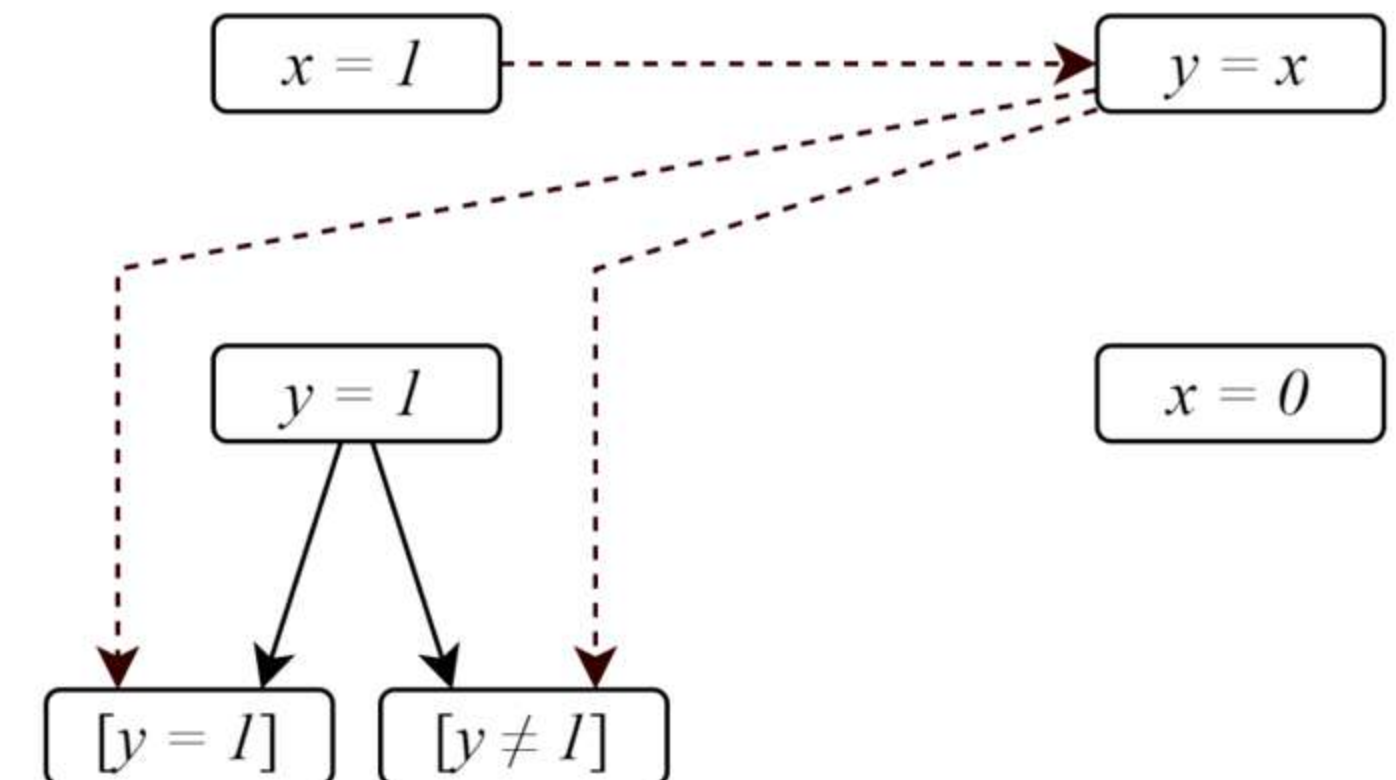
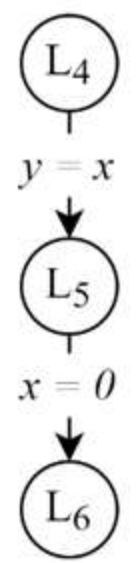
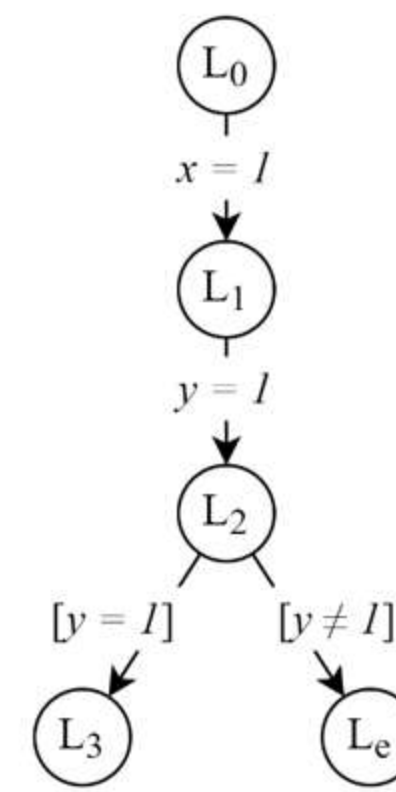
$x = 0$

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 $\dashrightarrow$

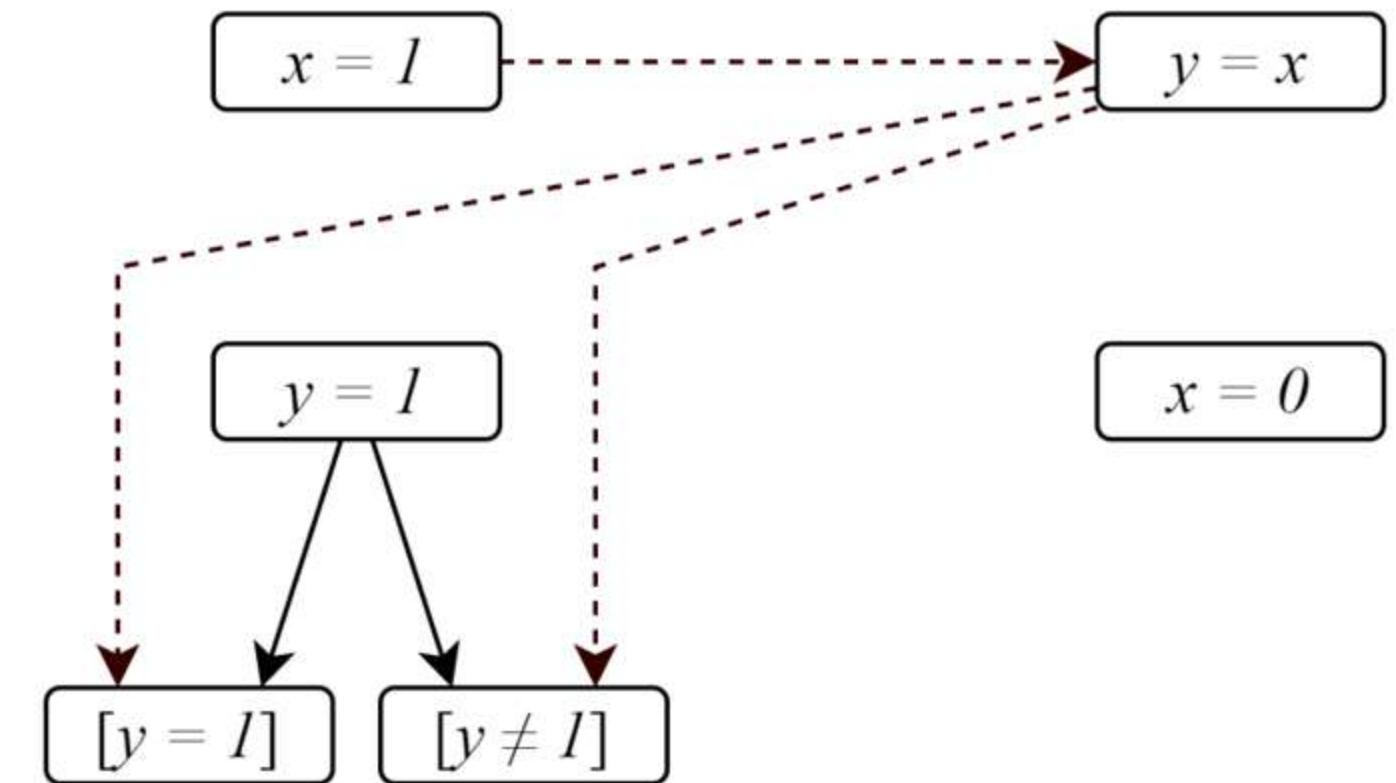
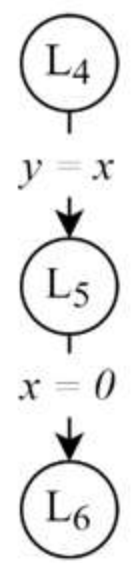
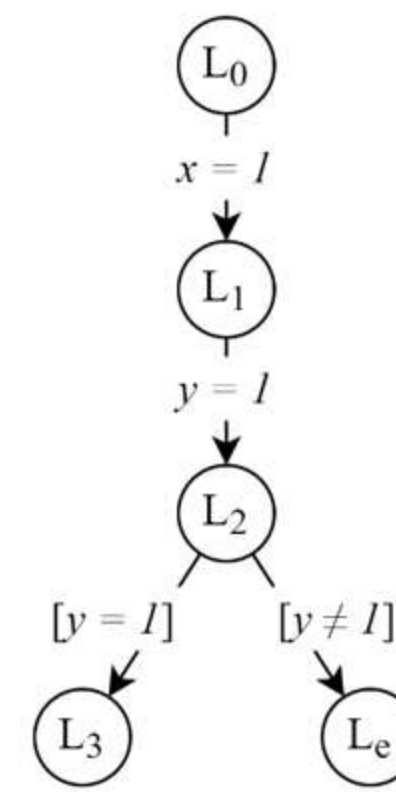


Formalization

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

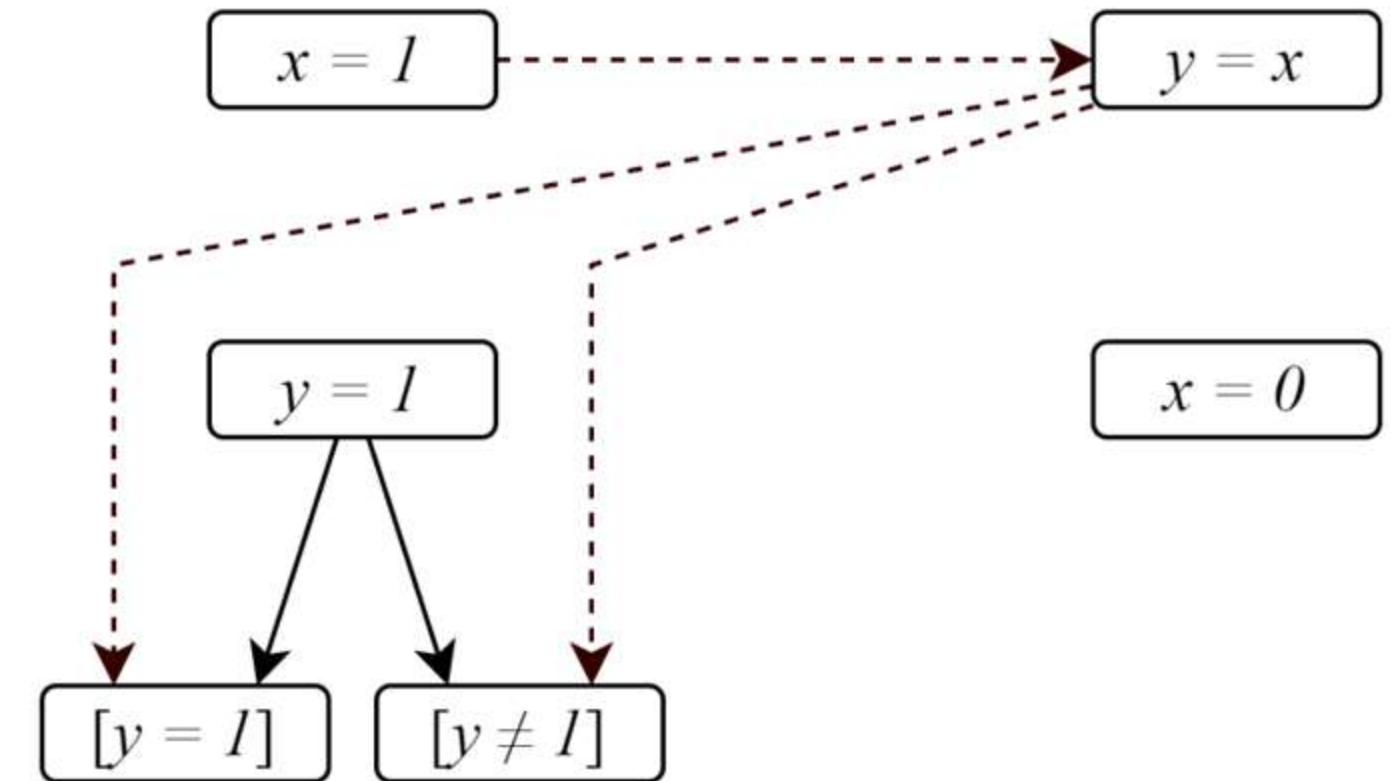
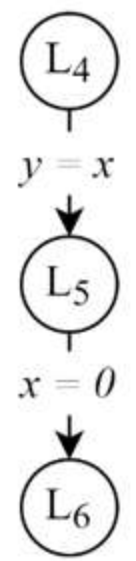
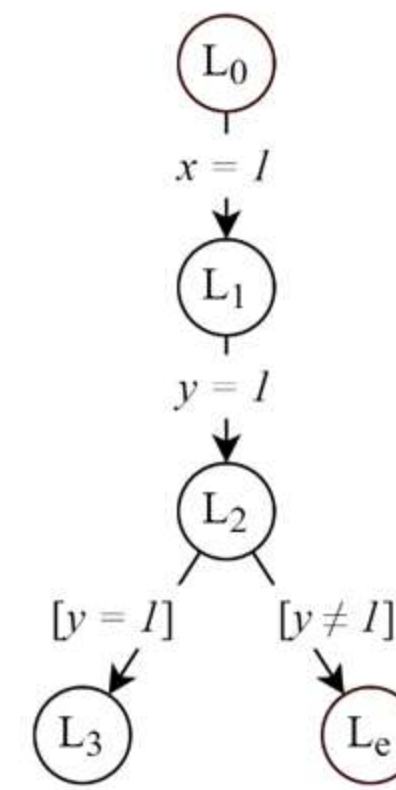
Abstract data-flow graph:

- **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 $\dashrightarrow$
- only consider data-flow visible in the abstraction



Formalization

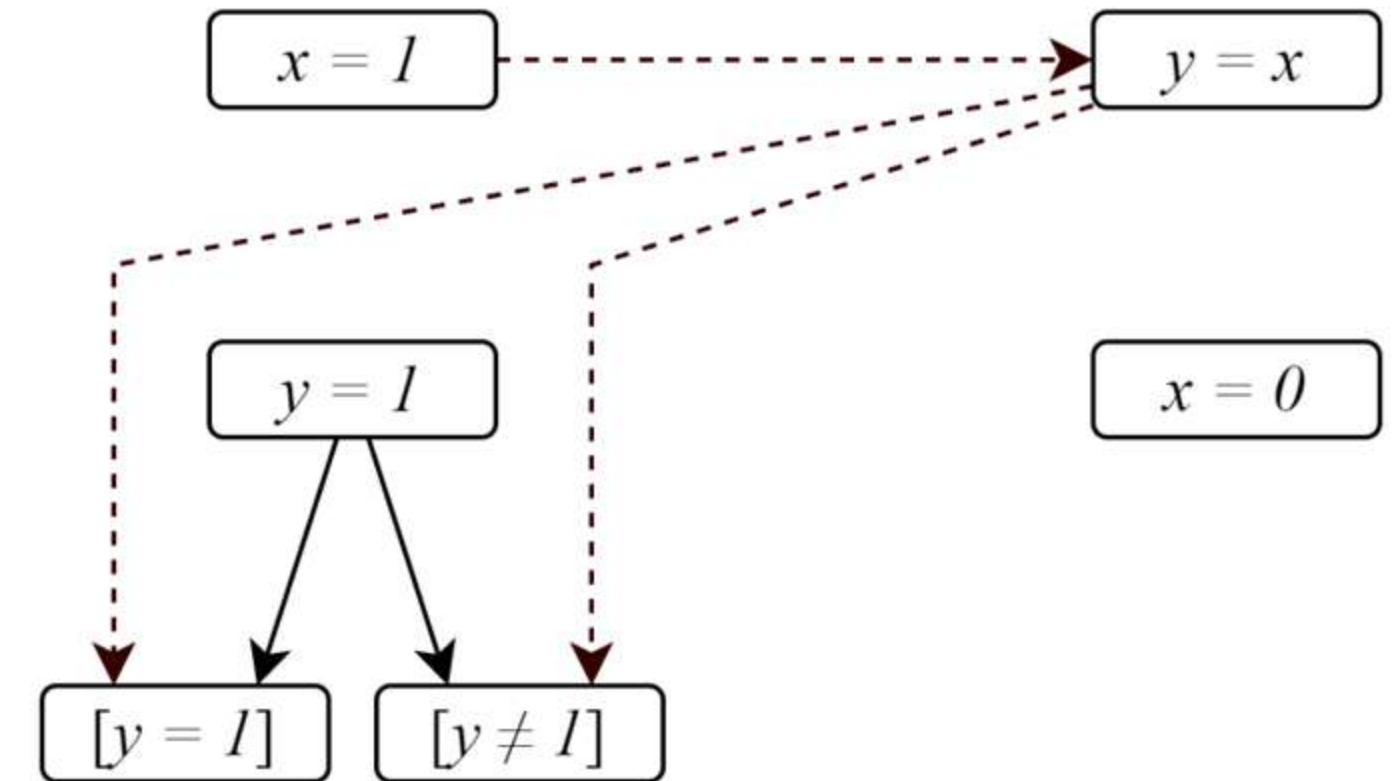
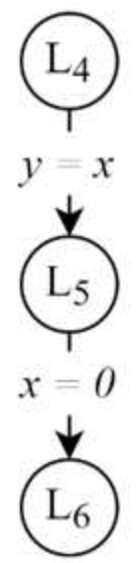
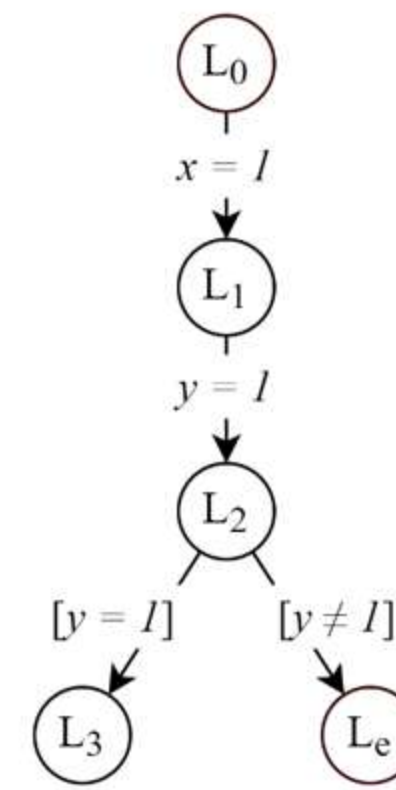
Question: is the **current statement relevant** for proving the property? Remove, if not.



Formalization

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

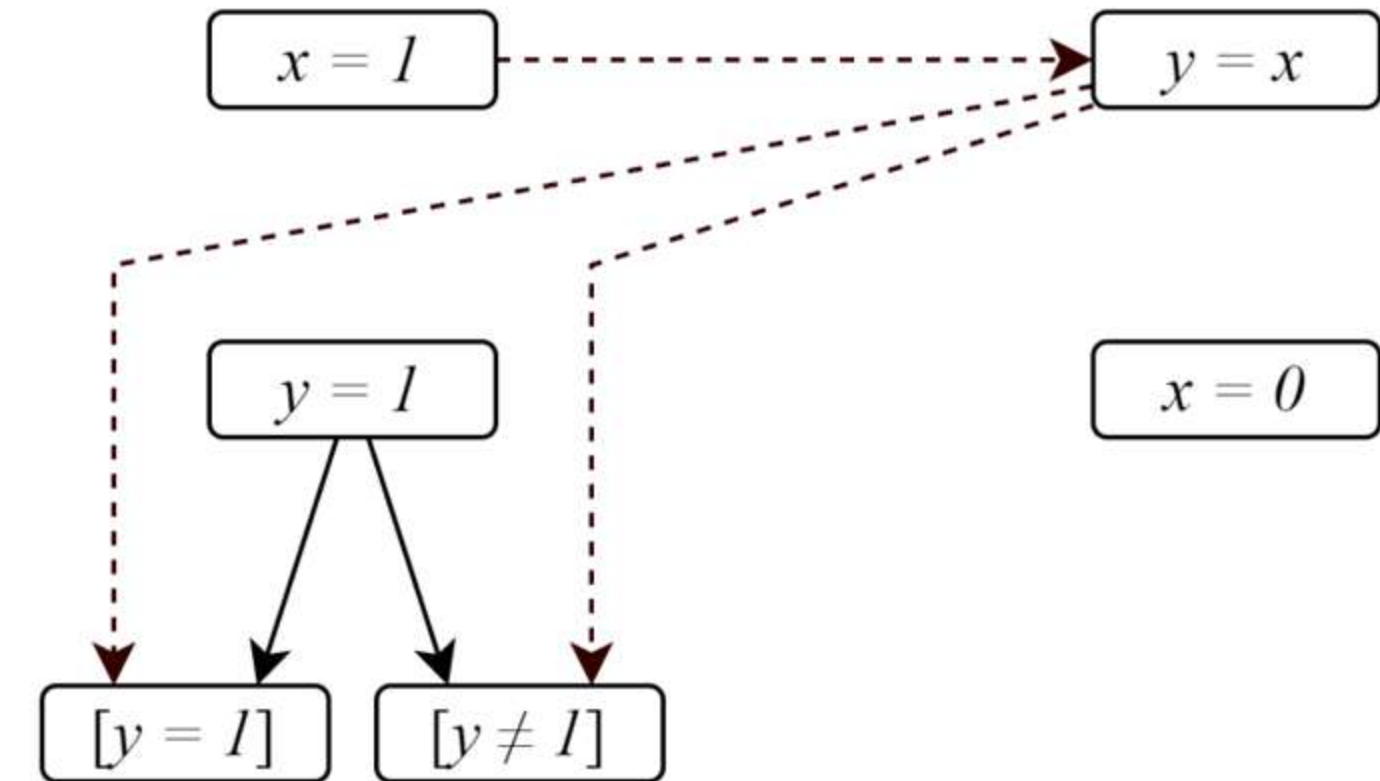
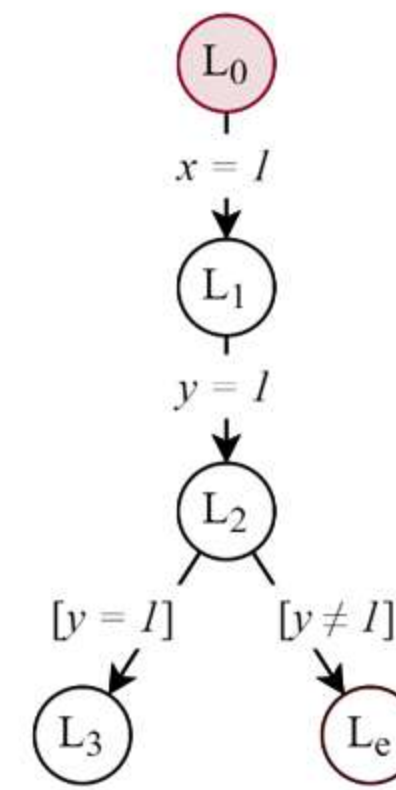
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process



Formalization

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

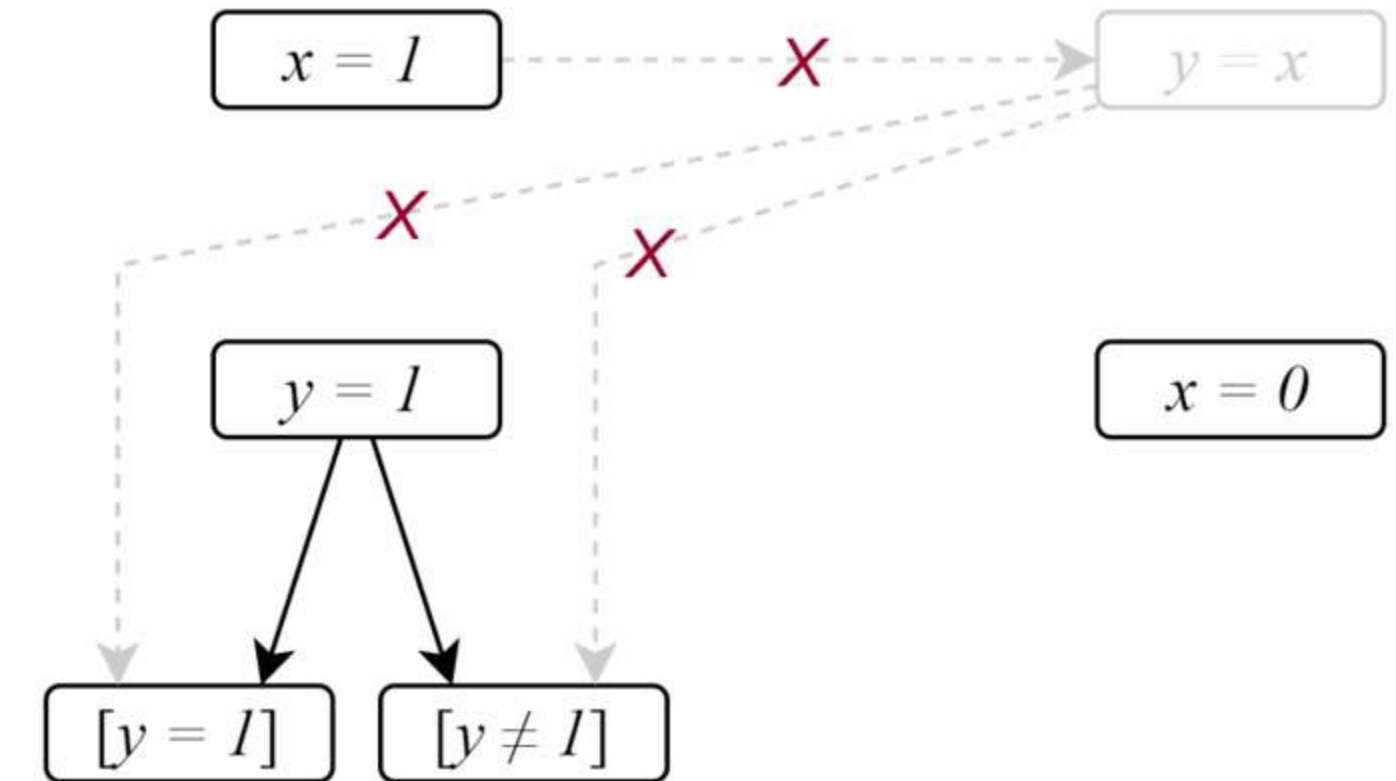
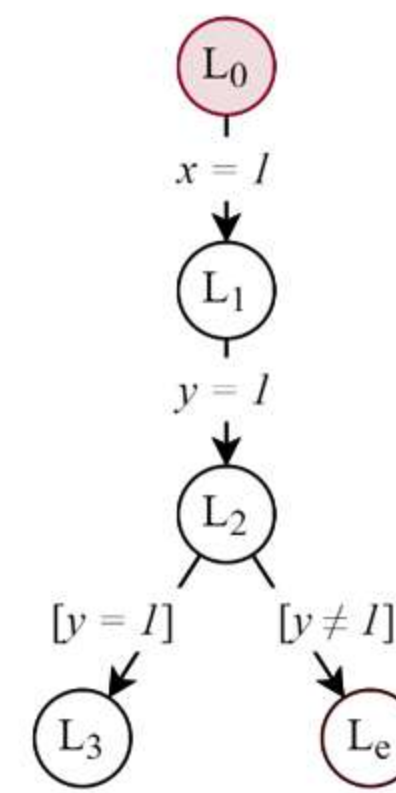
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process



Formalization

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

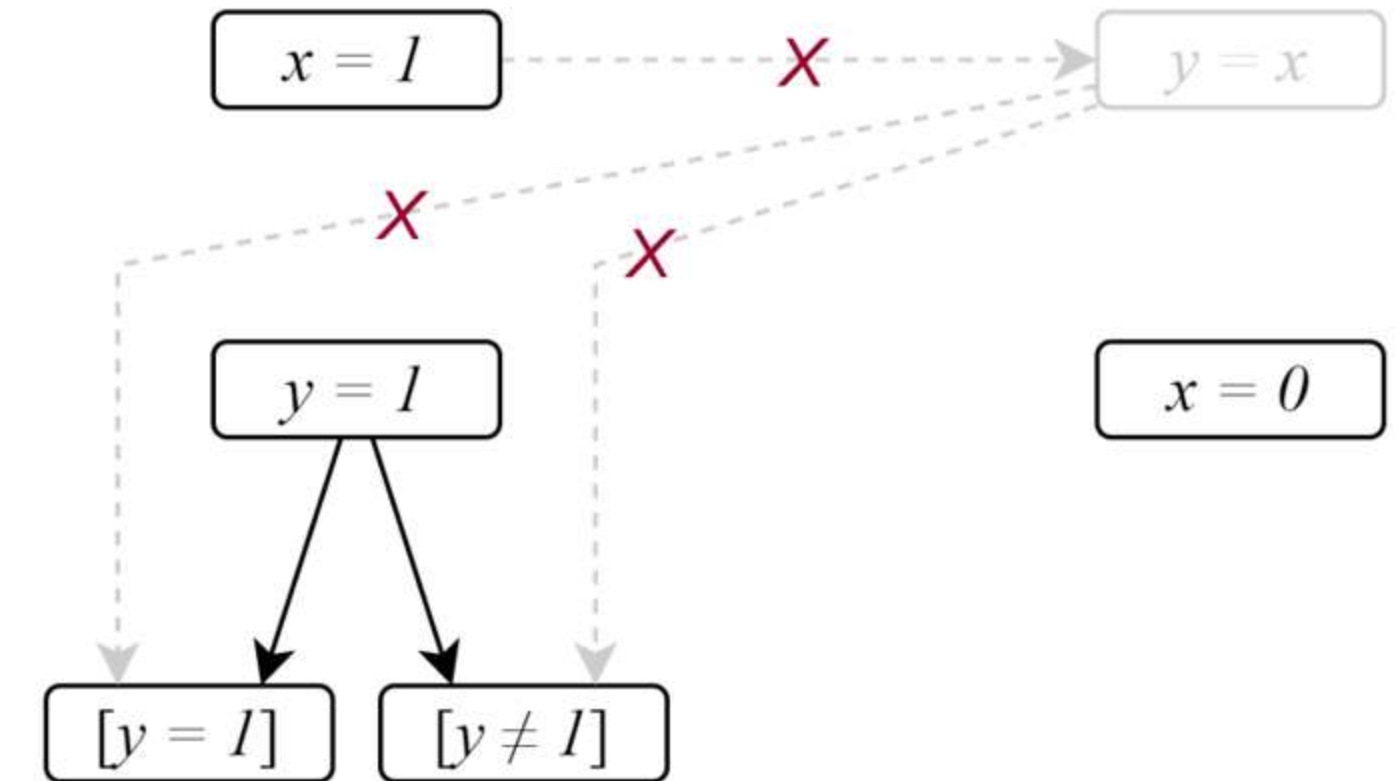
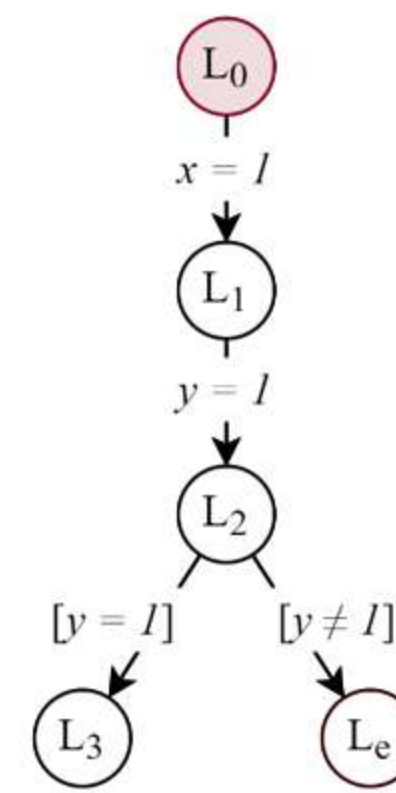
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process



Formalization

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

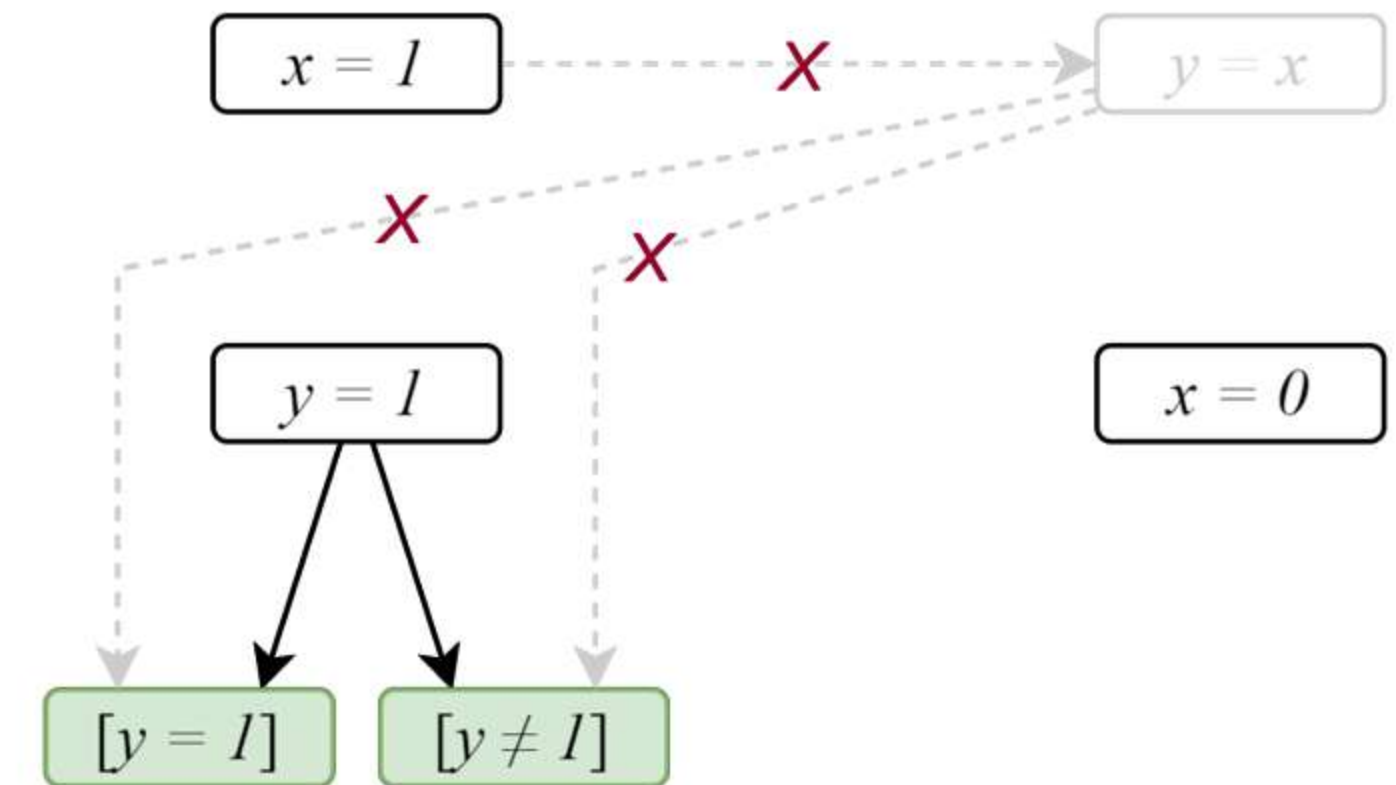
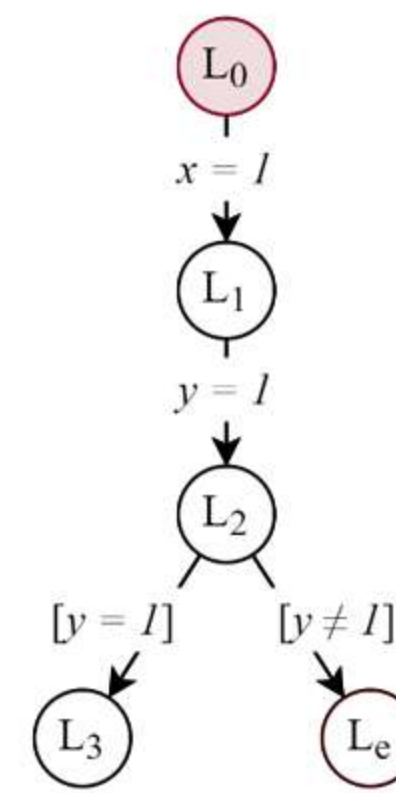
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process
- **Real observers**: conditional statements
 - reachability of error locations depends on them
 - **transitive observation** by a real observer: **path to a real observer** in the data-flow graph



Formalization

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

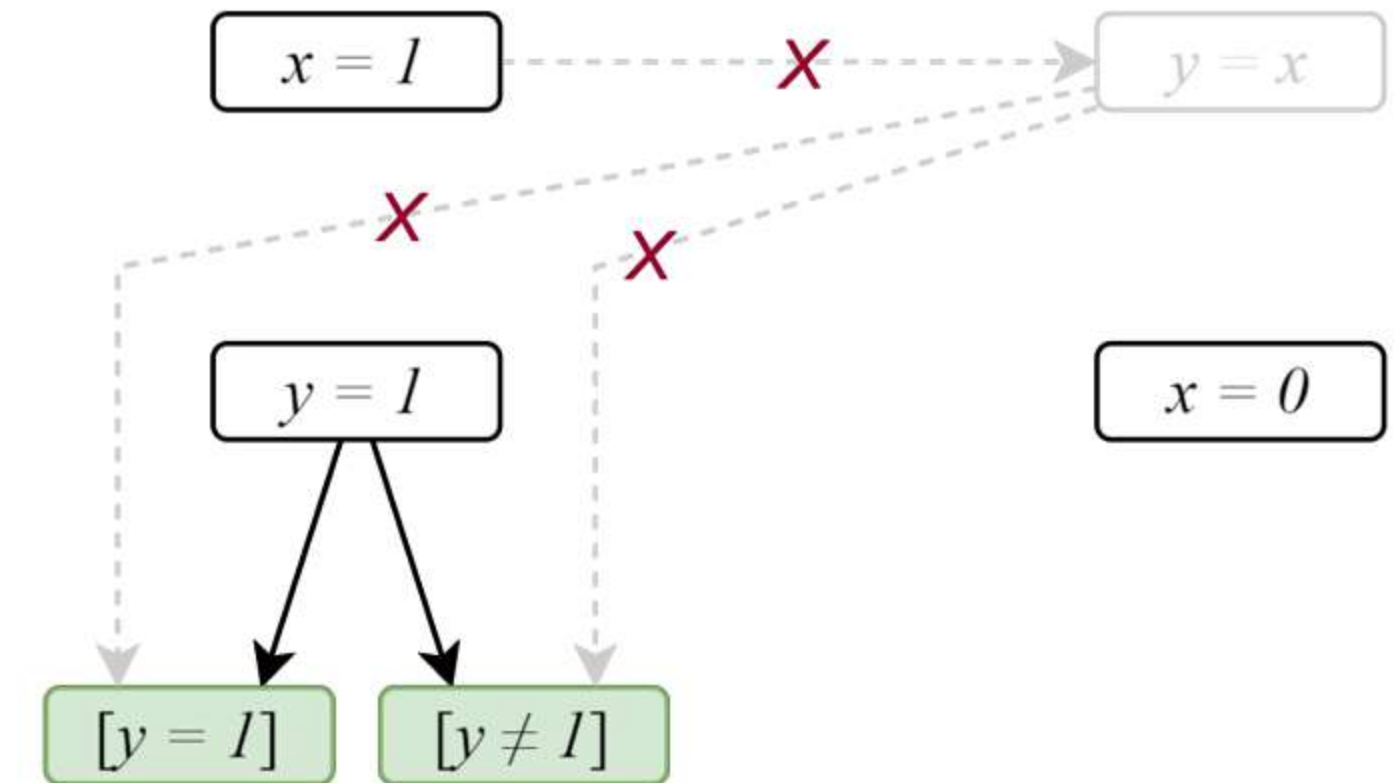
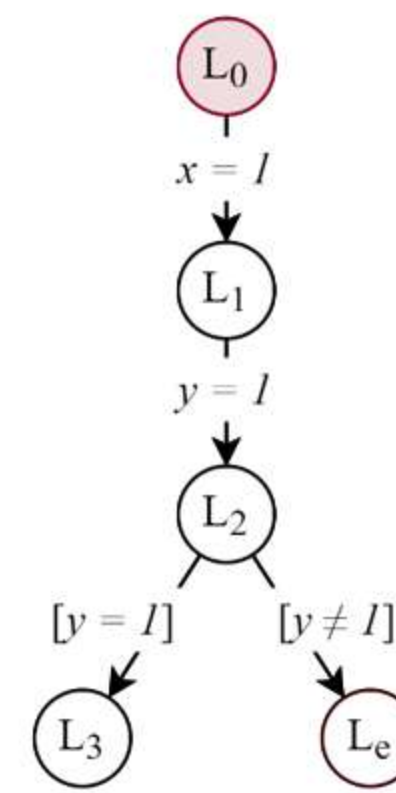
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process
- **Real observers**: conditional statements
 - reachability of error locations depends on them
 - **transitive observation** by a real observer: **path to a real observer** in the data-flow graph



Formalization

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

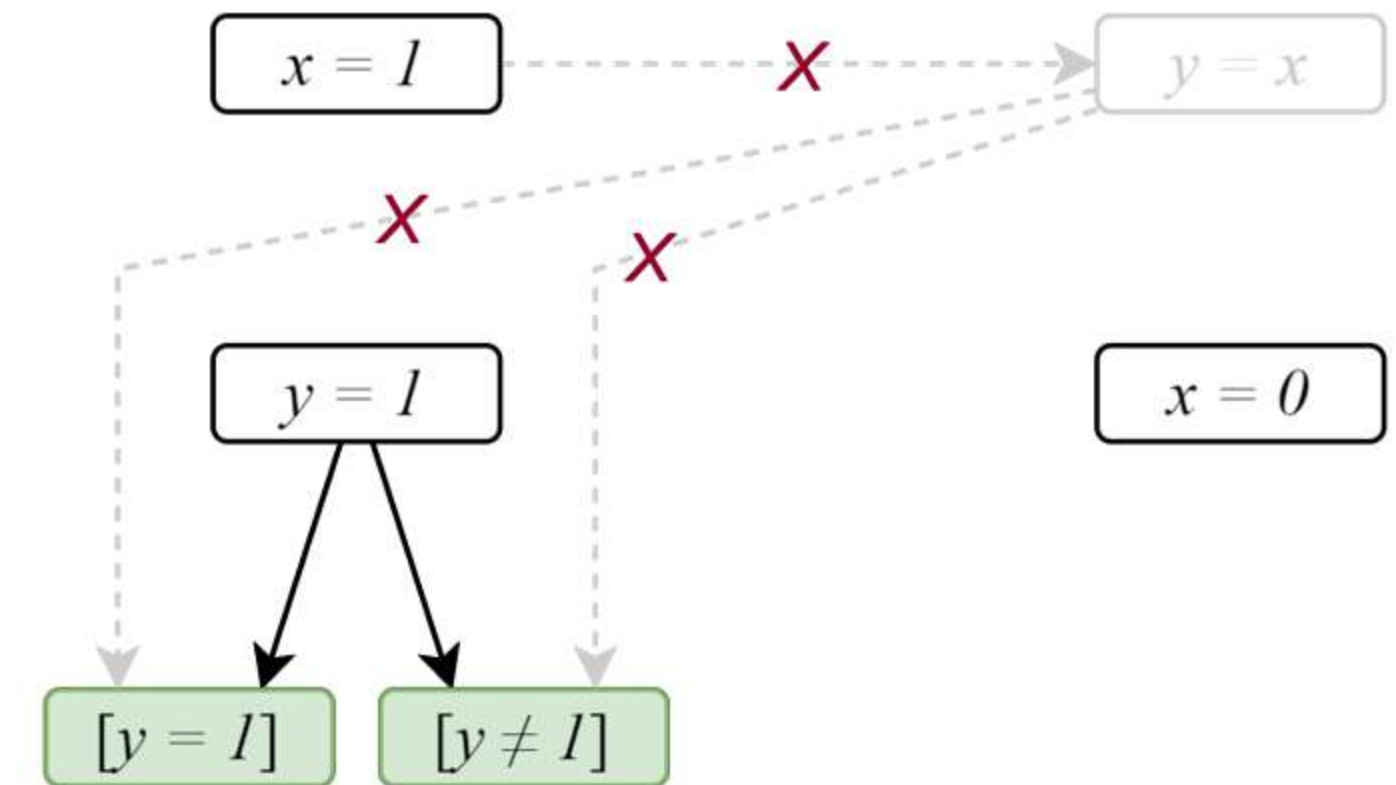
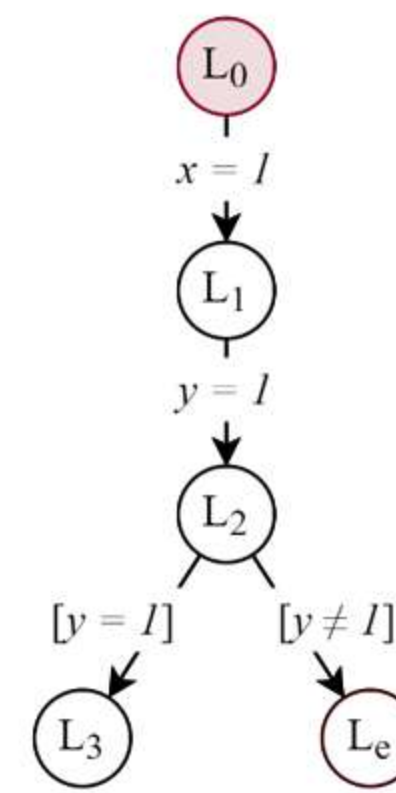
- Context-sensitivity:
 - a data-flow graph edge e is **enabled in a state** if both ends of e **can be executed in the future** by some process
- **Real observers**: conditional statements
 - reachability of error locations depends on them
 - **transitive observation** by a real observer:
path to a real observer in the data-flow graph
- If **no real observer** of the current statement:
it **can be removed** (not evaluated)



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 removed** (not evaluated)

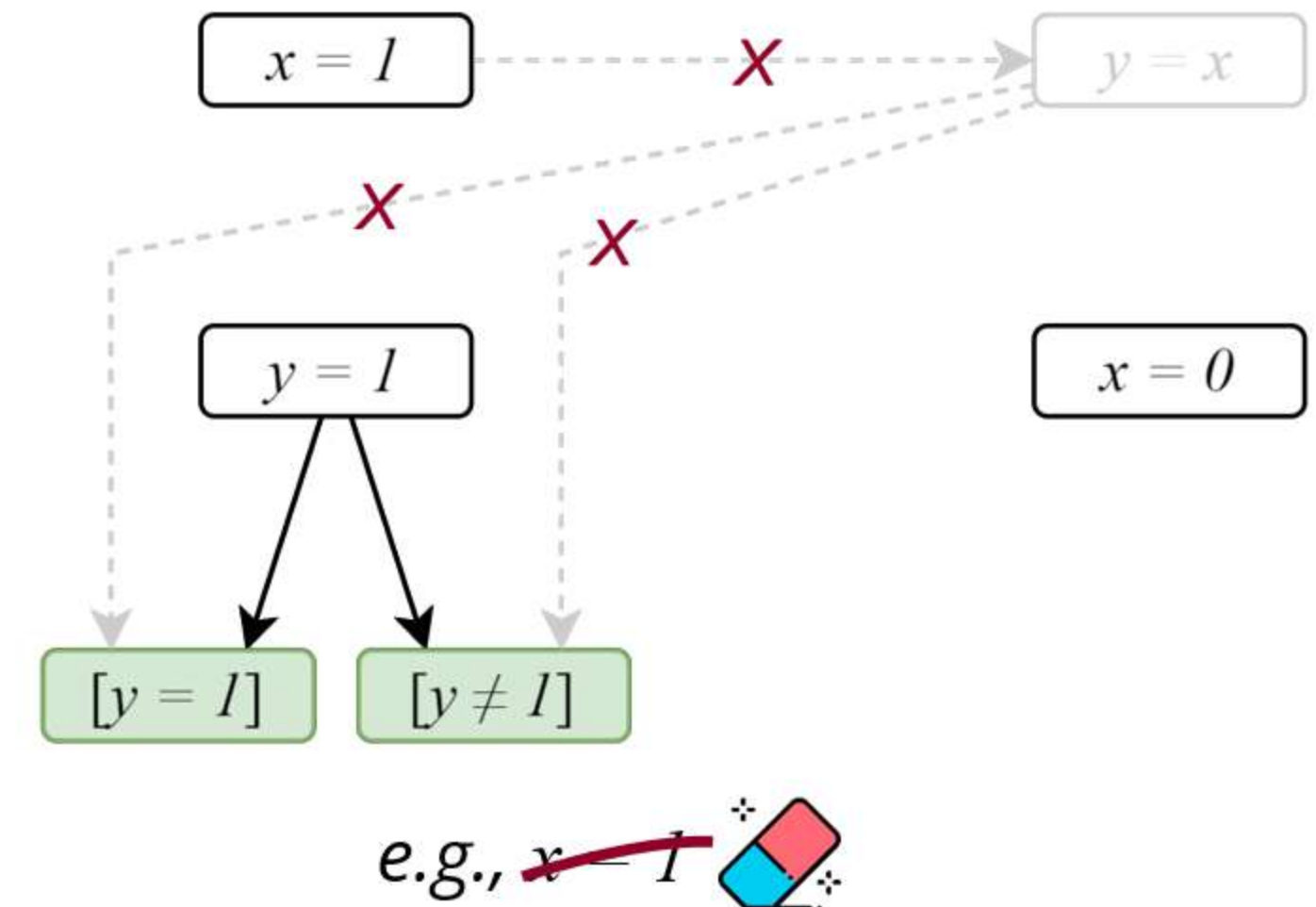
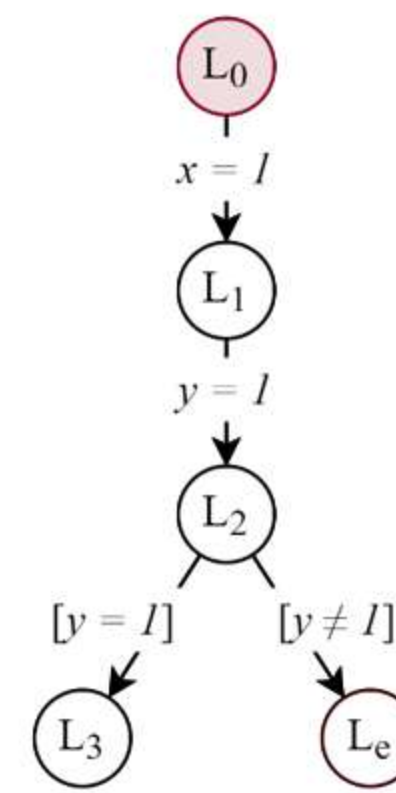


e.g., $x = 1$

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 removed** (not evaluated)



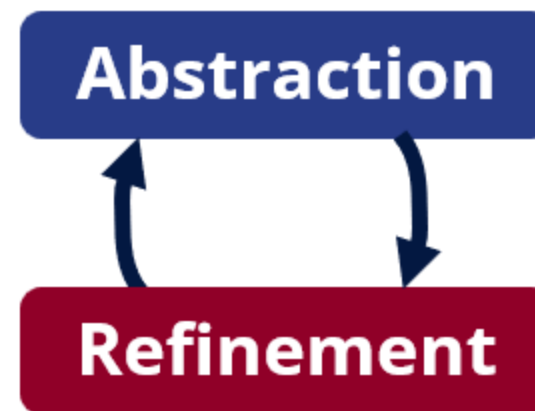
Algorithmic Integration – CEGAR

Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

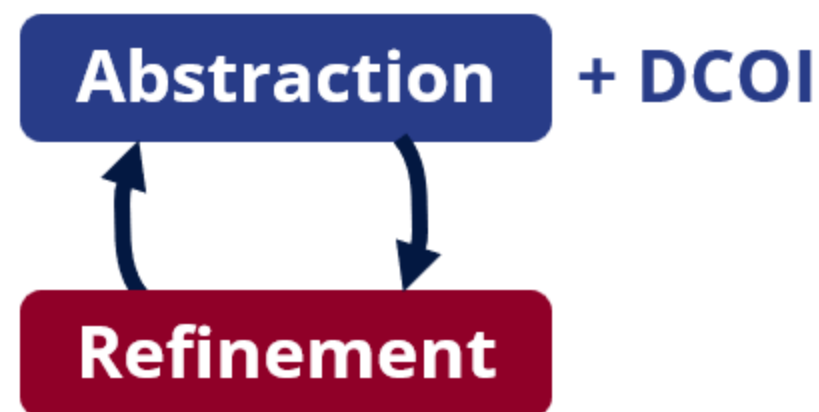
Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):
 - **Iterative** abstraction-based model checking algorithm



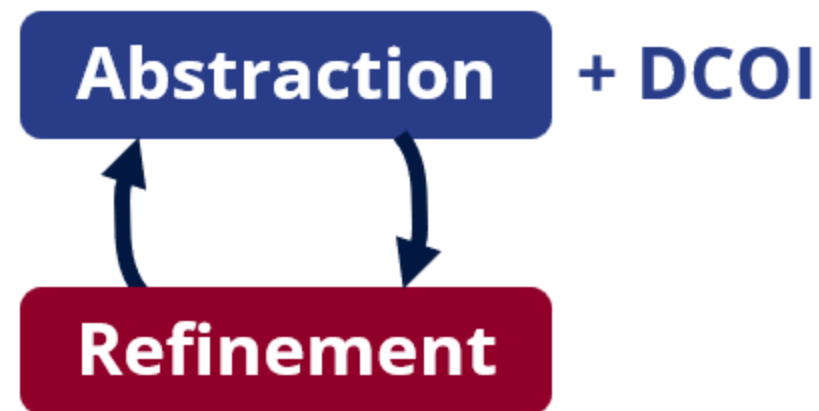
Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):
 - **Iterative** abstraction-based model checking algorithm
 - Abstraction phase: **abstract state space exploration**
+ the proposed **on-the-fly COI** (DCOI)



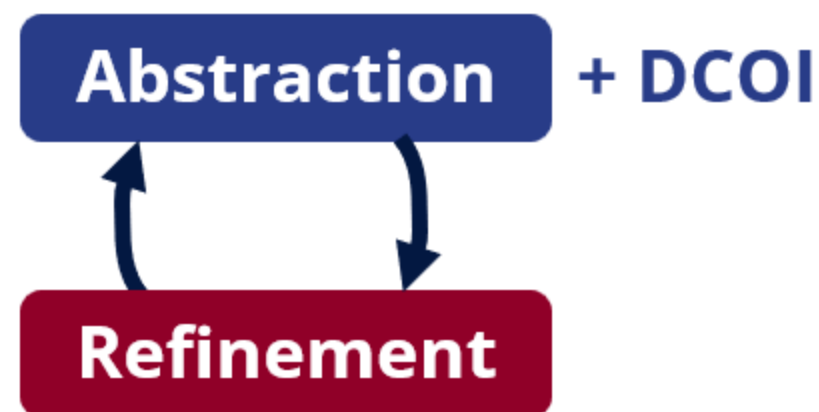
Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):
 - **Iterative** abstraction-based model checking algorithm
 - Abstraction phase: **abstract state space exploration**
+ the proposed **on-the-fly COI** (DCOI)
 - If no counterexample: program is safe



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):
 - **Iterative** abstraction-based model checking algorithm
 - Abstraction phase: **abstract state space exploration**
+ the proposed **on-the-fly COI** (DCOI)
 - If no counterexample: program is safe
 - Refinement phase: **check counterexample, refine abstraction**

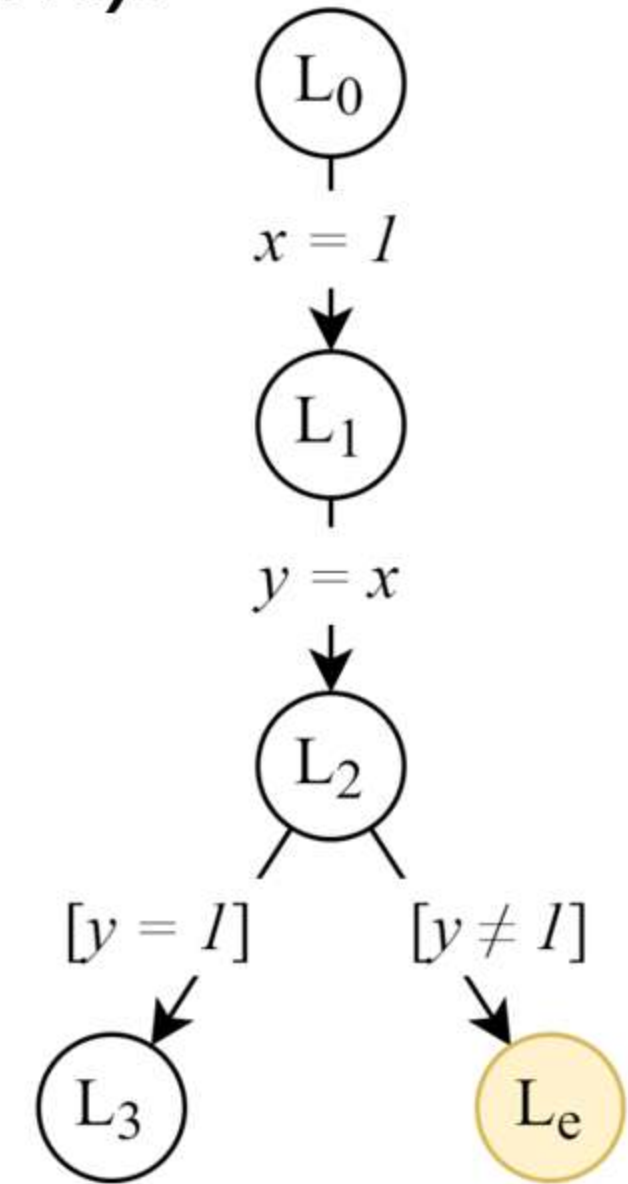


Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

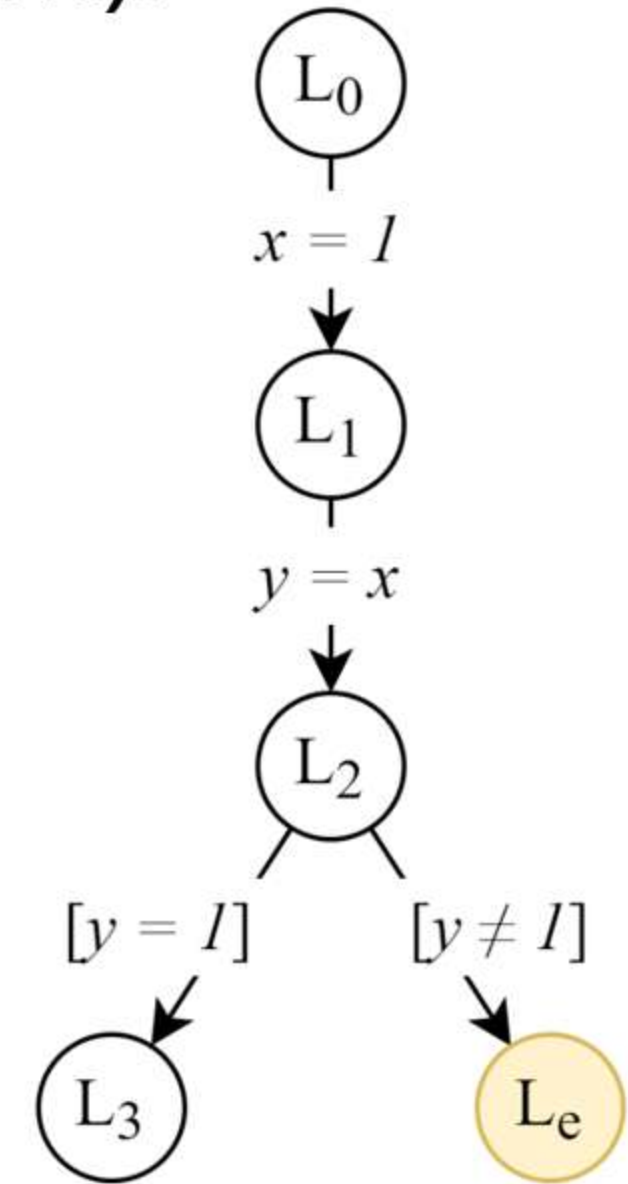
Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):



Algorithmic Integration – CEGAR

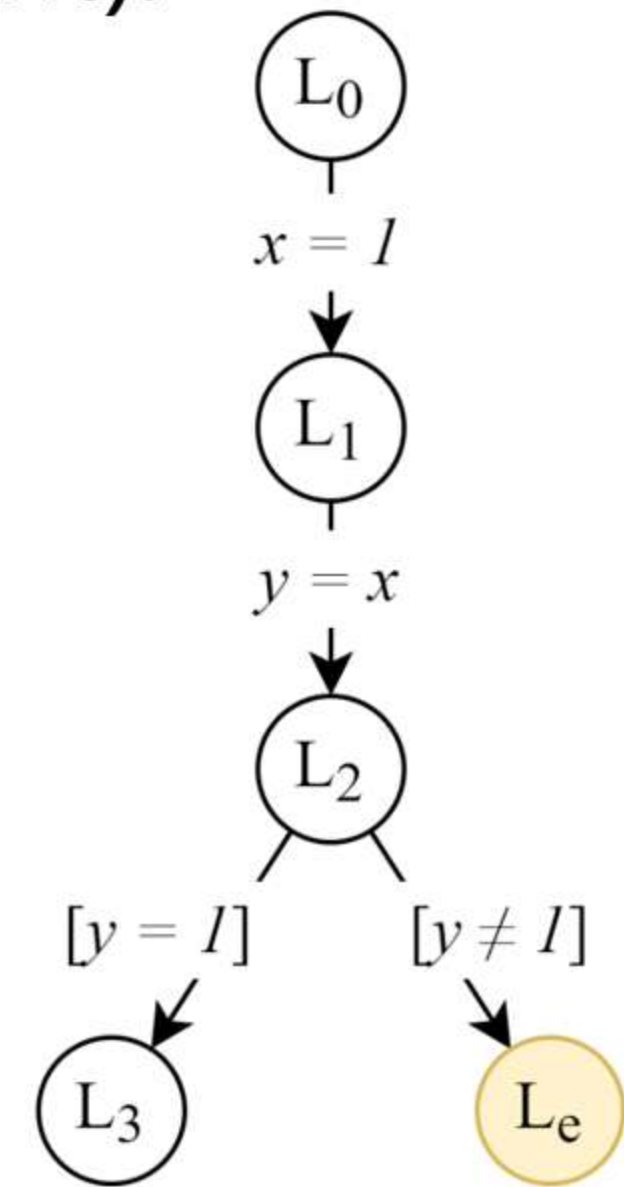
- CEGAR (Counterexample-Guided Abstraction Refinement):
 - E.g., abstract precision: only x is tracked
Counterexample: $(x = 1, y = x, [y \neq 1])$



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):
 - E.g., abstract precision: only x is tracked
Counterexample: $(x = 1, y = x, [y \neq 1])$

SPURIOUS



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

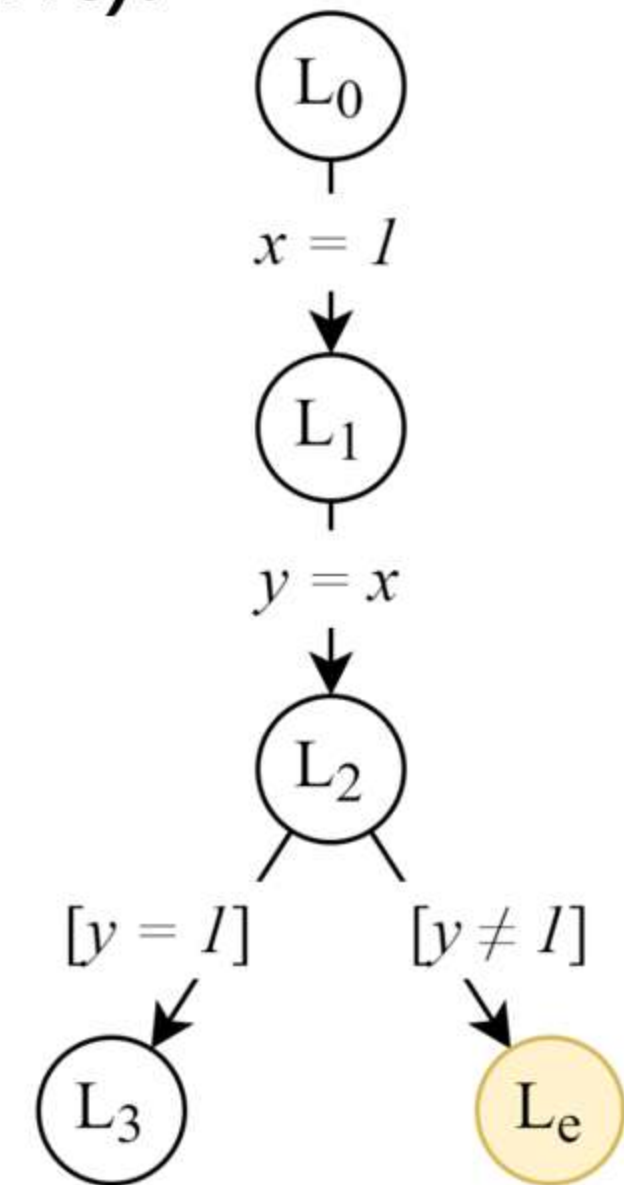
- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

SPURIOUS

- With our algorithm:

$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

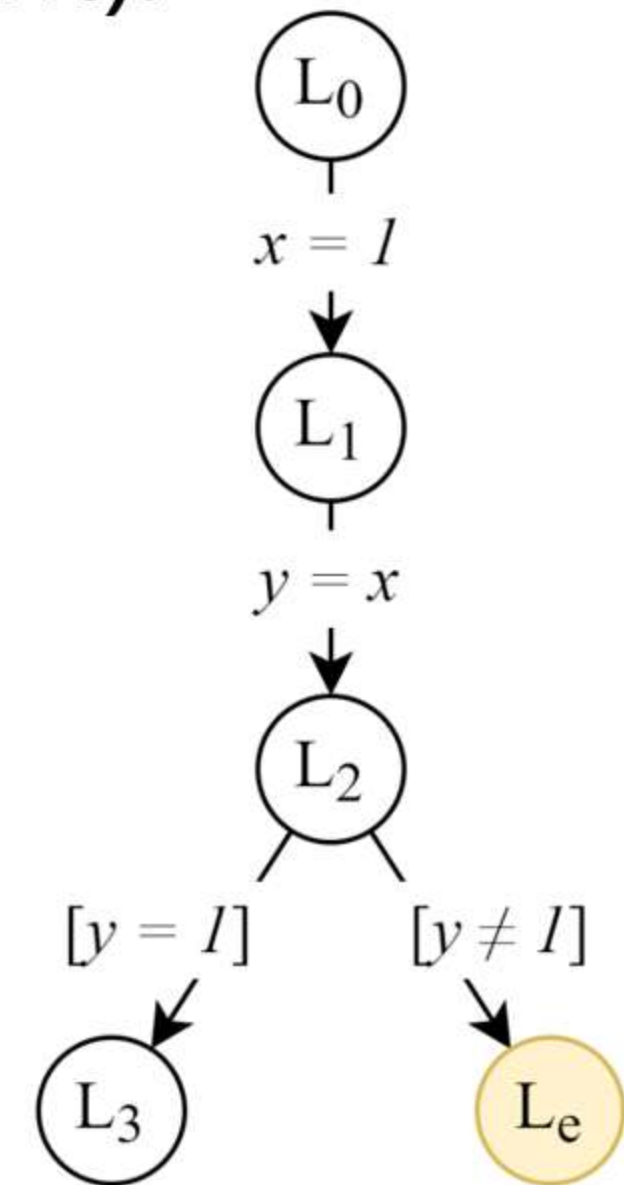
Counterexample: $(x = 1, y = x, [y \neq 1])$

- With our algorithm:

$\left. \begin{matrix} x=1 \\ y=x \end{matrix} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

SPURIOUS

NOT SPURIOUS



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

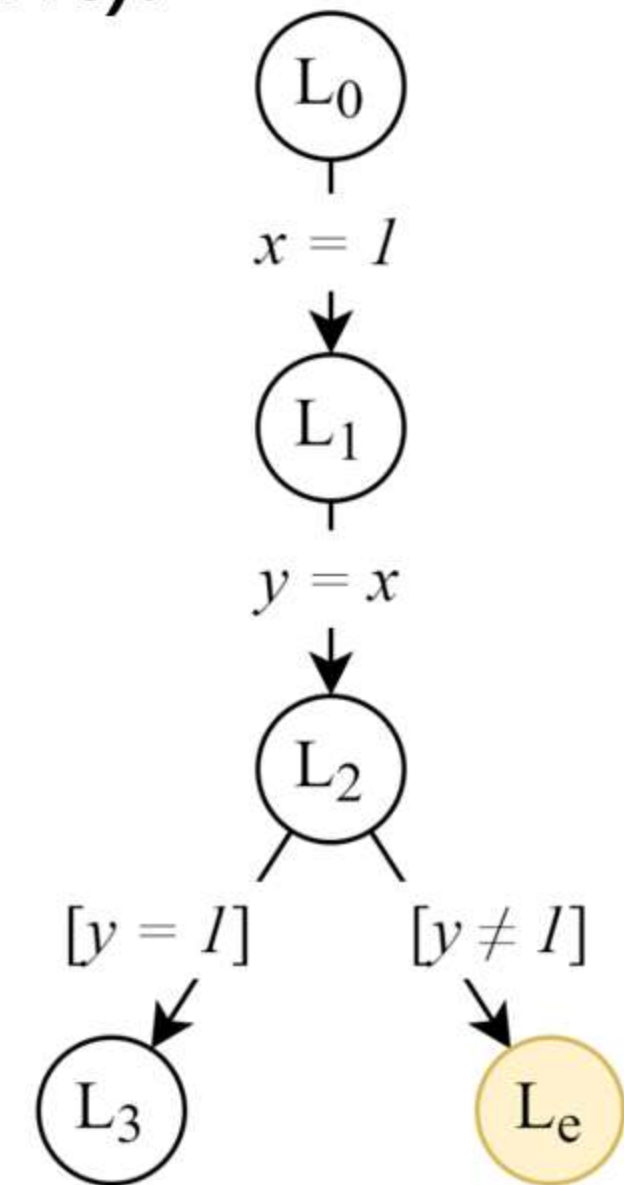
- With our algorithm:

$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

- **original statements must be used** for checking counterexamples

SPURIOUS

NOT SPURIOUS



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

- With our algorithm:

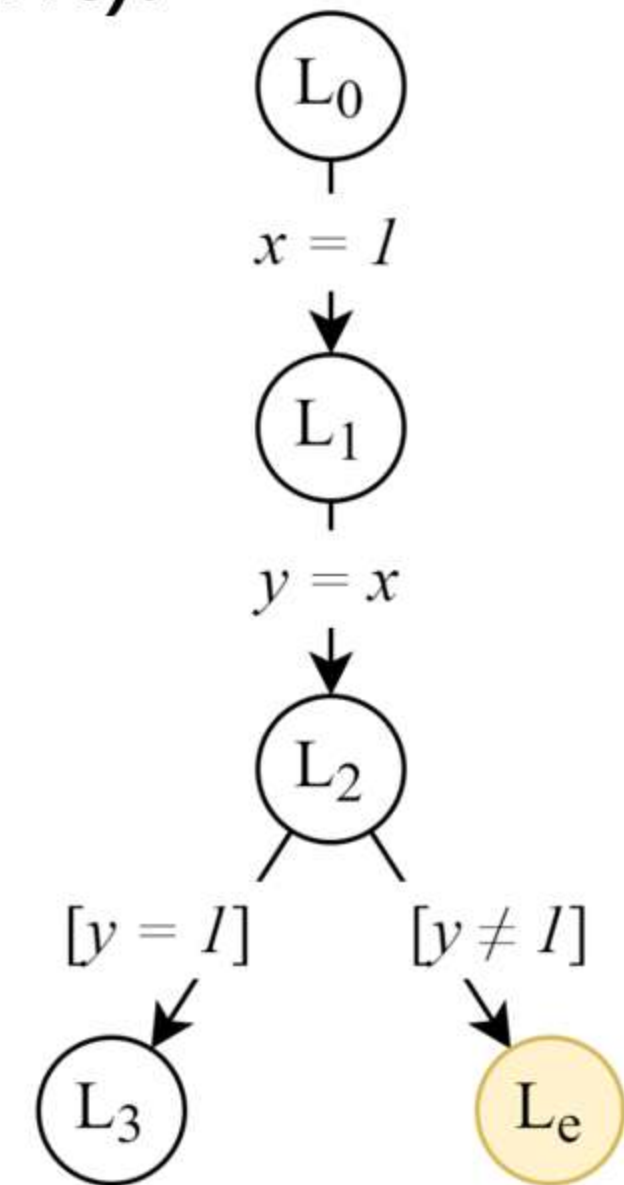
$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

SPURIOUS

NOT SPURIOUS

- **original statements must be used** for checking counterexamples

- Another problem:



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

SPURIOUS

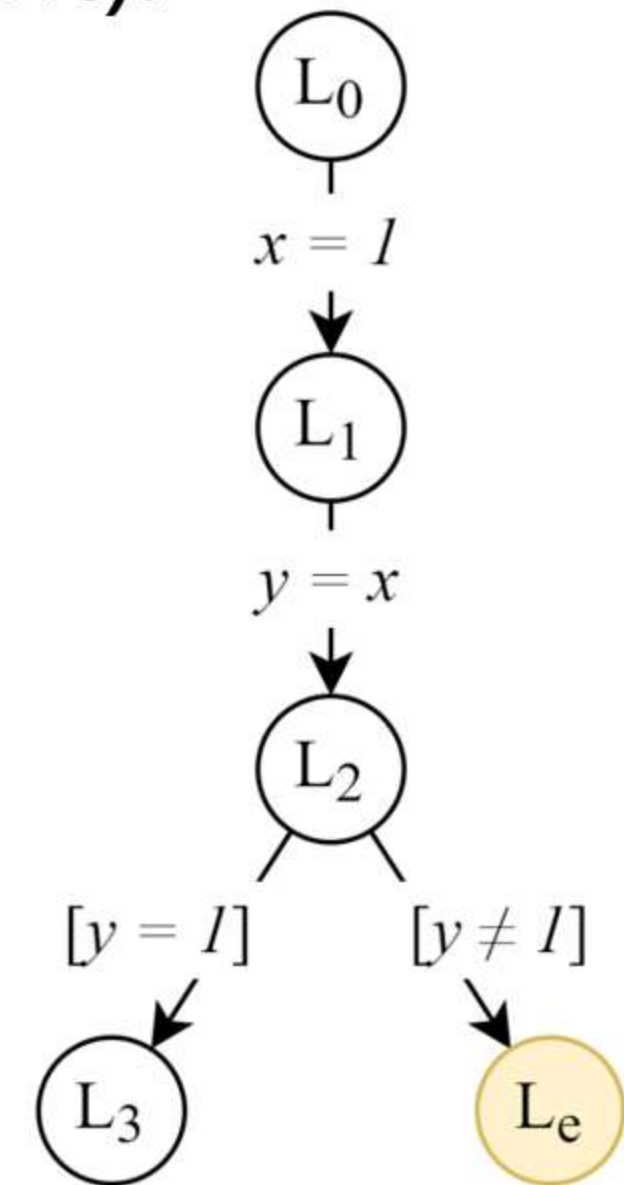
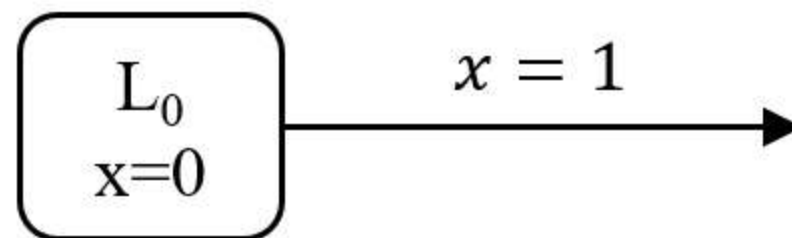
- With our algorithm:

$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

NOT SPURIOUS

- **original statements must be used** for checking counterexamples

- Another problem:



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

SPURIOUS

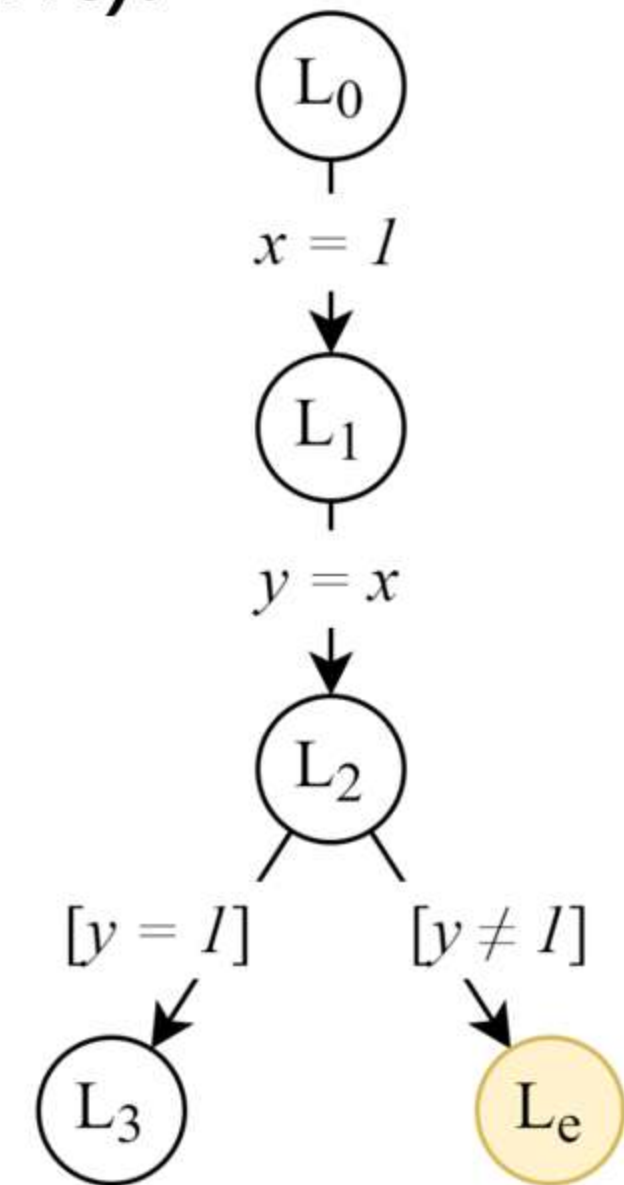
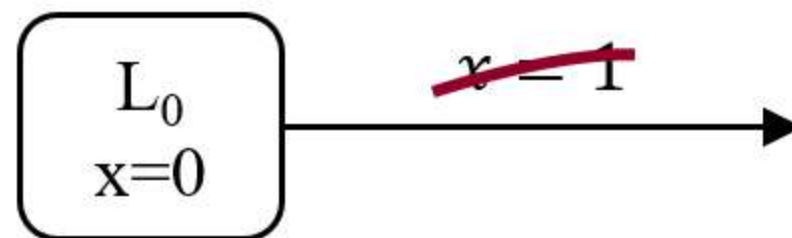
- With our algorithm:

$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

NOT SPURIOUS

- **original statements must be used** for checking counterexamples

- Another problem:



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

- With our algorithm:

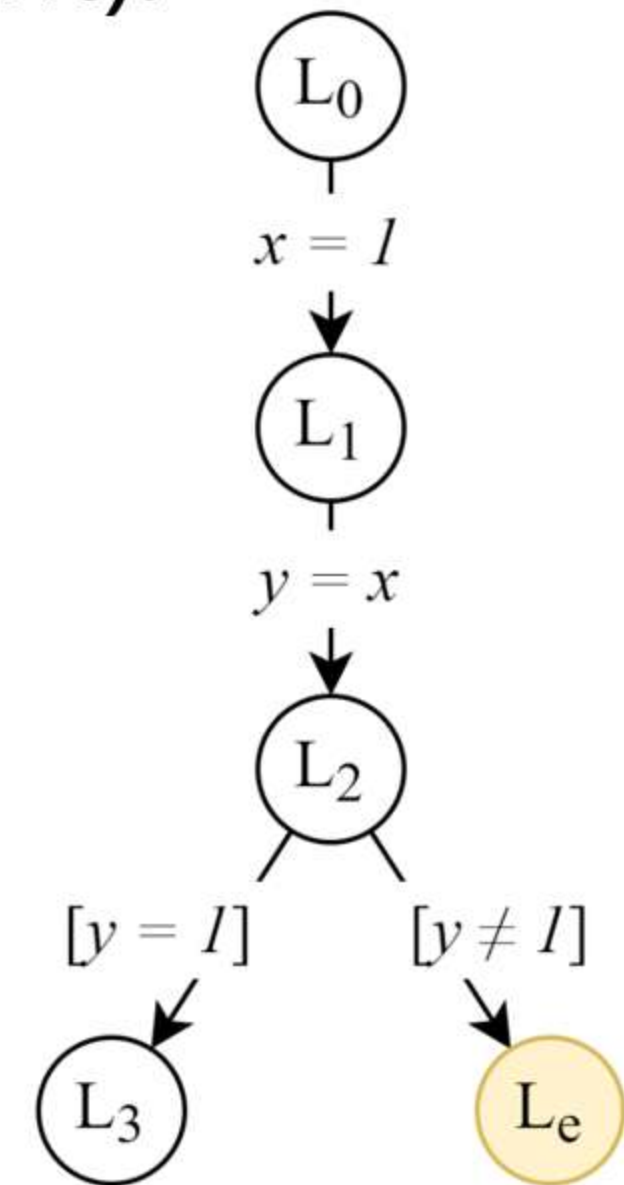
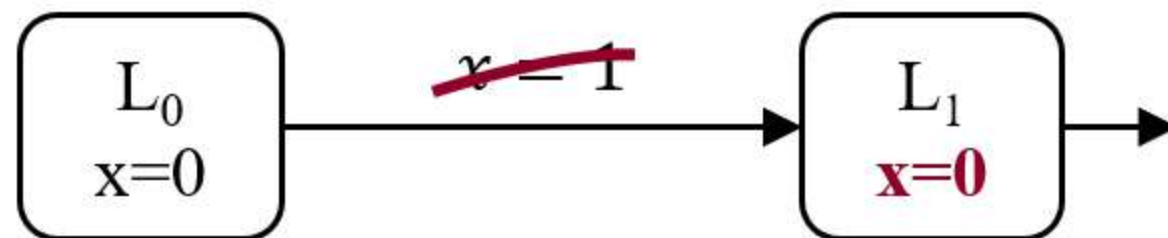
$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

SPURIOUS

NOT SPURIOUS

- **original statements must be used** for checking counterexamples

- Another problem:



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

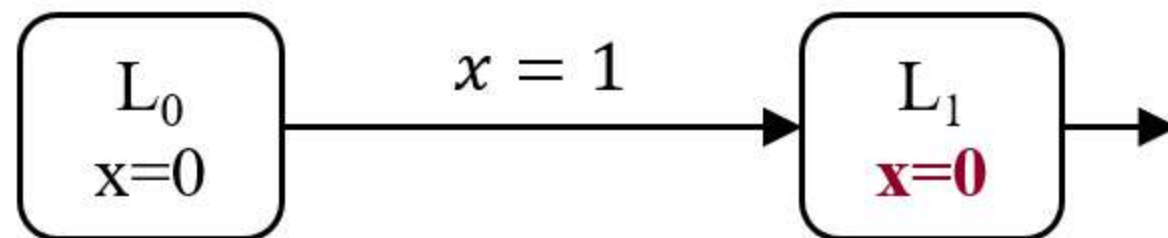
SPURIOUS

- With our algorithm:

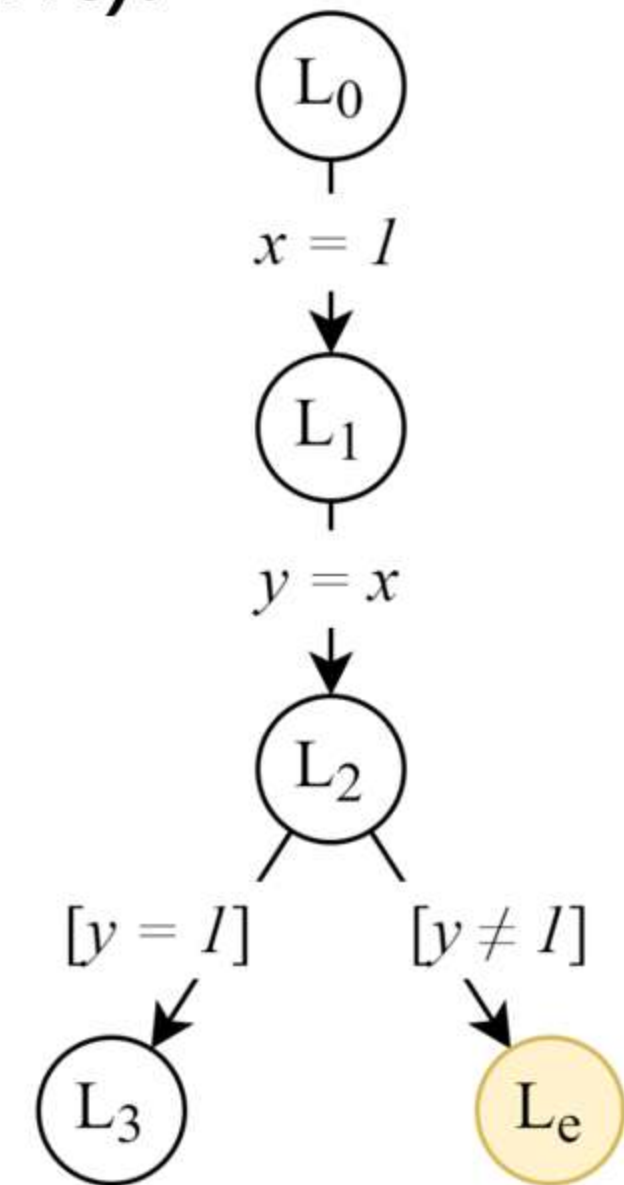
$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

NOT SPURIOUS

- **original statements must be used** for checking counterexamples
- Another problem:



Checking counterexample
with original statement



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

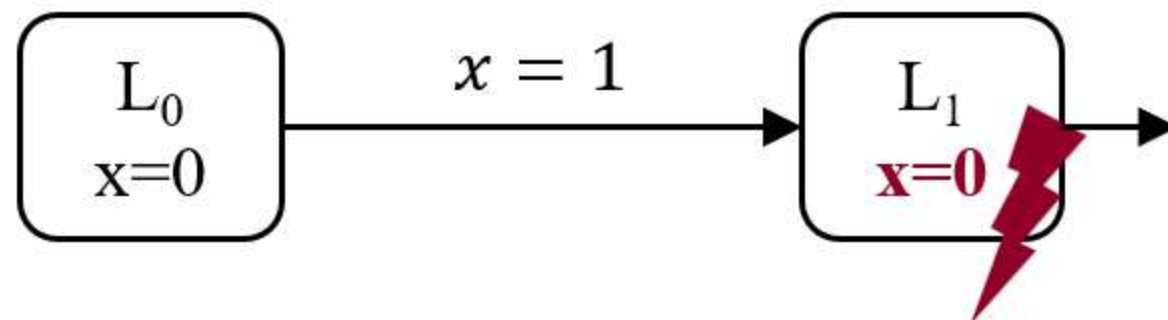
SPURIOUS

- With our algorithm:

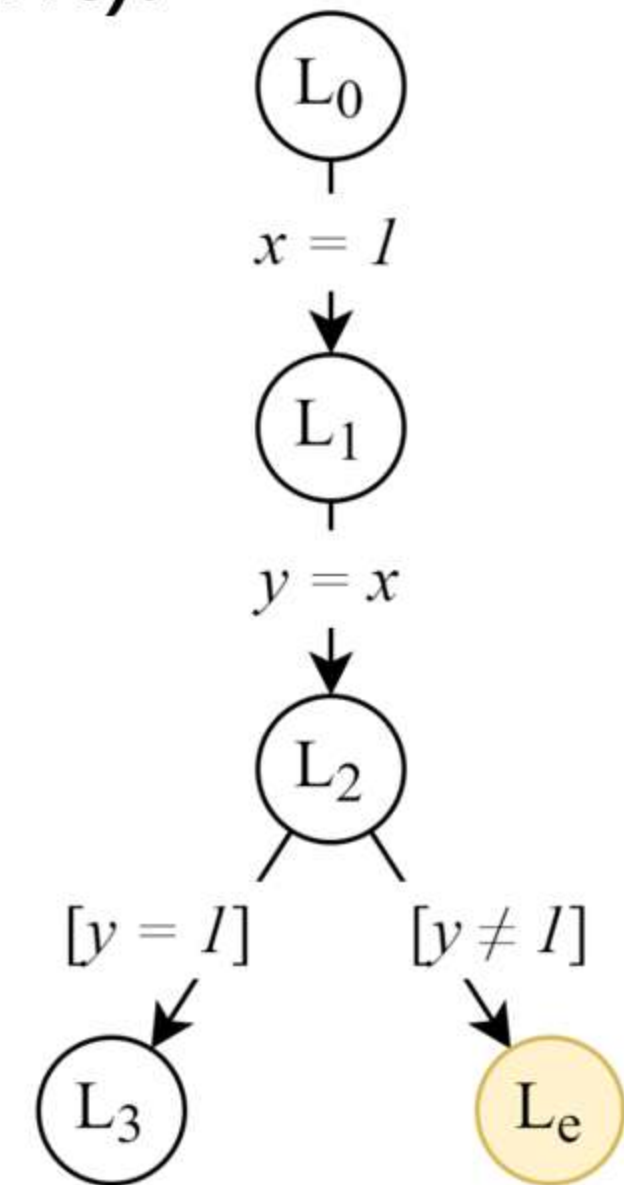
$\left. \begin{matrix} x=1 \\ y=x \end{matrix} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

NOT SPURIOUS

- **original statements must be used** for checking counterexamples
- Another problem:



Checking counterexample
with original statement



Algorithmic Integration – CEGAR

- CEGAR (Counterexample-Guided Abstraction Refinement):

- E.g., abstract precision: only x is tracked

Counterexample: $(x = 1, y = x, [y \neq 1])$

SPURIOUS

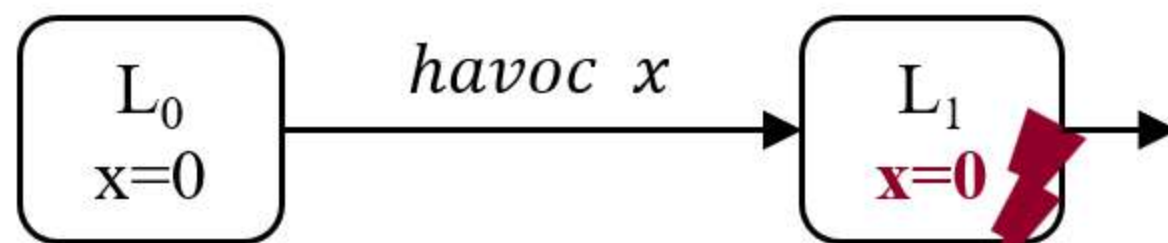
- With our algorithm:

$\left. \begin{array}{l} x=1 \\ y=x \end{array} \right\}$ not observed $\rightarrow$ erased: $(-, -, [y \neq 1])$

NOT SPURIOUS

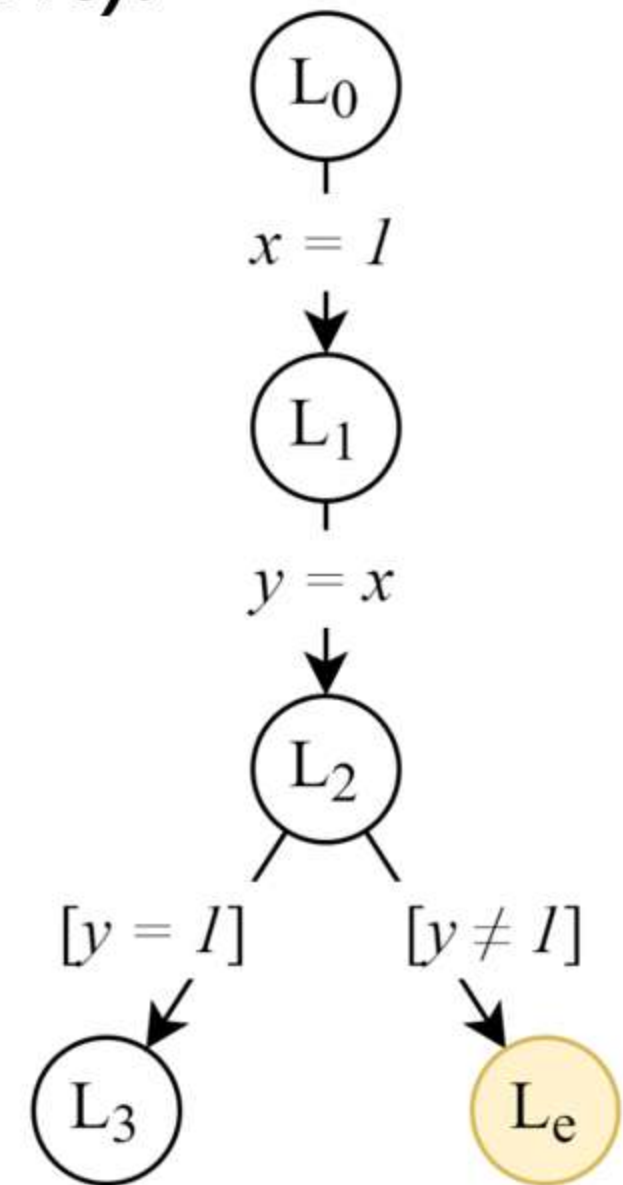
- **original statements must be used** for checking counterexamples

- Another problem:



Checking counterexample
with original statement

Solution: **non-deterministic assignment** instead of complete erasure



Algorithmic Integration

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**
 - **our algorithm erases variable accesses**
 - less dependent statements

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**
 - **our algorithm erases variable accesses**
 - less dependent statements
 - exponential reduction possible:

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**
 - **our algorithm erases variable accesses**
 - less dependent statements
 - exponential reduction possible:

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

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**
 - **our algorithm erases variable accesses**
 - less dependent statements
 - exponential reduction possible:

**Good abstraction:
only track $z \bmod 2 = 0$**

```
initially: x:=y:=z:=0

Process p0 | Processes p1 – pN
repeat N times: | y := y+1
  z := z+2*y
if z mod 2 = 0: |
  x := 0 | Processes pN+1 – p2N
else: | y := y*y
  x := 1
finally: assert(x*y=0)
```

Algorithmic Integration

- Partial-order Reduction (POR): interleaving-based reduction technique for model checking concurrent software
 - **based on statement dependence**
 - **our algorithm erases variable accesses**
 - less dependent statements
 - exponential reduction possible:

Not observed statements:
dependency removed

Good abstraction:
only track $z \bmod 2 = 0$

initially: $x := y := z := 0$

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

finally: $\text{assert}(x * y = 0)$

Evaluation – Experiment Design



- **Implementation** of the algorithm in the THETA model checker

Sources:

gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c

github.com/ftsrg/theta

Evaluation – Experiment Design



- **Implementation** of the algorithm in the THETA model checker
- **Benchmark** data: SV-COMP **C concurrent programs**

Sources:

gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c

github.com/ftsrg/theta

Evaluation – Experiment Design



- **Implementation** of the algorithm in the THETA model checker
- **Benchmark** data: SV-COMP **C concurrent programs**
- Configurations:
 - Baseline (simple cone-of-influence) + proposed algorithm + combination: SCOI, DCOI, SCOI+DCOI
 - Different abstract domains: EXPL, PRED

Sources:

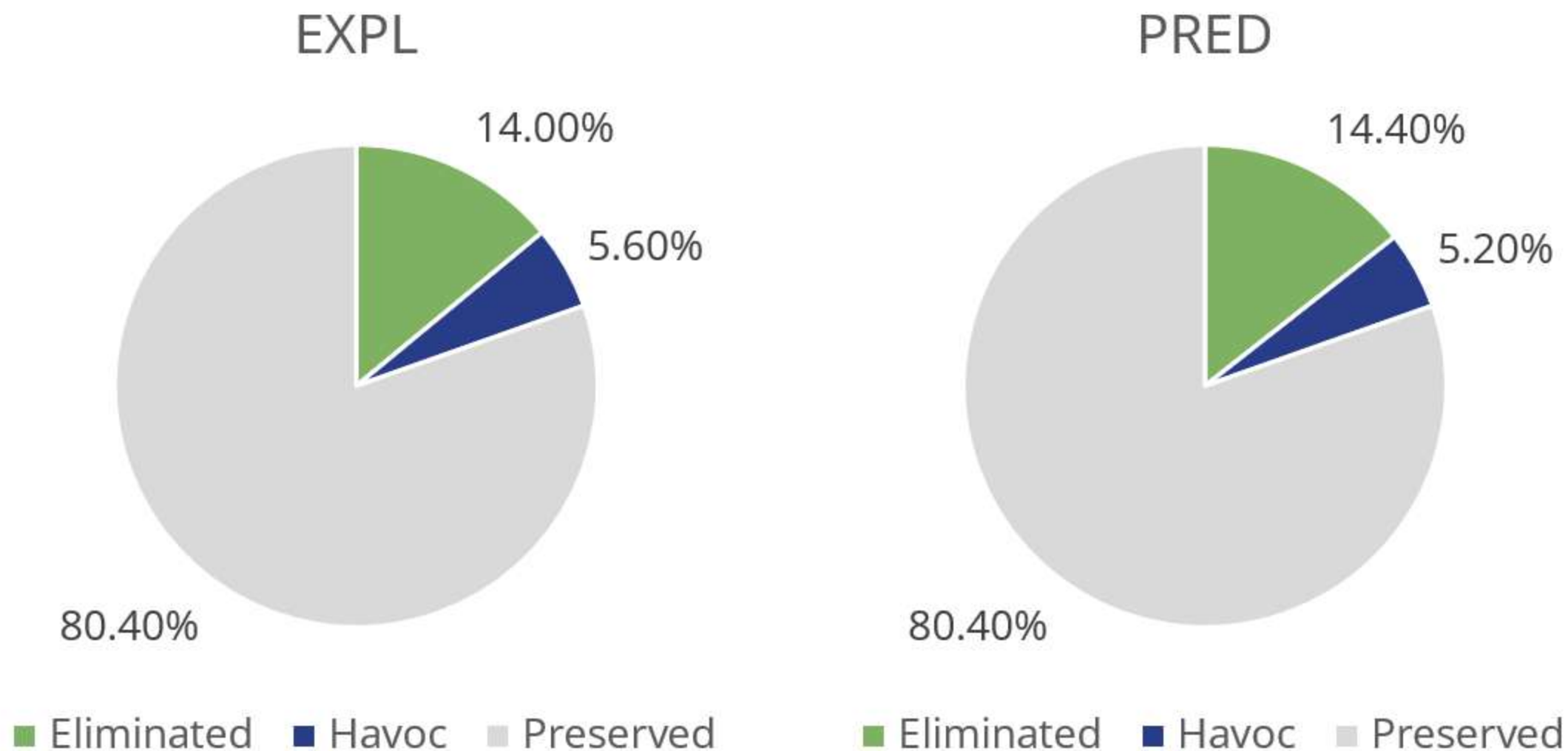
gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c

github.com/ftsrg/theta

Evaluation – Results

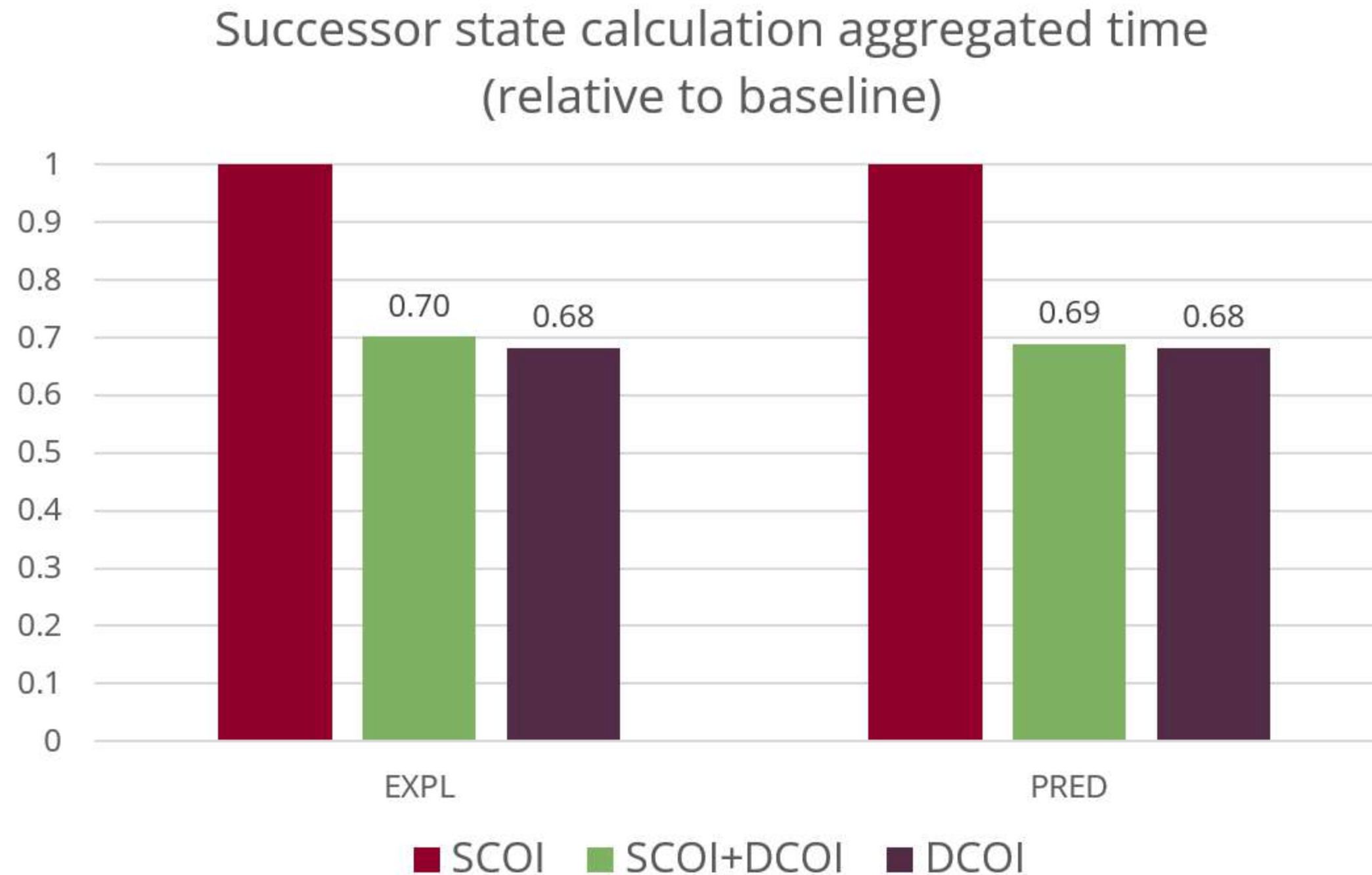
RQ1 What is the proportion of simplified statements?

Simplified by DCOI (when SCOI is applied first):



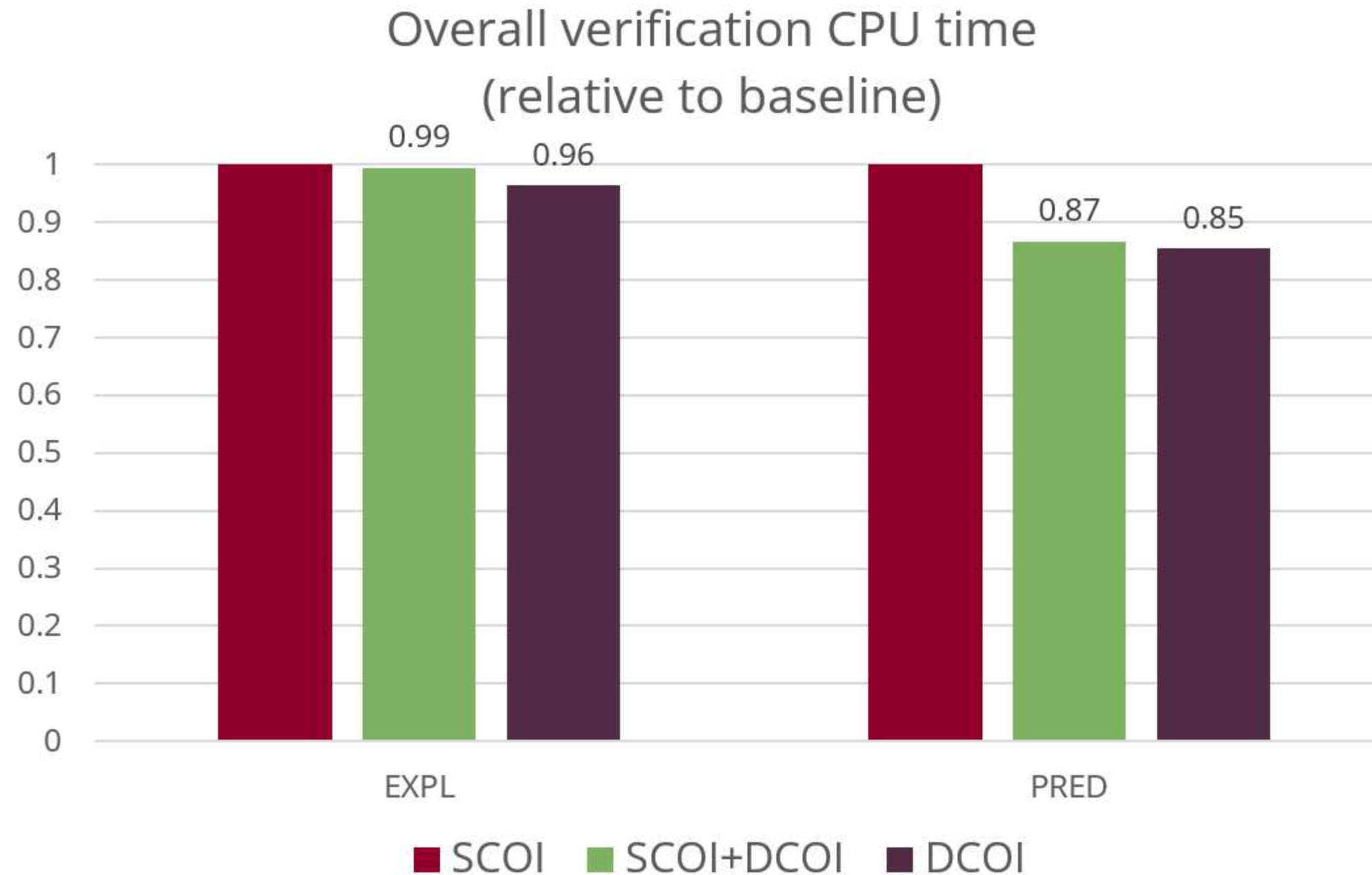
Evaluation – Results

RQ2 How is the time of successor state calculation affected?



Evaluation – Results

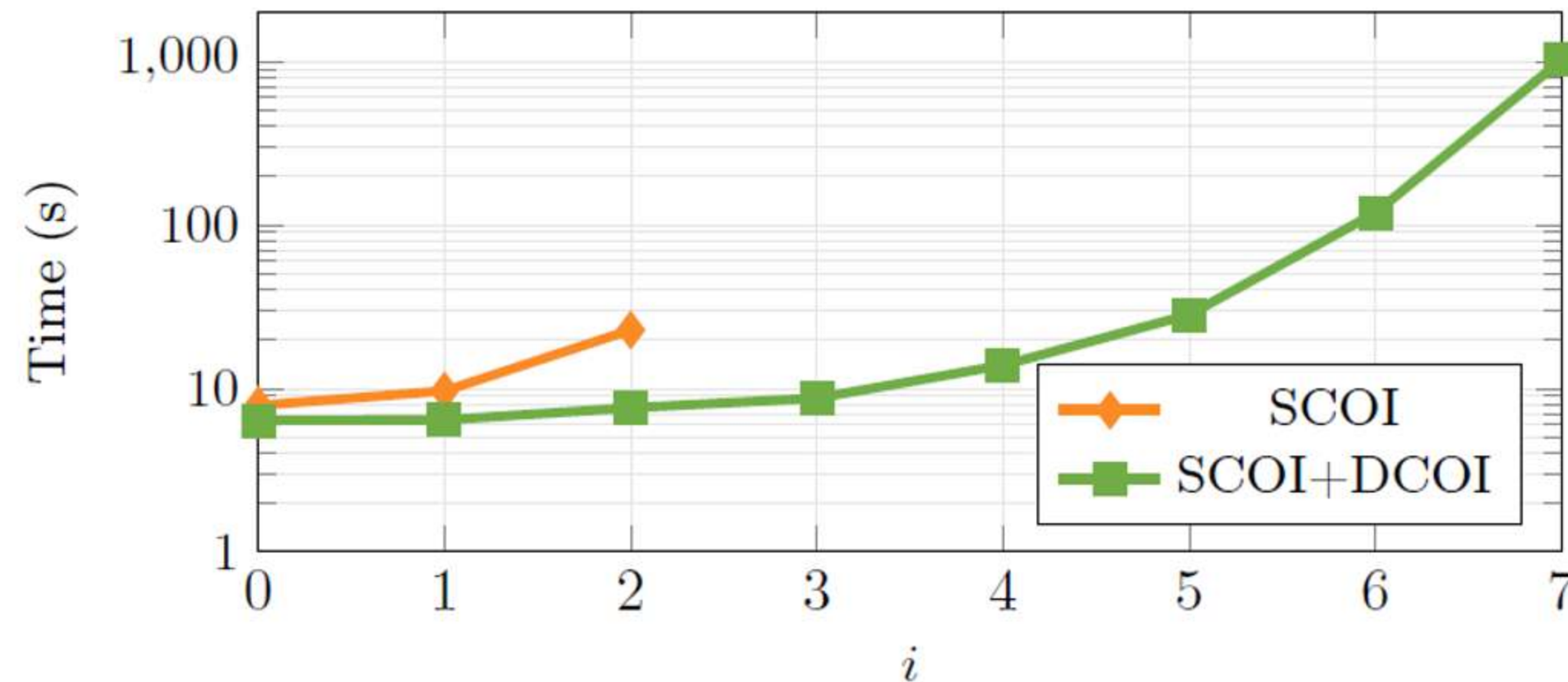
RQ3 How is the overall performance of the verification affected?



**Up to 15%
reduction**

Evaluation – Results

RQ4 What is the practical performance improvement where exponential gain is expected?



Number of threads: $2N + 1$ where $N = 2^i$

Summary & Questions

