

# SMT and Its Application in **Software Verification (Part II)**

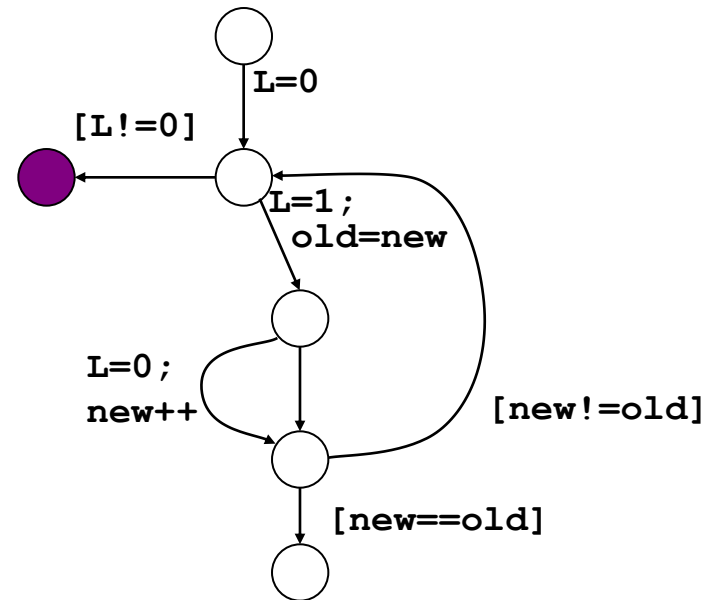
Yu-Fang Chen  
IIS, Academia Sinica

Based on the slides of Barrett, Sanjit, Kroening , Rummer, Sinha,  
Jhala, and Majumdar, McMillan

# Lazy abstraction -- an example

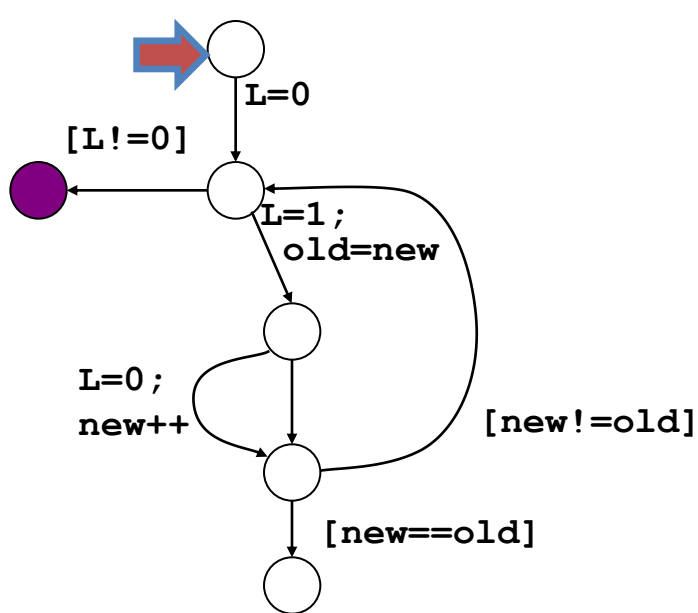
```
do{  
  lock();  
  old = new;  
  if(*){  
    unlock();  
    new++;  
  }  
} while (new != old);
```

program fragment



control-flow graph

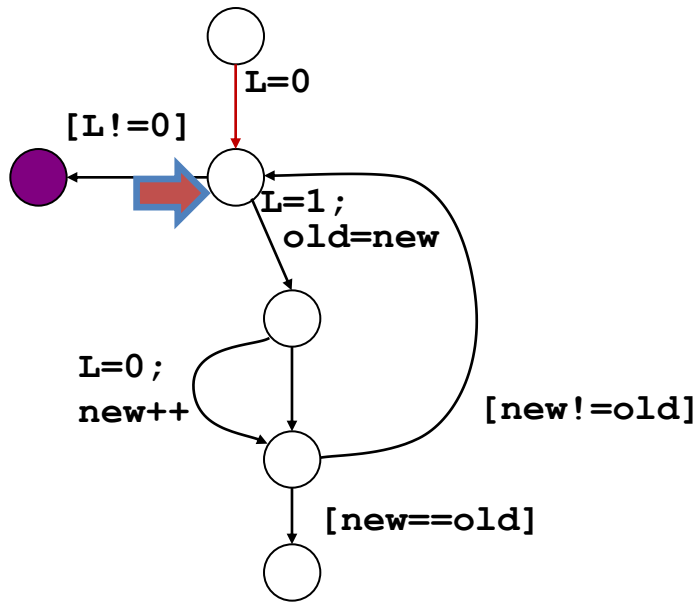
# Unwinding the CFG



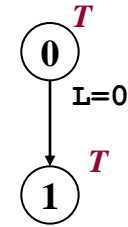
control-flow graph

$\textcircled{0}^T$

# Unwinding the CFG



control-flow graph

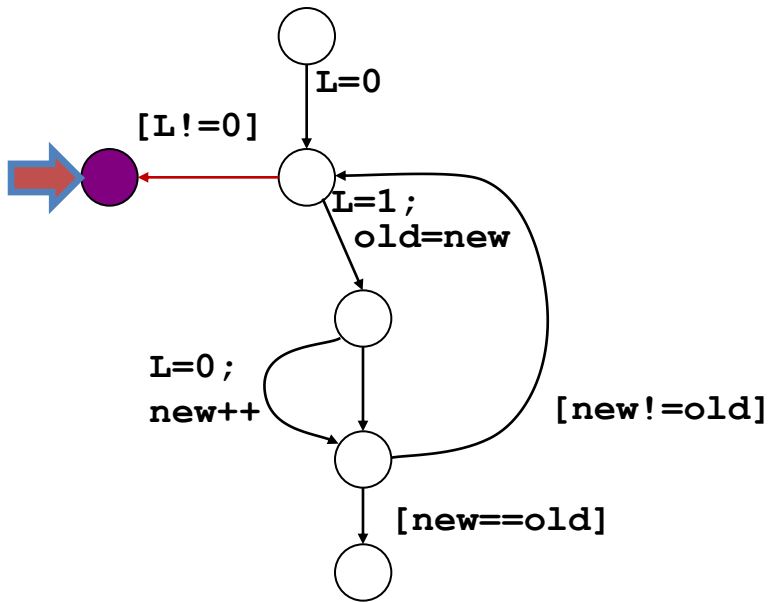


Replace all free occurrences of  $L$  in the formula with  $L'$

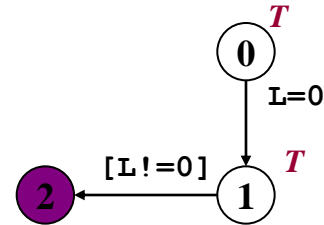
Compute  $\text{Post}(T, L=0) = T[L/L'] \wedge L=0[L/L']$   
 $= (L=0)$

Make Abstraction  $(L=0) \rightarrow T$  **Pass**

# Unwinding the CFG



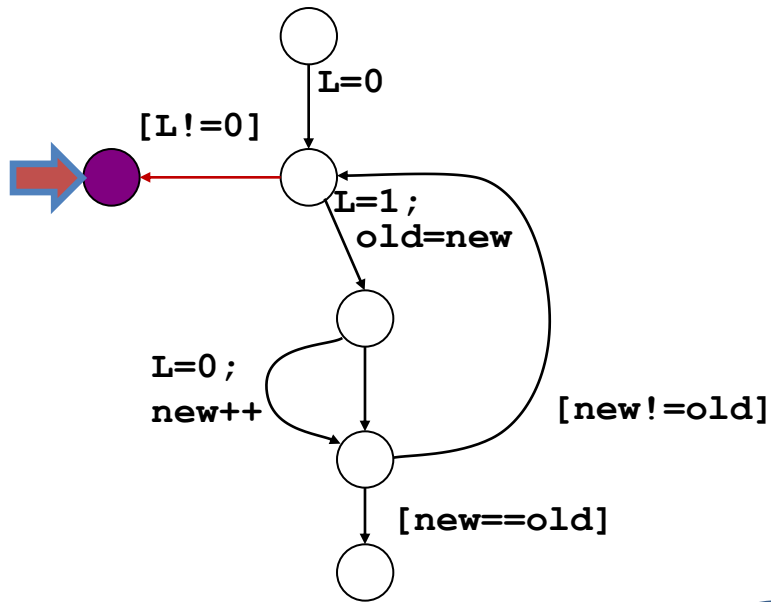
control-flow graph



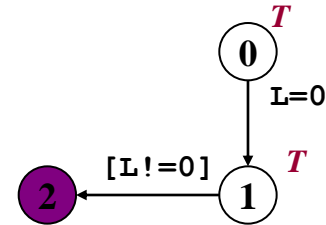
Compute  $\text{Post}(T, [L \neq 0]) = T \wedge (L \neq 0)$   
 $= (L \neq 0)$

**ERROR state reached!**

# Unwinding the CFG

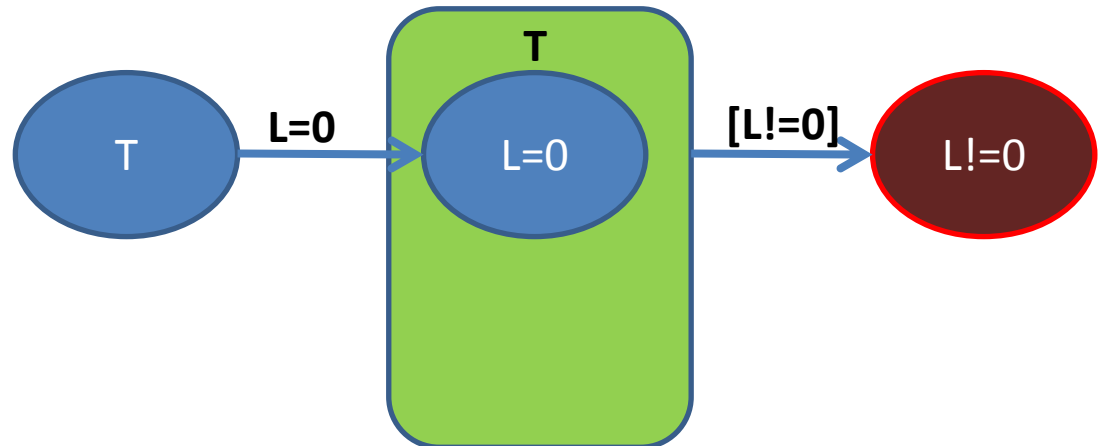


control-flow graph

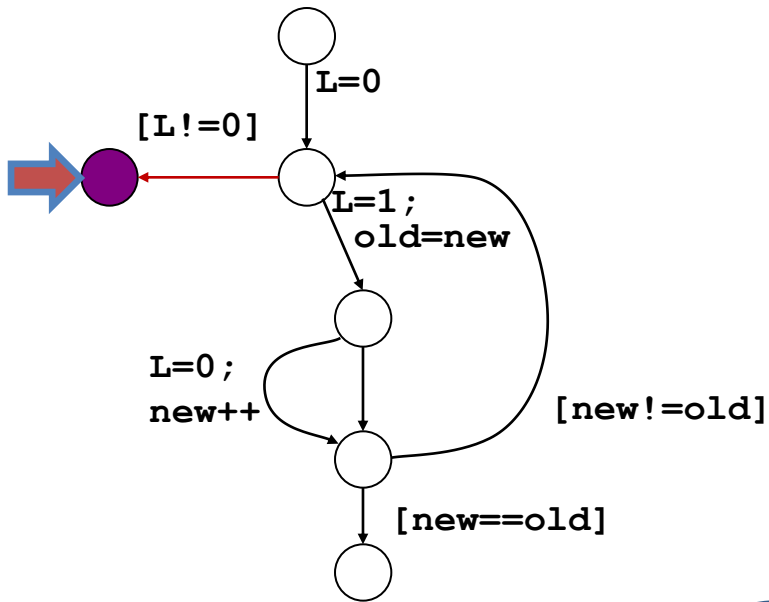


Compute  $\text{Post}(T, [L!=0]) = T \wedge (L!=0)$   
 $= (L!=0)$

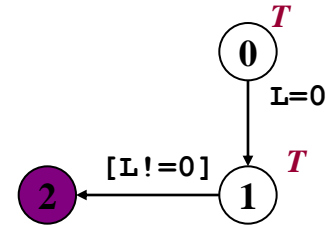
**ERROR state reached!**



# Unwinding the CFG

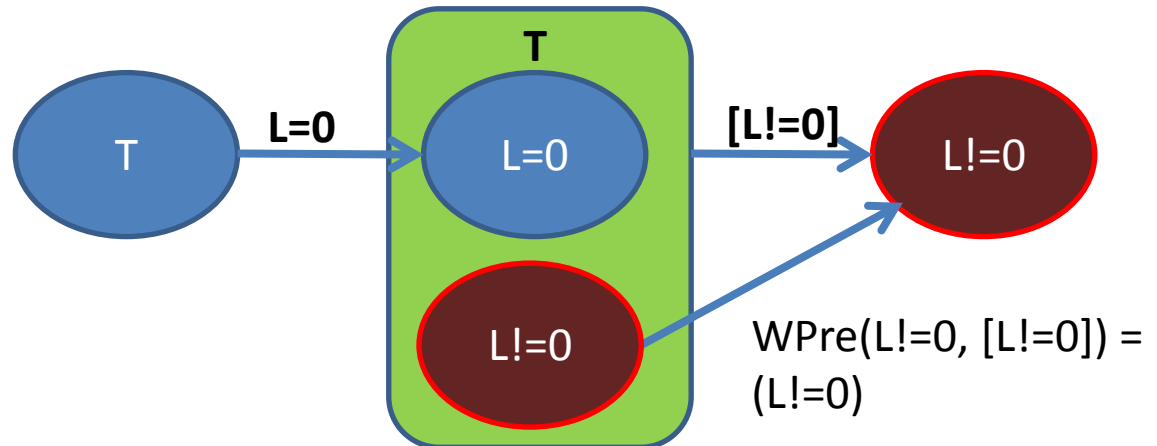


control-flow graph

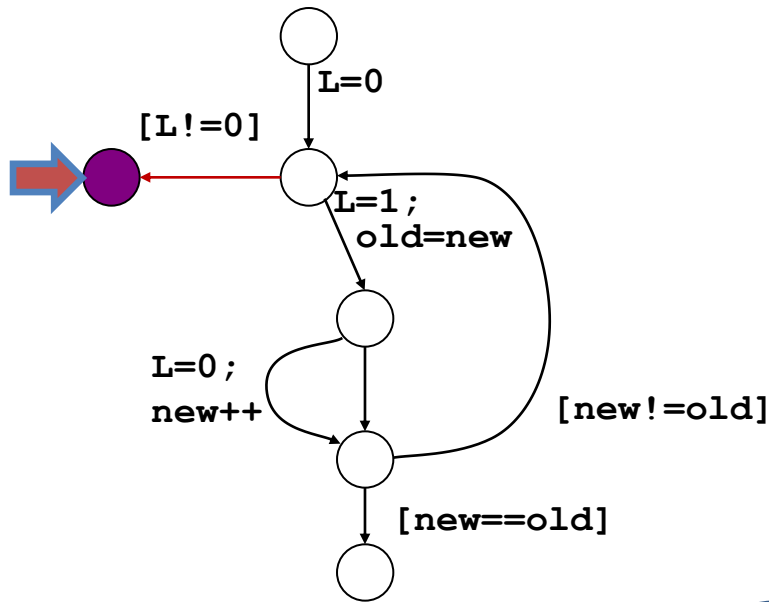


Compute  $\text{Post}(T, [L \neq 0]) = T \wedge (L \neq 0)$   
 $= (L \neq 0)$

**ERROR state reached!**



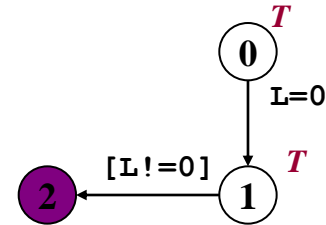
# Unwinding the CFG



control-flow graph

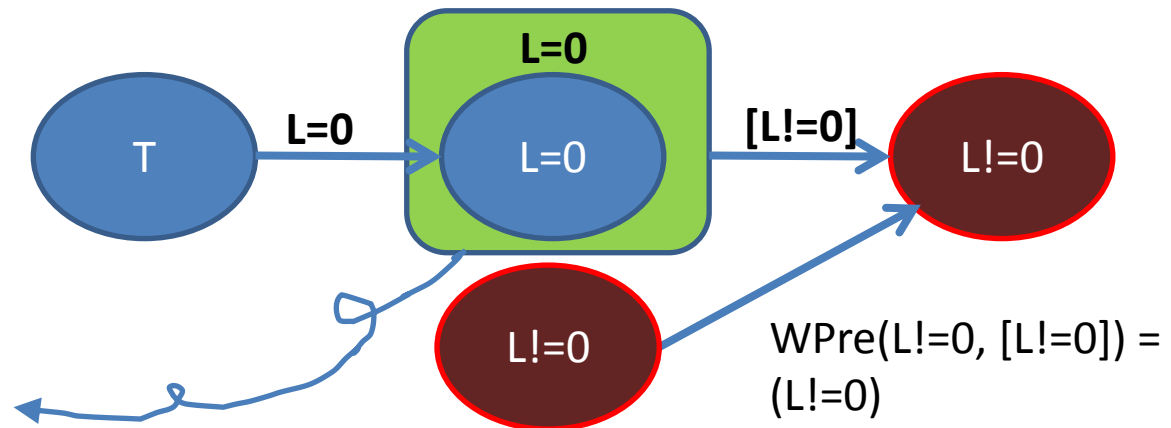
Compute Craig Interpolant:  $(L=0)$

1.  $(L=0) \rightarrow (L=0)$
2.  $(L=0) \wedge (L=0)$  is UNSAT
3. Use only share var. of  $(L=0)$  and  $(L \neq 0)$



Compute  $\text{Post}(T, [L \neq 0]) = T \wedge (L \neq 0)$   
 $= (L \neq 0)$

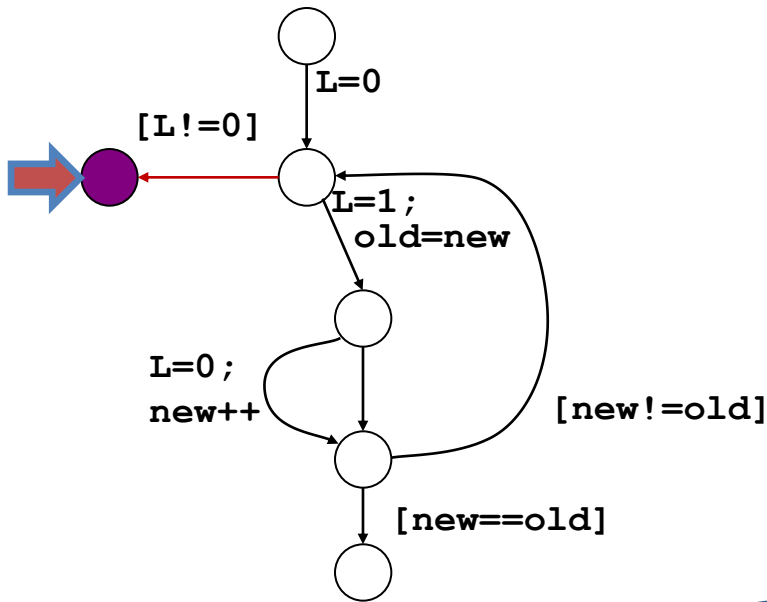
**ERROR state reached!**



$\text{WPre}(L \neq 0, [L \neq 0]) = (L \neq 0)$



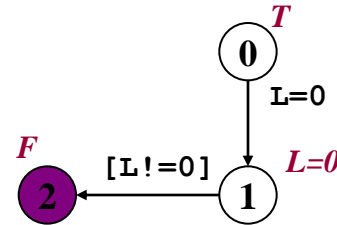
# Unwinding the CFG



control-flow graph

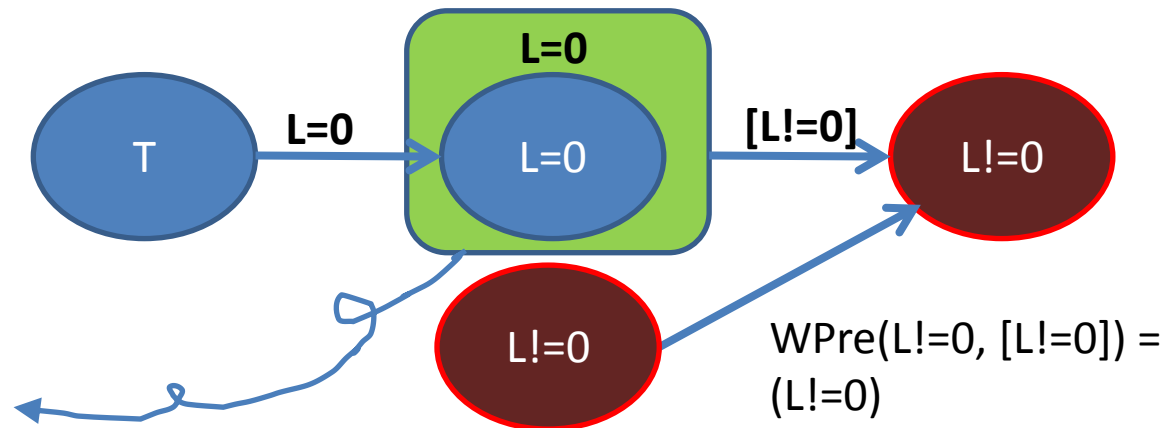
Compute Craig Interpolant:  $(L=0)$

1.  $(L=0) \rightarrow (L=0)$
2.  $(L=0) \wedge (L=0)$  is UNSAT
3. Use only share var. of  $(L=0)$  and  $(L \neq 0)$

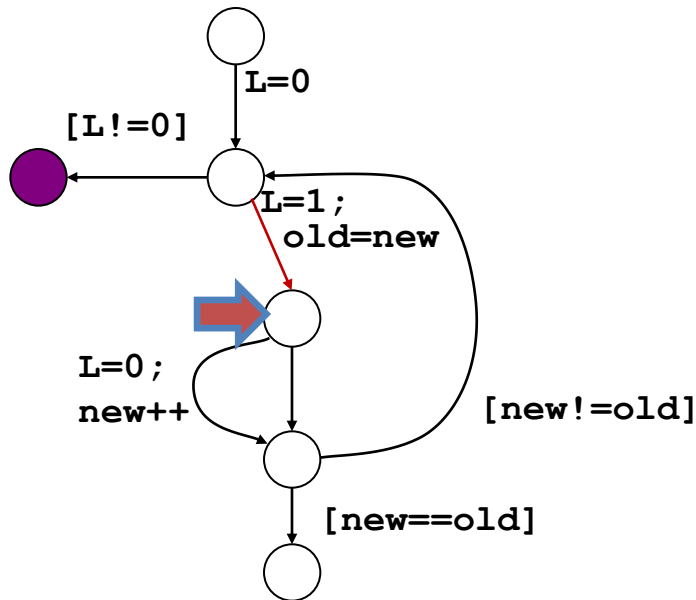


Compute  $\text{Post}(T, [L \neq 0]) = T \wedge (L \neq 0)$   
 $= (L \neq 0)$

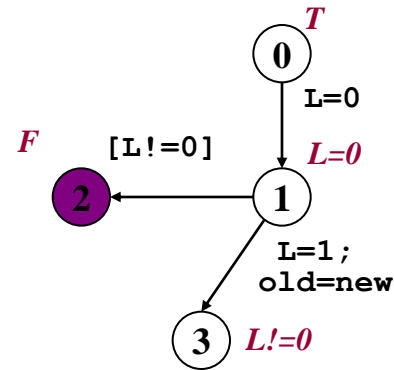
**ERROR state reached!**



# Unwinding the CFG



control-flow graph



Compute Post ( $L=0, L=1$ )

$$= (L=0)[L/L'] \wedge L=1[L/L']$$

$$= (L'=0 \wedge L=1)$$

Compute Post ( $L'=0 \wedge L=1, \text{old}=\text{new}$ )

$$= (L'=0 \wedge L=1)[\text{old}/\text{old}'] \wedge \text{old}=\text{new}[\text{old}/\text{old}']$$

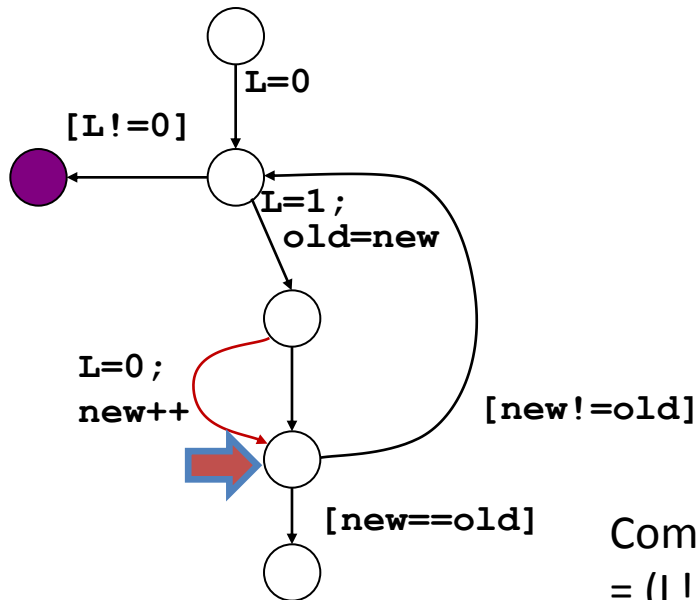
$$= L'=0 \wedge L=1 \wedge \text{old}=\text{new}$$

Make Abstraction

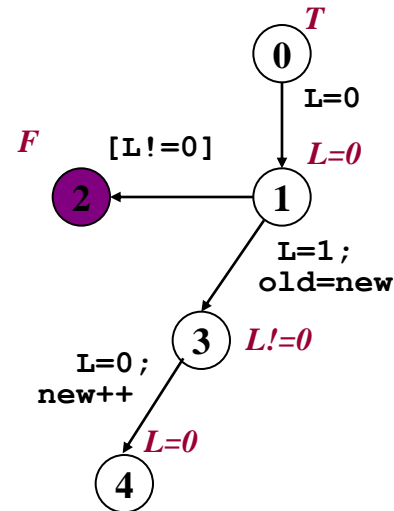
$$(L'=0 \wedge L=1 \wedge \text{old}=\text{new}) \rightarrow (L \neq 0) \quad \text{Pass}$$

$$(L'=0 \wedge L=1 \wedge \text{old}=\text{new}) \rightarrow (L=0) \quad \text{Not Passed}$$

# Unwinding the CFG



control-flow graph



Compute Post ( $L \neq 0, L=0$ )

$$= (L \neq 0)[L/L'] \wedge L=0[L/L']$$

$$= (L' \neq 0 \wedge L=0)$$

Compute Post ( $L' \neq 0 \wedge (L=0), \text{new}=\text{new}+1$ )

$$= (L' \neq 0 \wedge L=0)[\text{new}/\text{new}'] \wedge \text{new}=(\text{new}+1)[\text{new}/\text{new}']$$

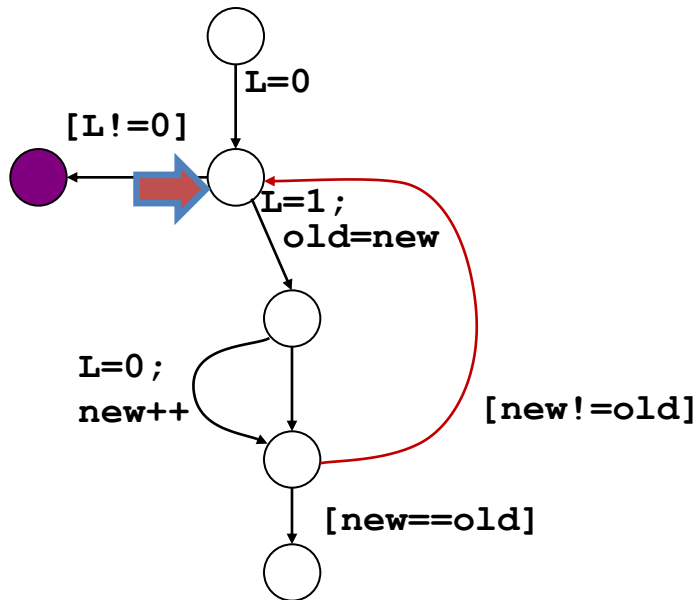
$$= (L' \neq 0 \wedge L=0 \wedge \text{new}=\text{new}'+1)$$

Make Abstraction

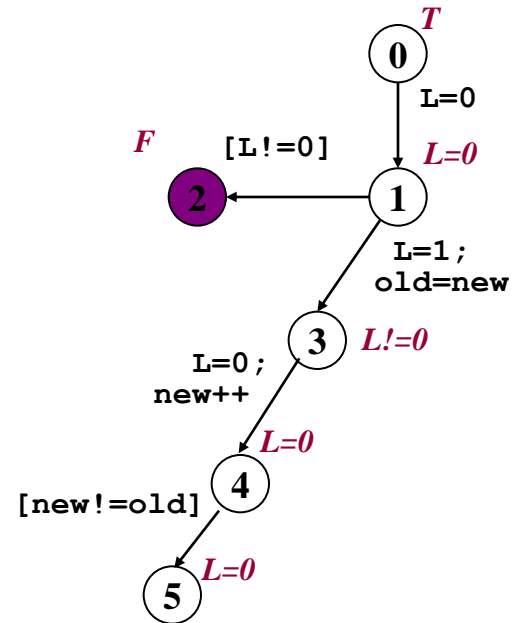
$$(L' \neq 0 \wedge L=0 \wedge \text{new}=\text{new}'+1) \rightarrow (L \neq 0) \quad \text{Not Passed}$$

$$(L' \neq 0 \wedge L=0 \wedge \text{new}=\text{new}'+1) \rightarrow (L=0) \quad \text{Pass}$$

# Unwinding the CFG



control-flow graph



Compute Post ( $L=0, [new!=old]$ )

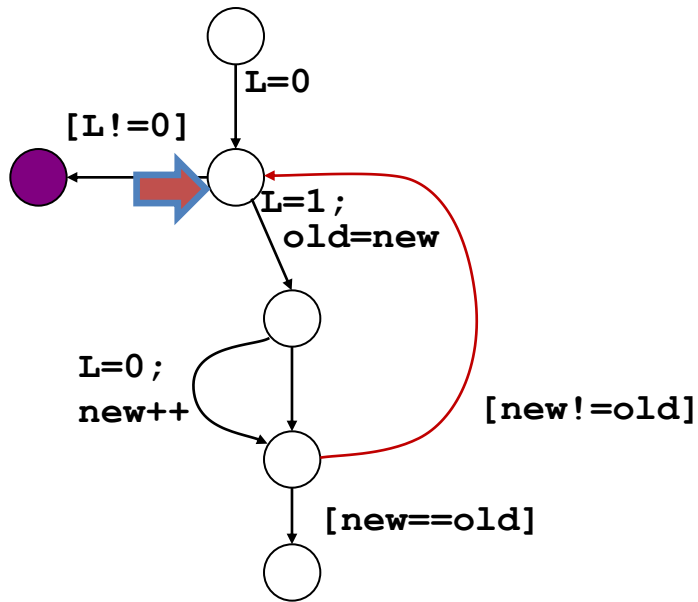
$= (L=0 \wedge new!=old)$

Make Abstraction

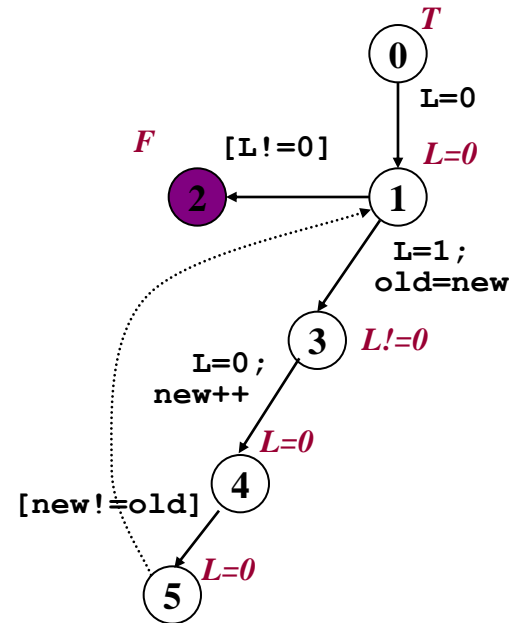
$(L=0 \wedge new!=old) \rightarrow (L!=0)$  **Not Passed**

$(L=0 \wedge new!=old) \rightarrow (L=0)$  **Pass**

# Unwinding the CFG



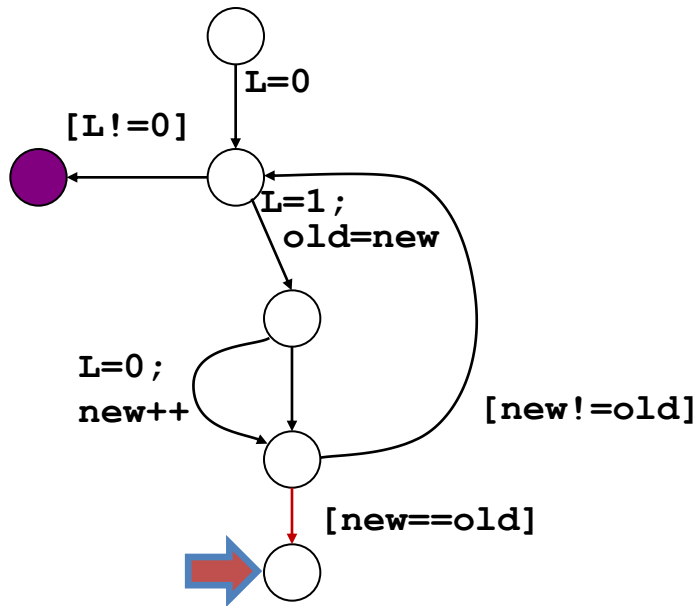
control-flow graph



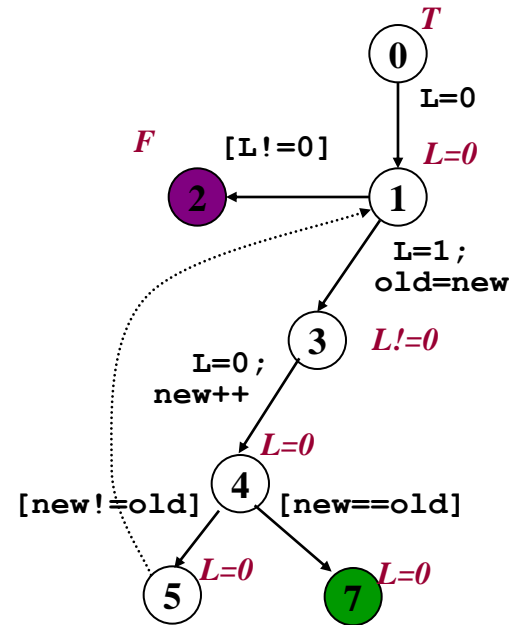
Covering: state 5 is subsumed by state 1.

$L=1 \rightarrow L=1$  **Pass**

# Unwinding the CFG



control-flow graph



Compute Post ( $L=0, [new == old]$ )

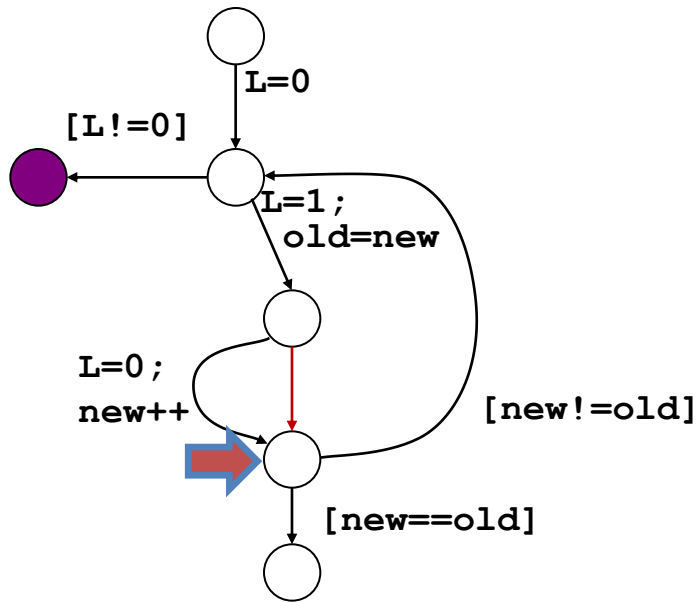
$= (L=0 \wedge new == old)$

Make Abstraction

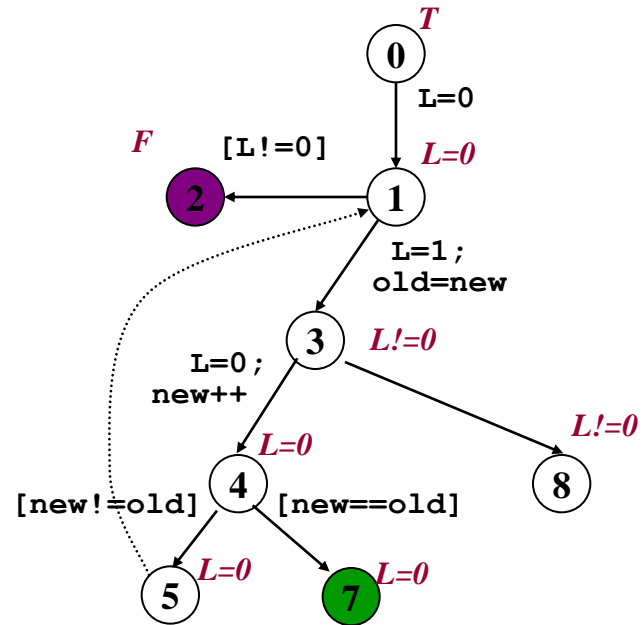
$(L=0 \wedge new \neq old) \rightarrow (L \neq 0)$  **Not Passed**

$(L=0 \wedge new \neq old) \rightarrow (L=0)$  **Pass**

# Unwinding the CFG

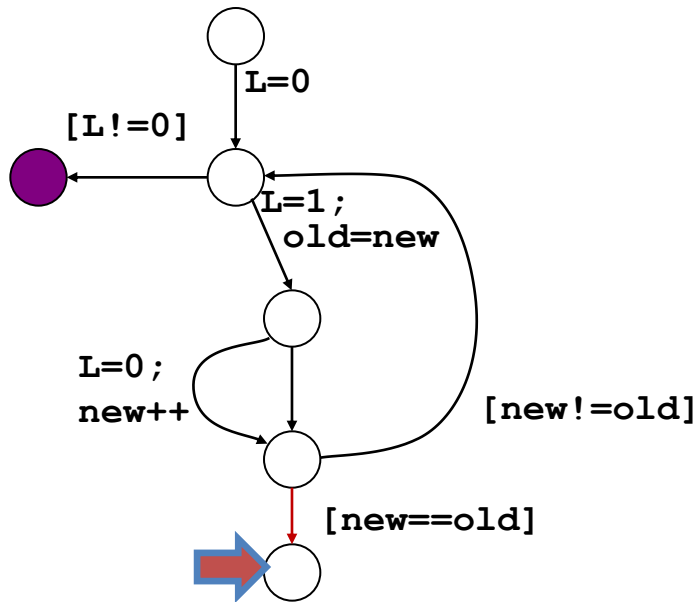


control-flow graph

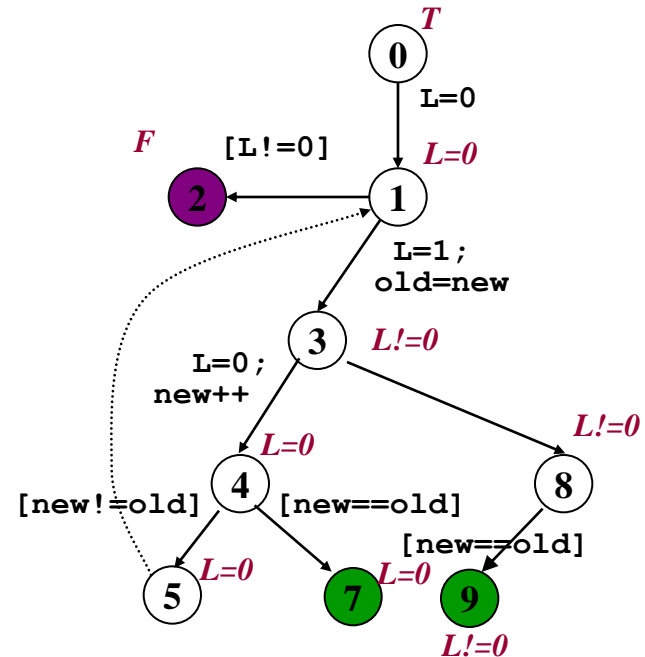


No Actions

# Unwinding the CFG



control-flow graph



Compute Post ( $L \neq 0, [\text{new} == \text{old}]$ )

$= (L \neq 0 \wedge \text{new} == \text{old})$

Make Abstraction

$(L \neq 0 \wedge \text{new} == \text{old}) \rightarrow (L \neq 0)$

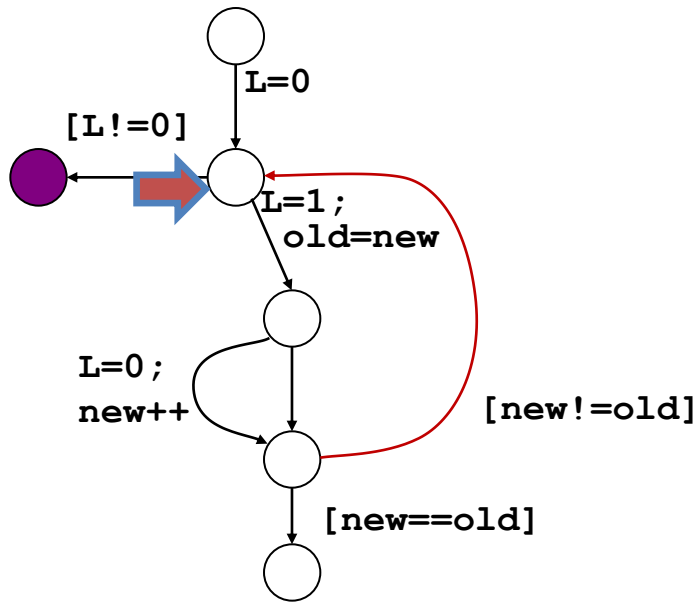
Pass

$(L \neq 0 \wedge \text{new} == \text{old}) \rightarrow (L=0)$

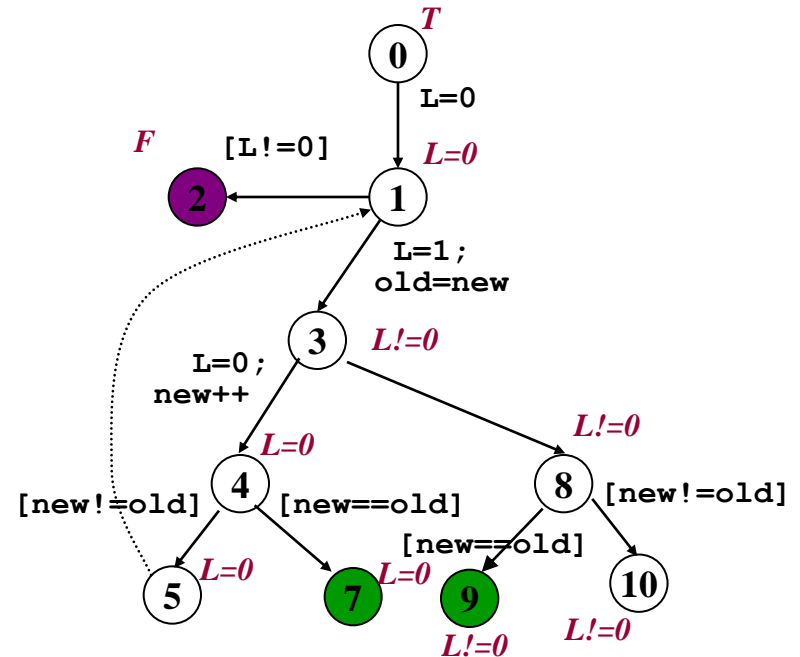
Not Passed



# Unwinding the CFG



control-flow graph



Compute Post ( $L \neq 0, [\text{new} \neq \text{old}]$ )

=  $(L \neq 0 \wedge \text{new} \neq \text{old})$

Make Abstraction

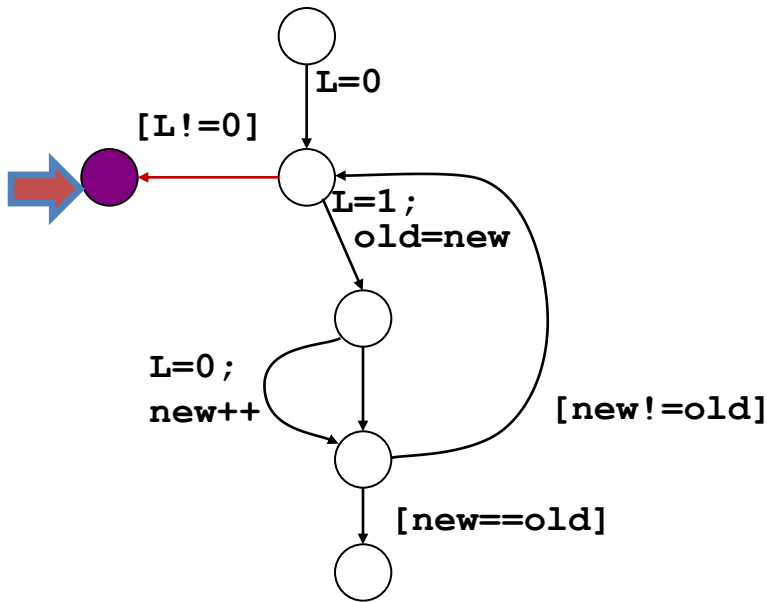
$(L \neq 0 \wedge \text{new} \neq \text{old}) \rightarrow (L \neq 0)$

Pass

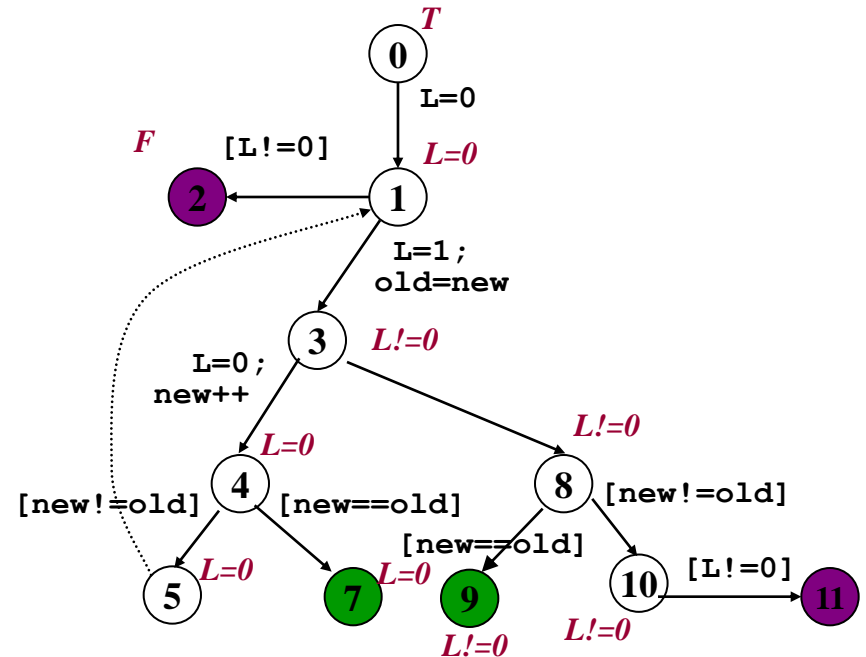
$(L \neq 0 \wedge \text{new} \neq \text{old}) \rightarrow (L=0)$

Not Passed

# Unwinding the CFG



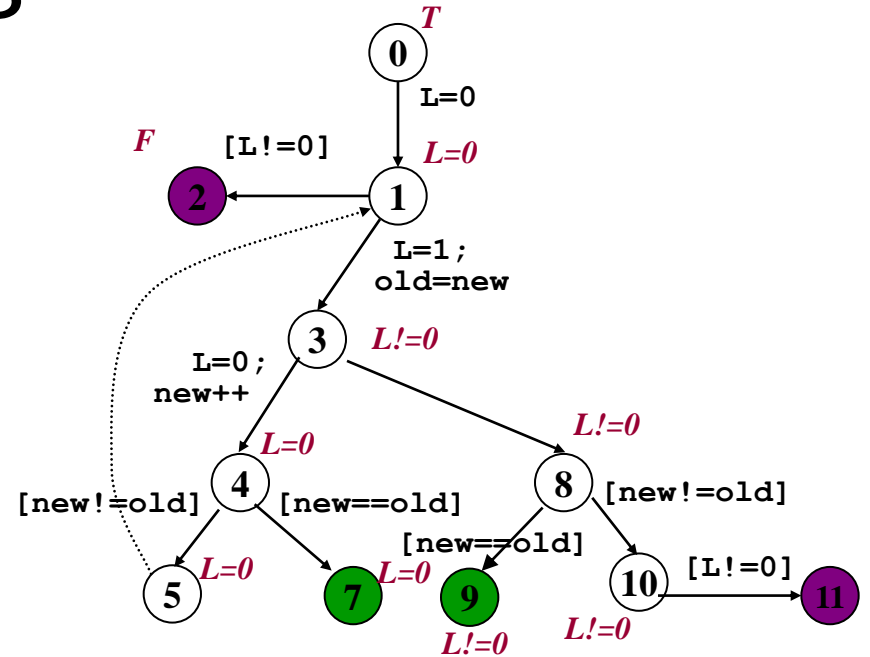
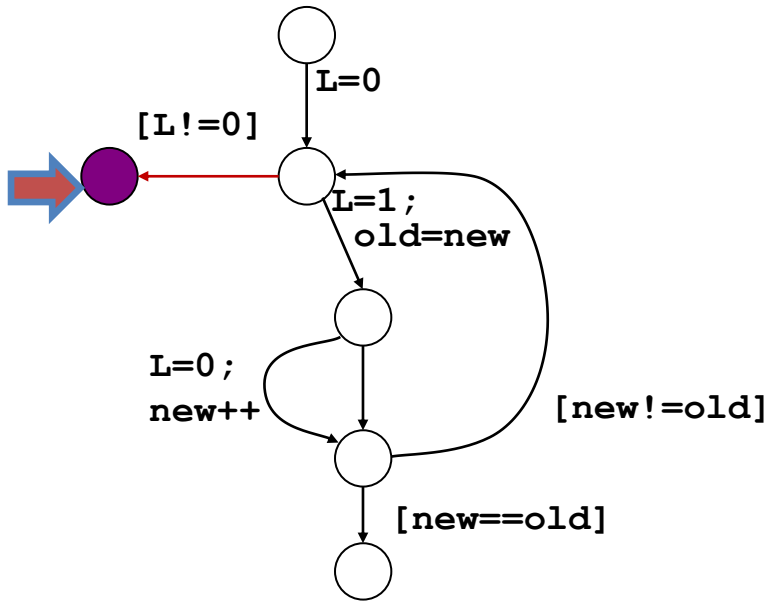
control-flow graph



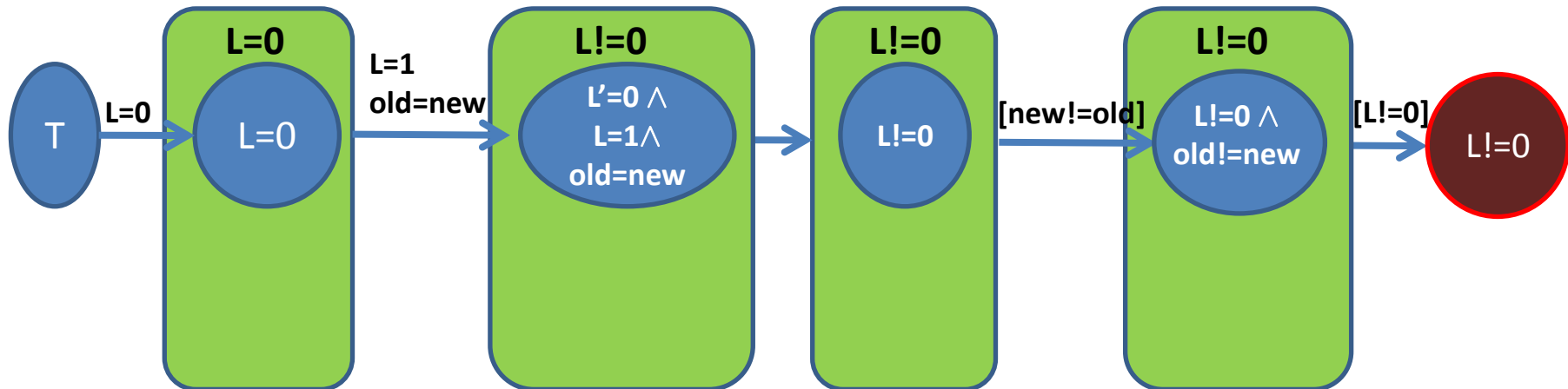
Compute Post ( $L!=0, [L!=0]$ )  
 $= (L!=0 \wedge L!0)$

**ERROR state reached!**

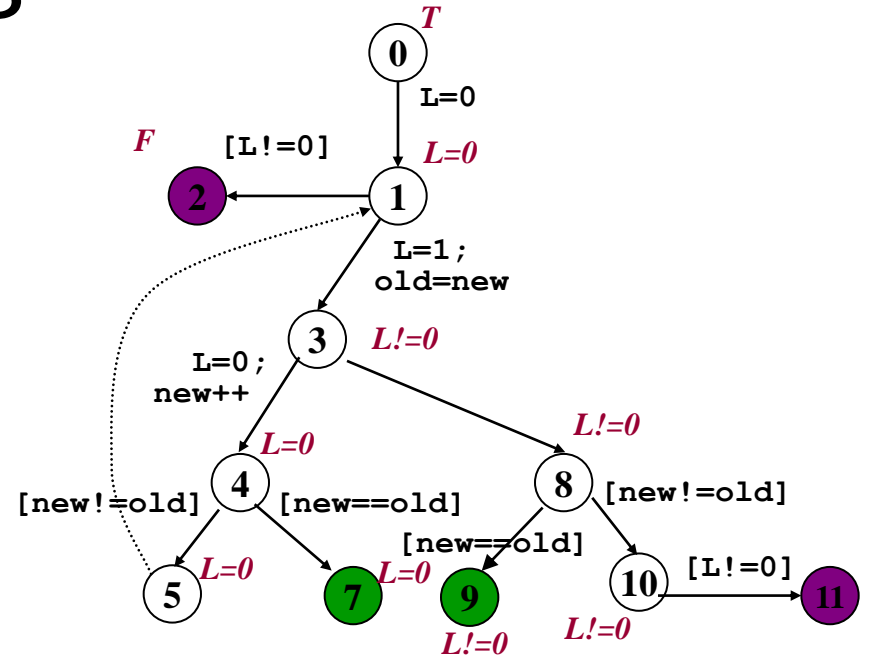
# Unwinding the CFG



control-flow graph



The control flow graph consists of seven nodes. The first node is a white circle. An edge labeled  $L=0$  leads to a second white circle. From the second node, a red edge labeled  $[L!=0]$  leads to a purple circle, which is the final state indicated by a large blue arrow. Another edge from the second node, labeled  $L=1;$  and  $old=new$ , leads to a third white circle. From the third node, an edge labeled  $L=0;$  and  $new++$  leads to a fourth white circle. From the fourth node, an edge labeled  $[new!=old]$  loops back to the second node. Another edge from the fourth node, labeled  $[new==old]$ , leads to a fifth white circle.



The diagram illustrates the execution of a program with memory consistency constraints. It shows a sequence of states and transitions:

- Initial State:** Thread  $T$  is in a state where  $L=0$ .
- Transition 1:** Labeled  $L=0$ , leading to a state where  $L=0$ .
- Transition 2:** Labeled  $L=1$  and  $old=new$ , leading to a state where  $L \neq 0$  and  $L'=0 \wedge L=1 \wedge old=new$ .
- Transition 3:** Labeled  $L \neq 0$ , leading to a state where  $L \neq 0$ .
- Transition 4:** Labeled  $[new \neq old]$ , leading to a state where  $L \neq 0$  and  $L \neq 0 \wedge old \neq new$ .
- Transition 5:** Labeled  $[L \neq 0]$ , leading to a final state where  $L \neq 0$ .

A red circle highlights the final state  $L \neq 0$ , and a green circle highlights the state  $L \neq 0 \wedge old \neq new$ .

The control flow graph consists of several nodes and edges:

- Initial Node:** A white circle at the top.
- Edge 1:** Labeled  $L=0$ , leading to a white circle.
- Edge 2:** Labeled  $[L!=0]$  (in red), leading from the white circle to a purple circle on the left.
- Edge 3:** Labeled  $L=1; \text{old}=\text{new}$ , leading from the white circle to a white circle below it.
- Edge 4:** Labeled  $L=0; \text{new}++$ , leading from the white circle below to another white circle below it.
- Edge 5:** Labeled  $[\text{new}!=\text{old}]$ , leading from the white circle below to the white circle above it (forming a loop).
- Edge 6:** Labeled  $[\text{new}==\text{old}]$ , leading from the white circle below to a final white circle at the bottom.

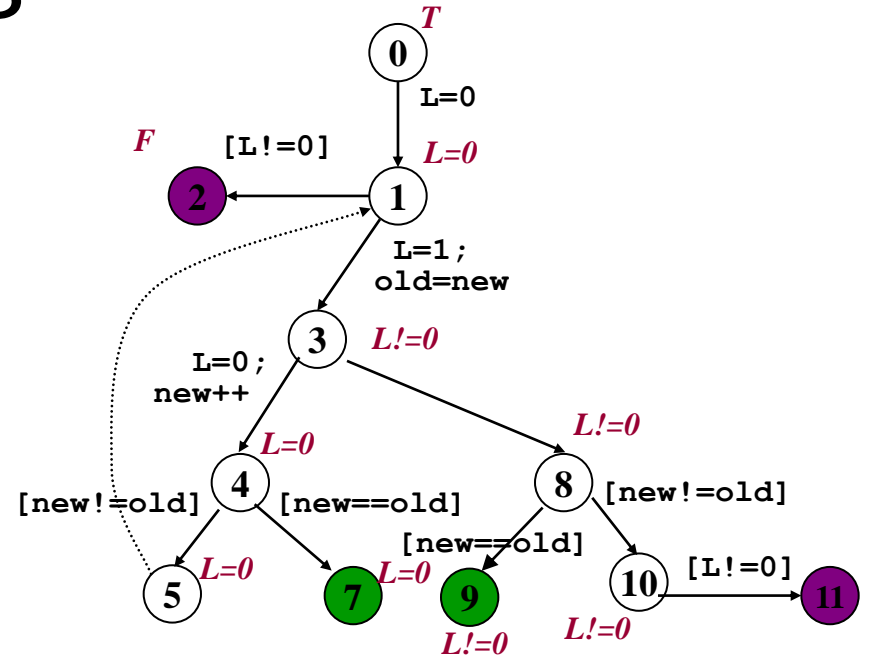
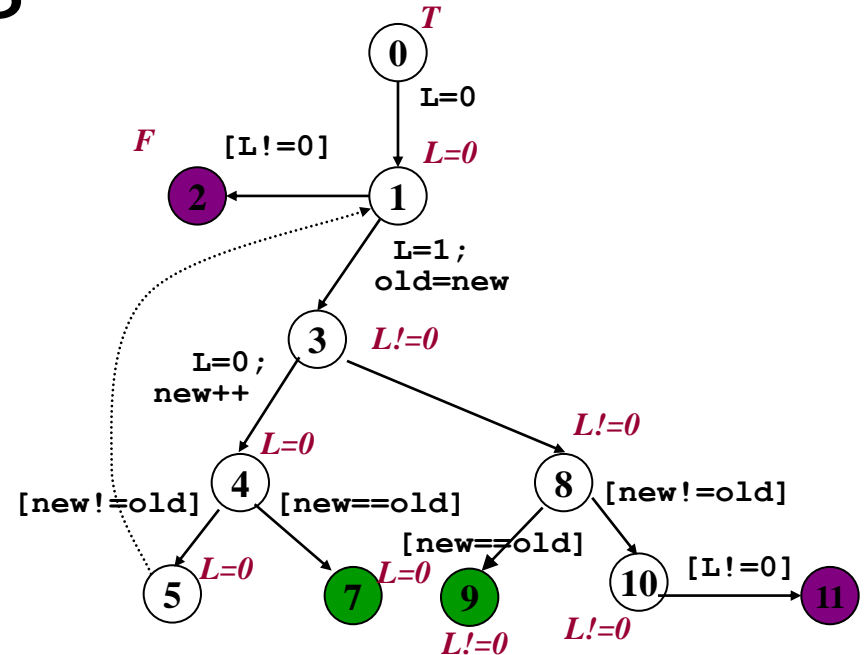
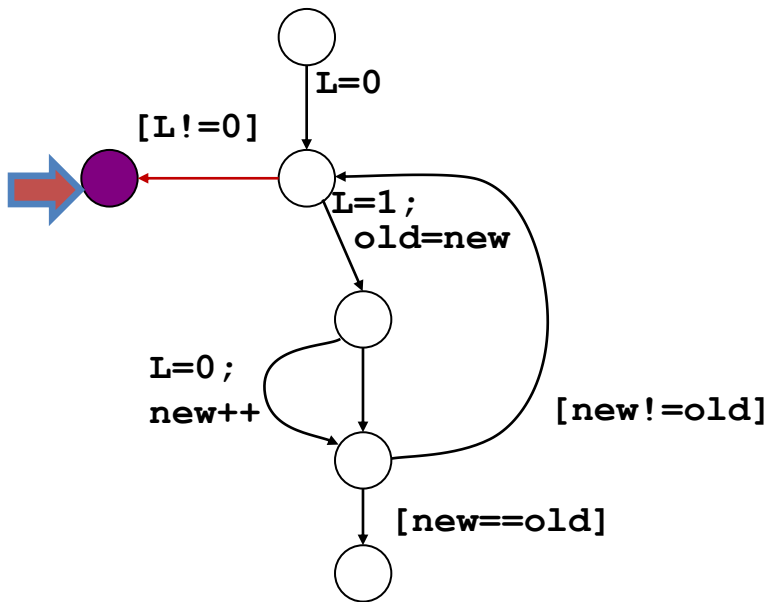
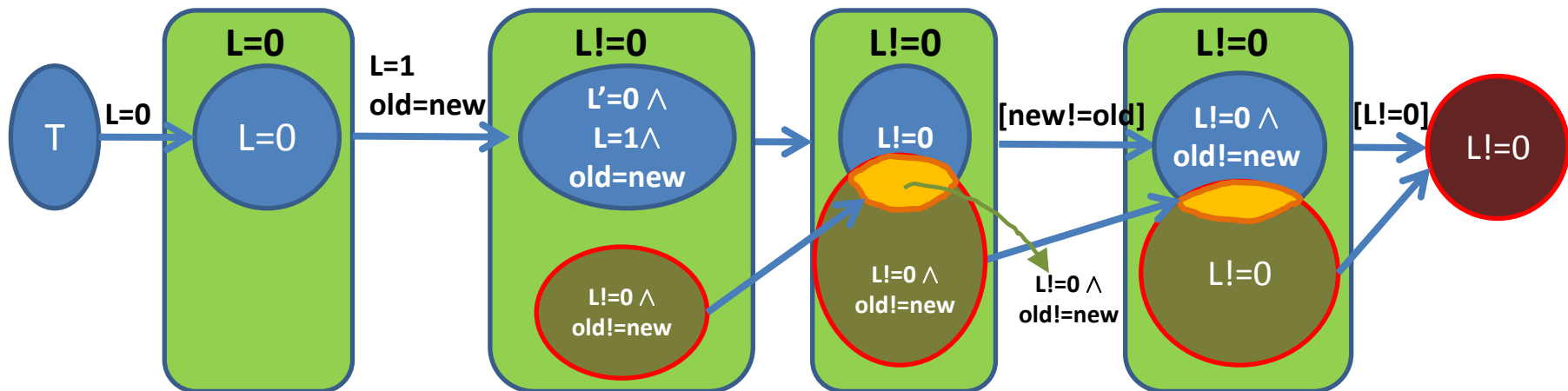


Diagram illustrating the forward reachability analysis for the variable  $L$ . The analysis starts with a node  $T$  (blue oval) labeled  $L=0$ . It transitions to a node (green rounded rectangle) labeled  $L=0$ , which contains a blue oval labeled  $L=0$ . This node transitions to another node (green rounded rectangle) labeled  $L \neq 0$ , which contains a blue oval labeled  $L' = 0 \wedge L = 1 \wedge \text{old} = \text{new}$ . This node transitions to a third node (green rounded rectangle) labeled  $L \neq 0$ , which contains a blue oval labeled  $L \neq 0$  and a red oval labeled  $L \neq 0 \wedge \text{old} = \text{new}$ . This node transitions to a fourth node (green rounded rectangle) labeled  $L \neq 0$ , which contains a blue oval labeled  $L \neq 0 \wedge \text{old} \neq \text{new}$  and a red oval labeled  $L \neq 0$ . This node transitions to a final node (red oval) labeled  $L \neq 0$ . The transitions are labeled with conditions:  $L=0$ ,  $L=1 \text{ old} = \text{new}$ ,  $[\text{new} \neq \text{old}]$ , and  $[L \neq 0]$ . The red ovals represent the summary of the analysis at each step.

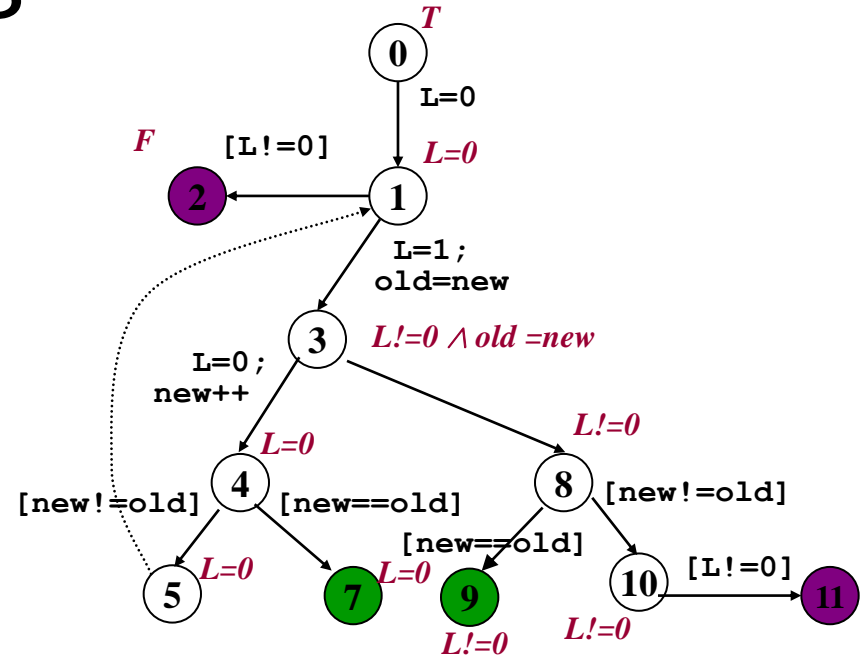
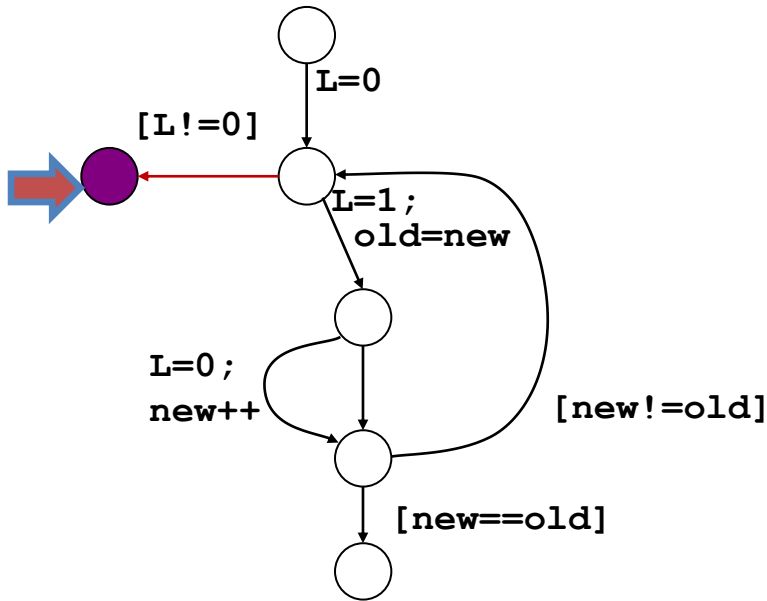
# Unwinding the CFG



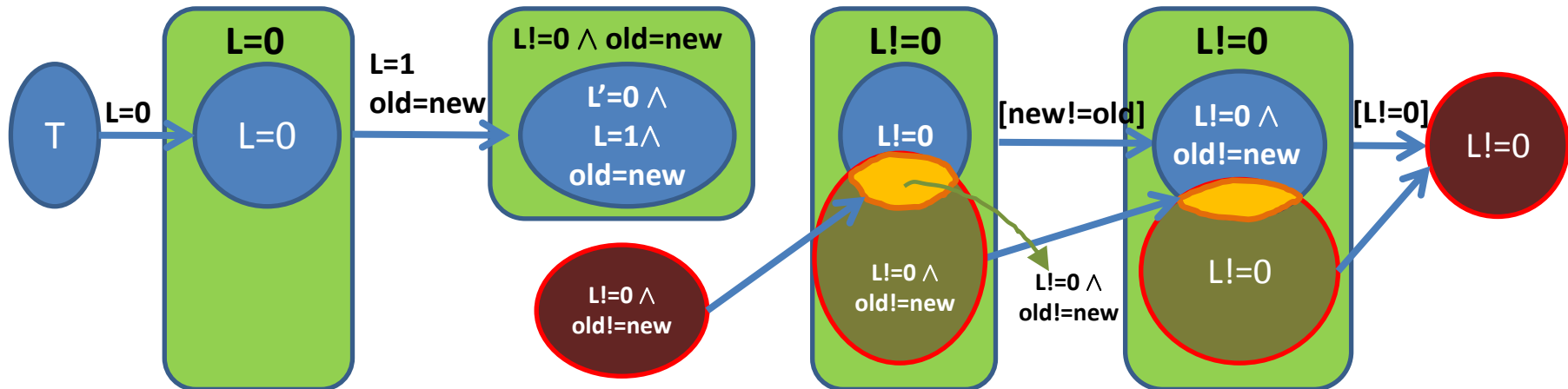
control-flow graph



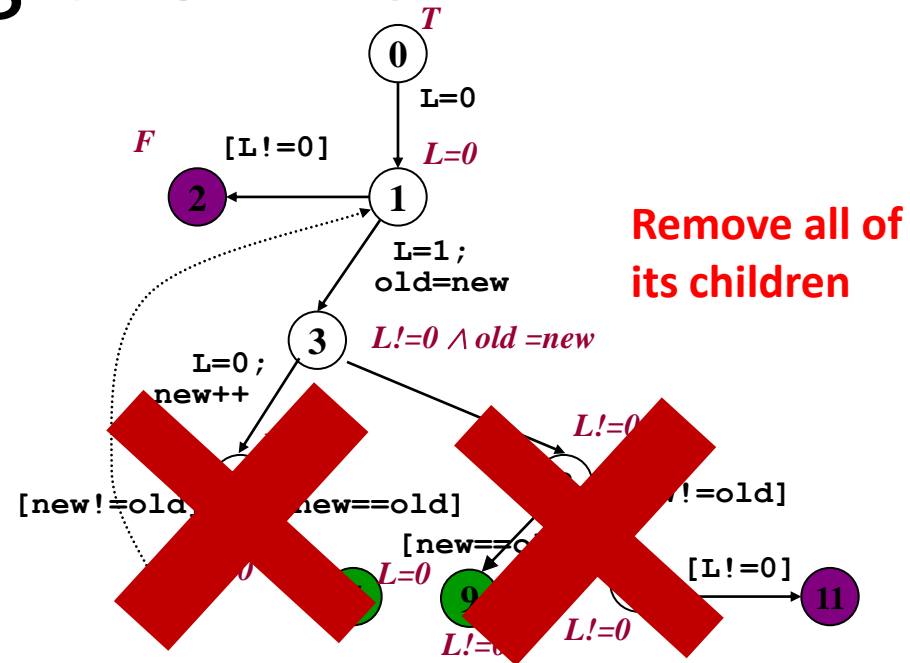
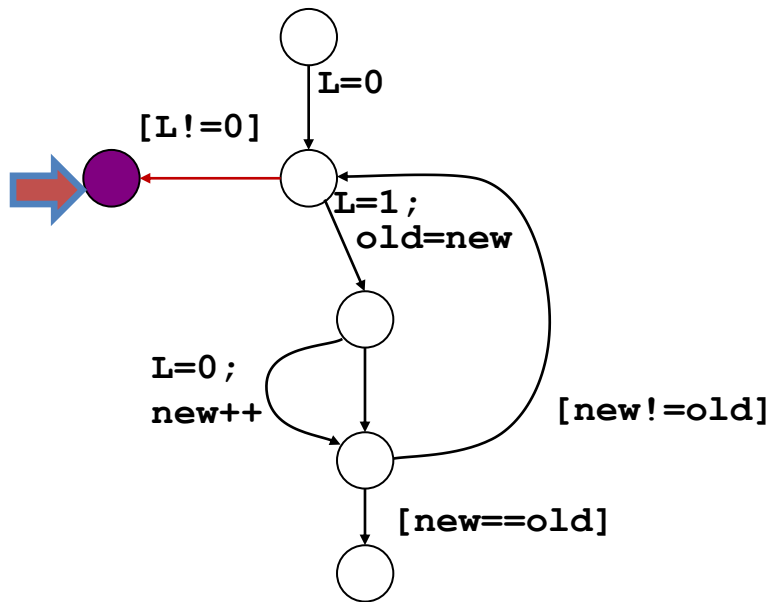
# Unwinding the CFG



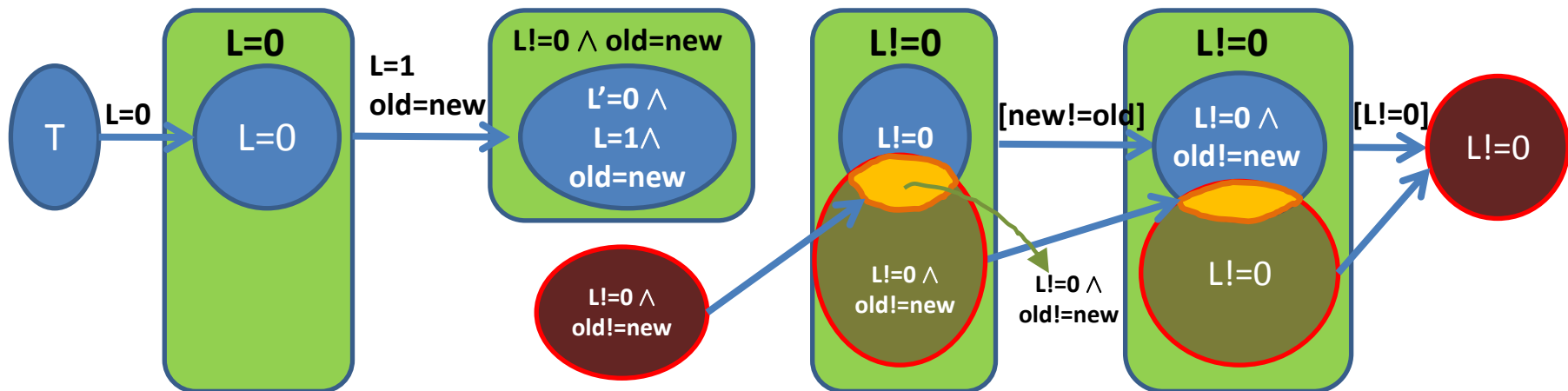
control-flow graph



# Unwinding the CFG

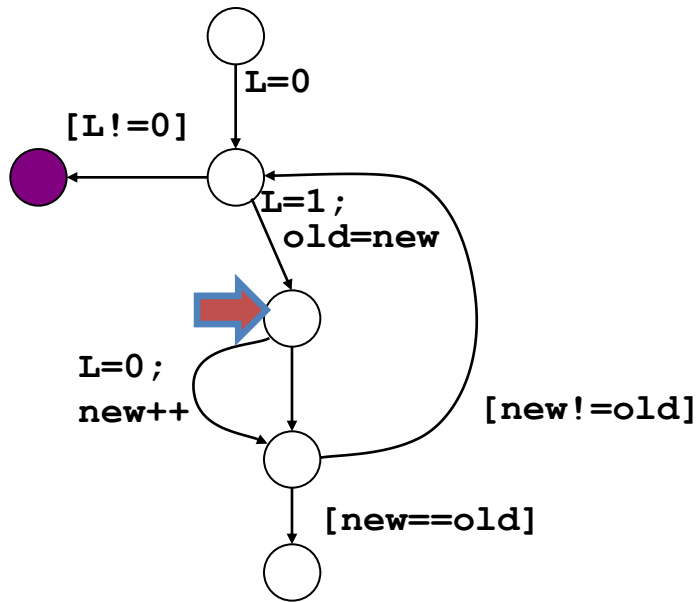


control-flow graph

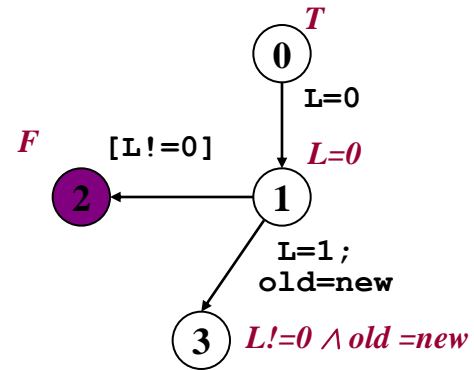




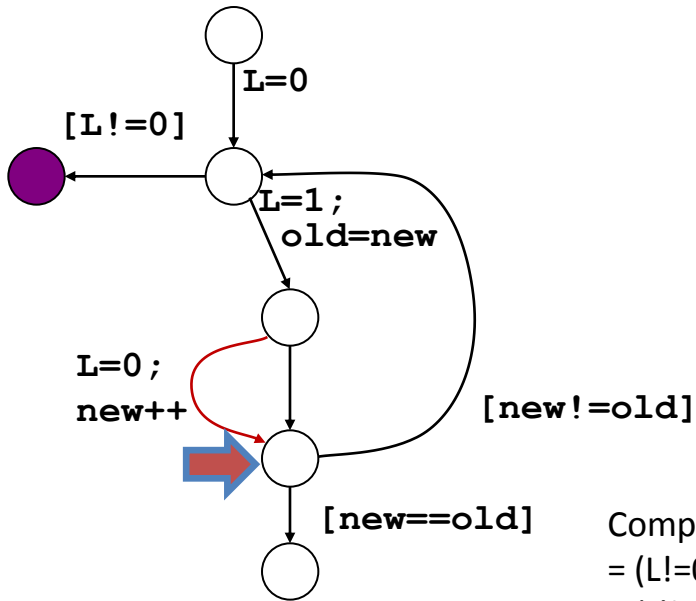
# Unwinding the CFG



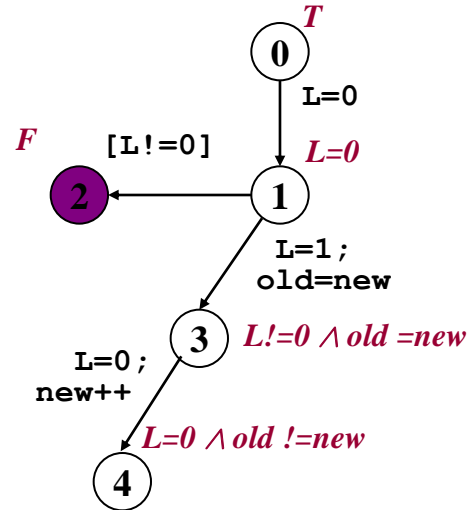
control-flow graph



# Unwinding the CFG



## control-flow graph



Compute Post ( $L \neq 0 \wedge \text{old} = \text{new}, L=0$ )

$$= (L \neq 0 \wedge \text{old} = \text{new})[L/L'] \wedge L = 0[L/L']$$
$$= (L' \neq 0 \wedge \text{old} = \text{new} \wedge L = 0)$$

Compute Post ( $L' \neq 0 \wedge \text{old} = \text{new} \wedge (L = 0)$ ,  $\text{new} = \text{new} + 1$ )

$$= (L' \neq 0 \wedge \text{old} = \text{new} \wedge L = 0)[\text{new}/\text{new}'] \wedge \text{new} = (\text{new} + 1)[\text{new}/\text{new}']$$
$$= (L' \neq 0 \wedge \text{old} = \text{new}' \wedge L = 0 \wedge \text{new} = \text{new}' + 1)$$

## Make Abstraction

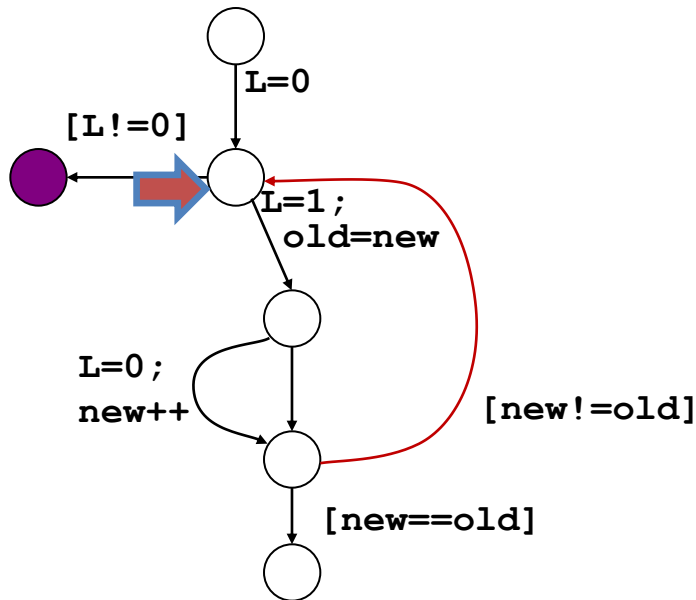
$(L' \neq 0 \wedge \text{old} = \text{new}' \wedge L = 0 \wedge \text{new} = \text{new}' + 1) \rightarrow (L' \neq 0)$  **Not Passed**

$$(L' \neq 0 \wedge \text{old} = \text{new}' \wedge L = 0 \wedge \text{new} = \text{new}' + 1) \rightarrow (L = 0) \quad \text{Pass}$$

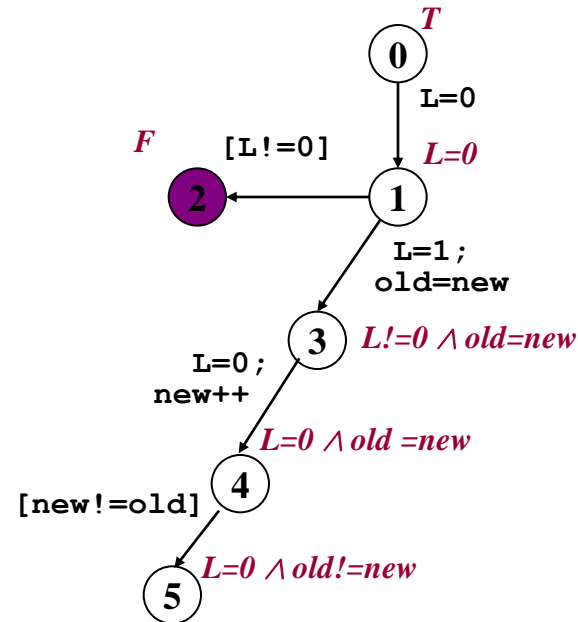
$(L' \neq 0 \wedge \text{old} = \text{new}' \wedge L = 0 \wedge \text{new} = \text{new}' + 1) \rightarrow (\text{old} = \text{new})$  **Not Passed**

$$(L' \neq 0 \wedge \text{old} = \text{new}' \wedge L = 0 \wedge \text{new} = \text{new}' + 1) \rightarrow (\text{old} \neq \text{new}) \text{ Pass}$$

# Unwinding the CFG



control-flow graph



Compute Post ( $L=0 \wedge \text{old}!=\text{new}, [new!=old]$ )

= ( $L=0 \wedge \text{new}!=\text{old}$ )

Make Abstraction

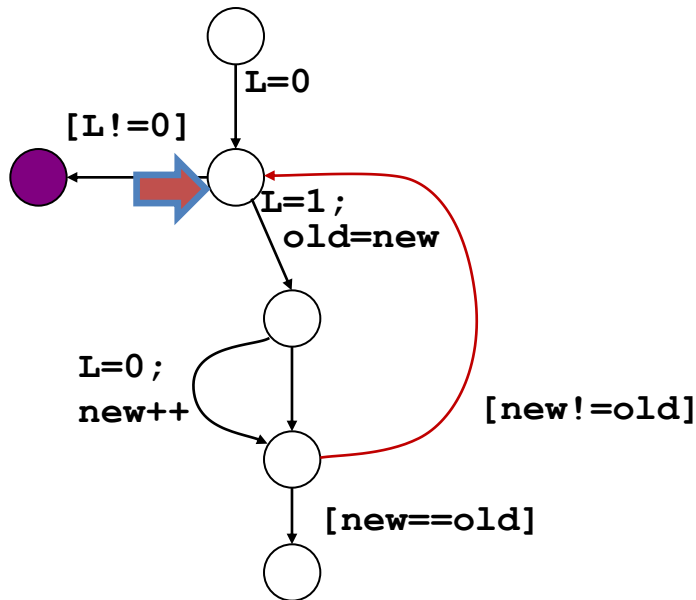
( $L=0 \wedge \text{new}!=\text{old}$ )  $\rightarrow$  ( $L!=0$ ) **Not Passed**

( $L=0 \wedge \text{new}!=\text{old}$ )  $\rightarrow$  ( $L=0$ ) **Pass**

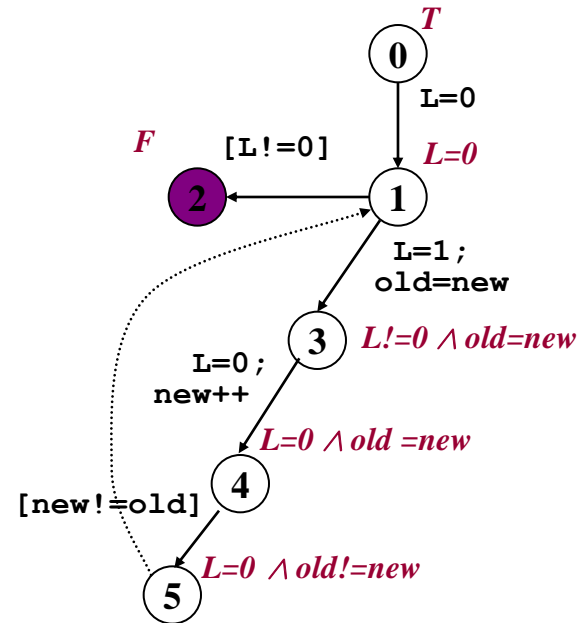
( $L=0 \wedge \text{new}!=\text{old}$ )  $\rightarrow$  ( $\text{old}=\text{new}$ ) **Not Passed**

( $L=0 \wedge \text{new}!=\text{old}$ )  $\rightarrow$  ( $\text{old}!=\text{new}$ ) **Pass**

# Unwinding the CFG



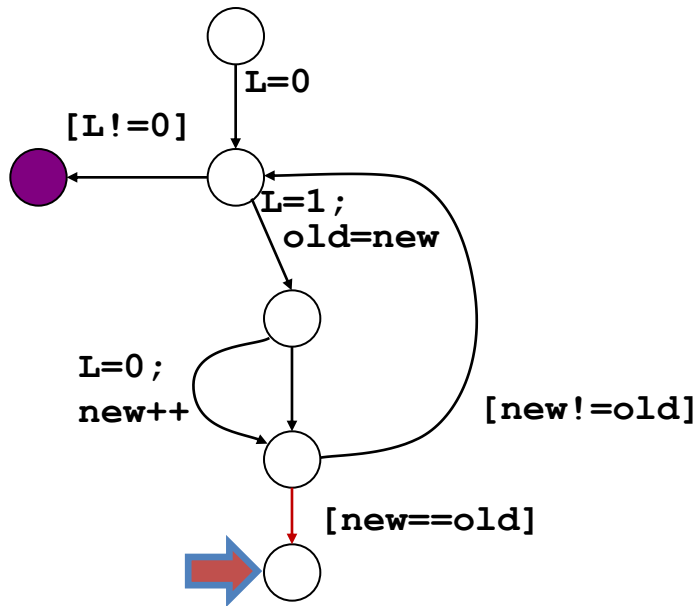
control-flow graph



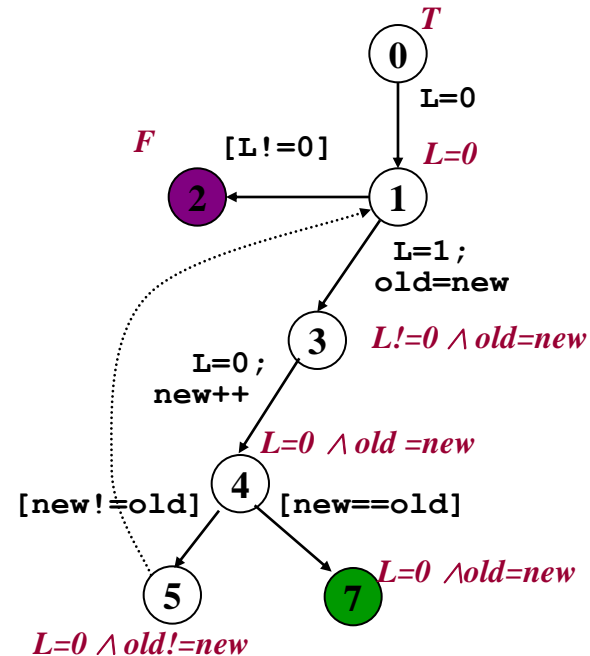
Covering: state 5 is subsumed by state 1.

$L=1 \wedge old!=new \rightarrow L=1$  **Pass**

# Unwinding the CFG



control-flow graph



Compute Post ( $L=0, [new==old]$ )

= ( $L=0 \wedge \text{new}=\text{old}$ )

Make Abstraction

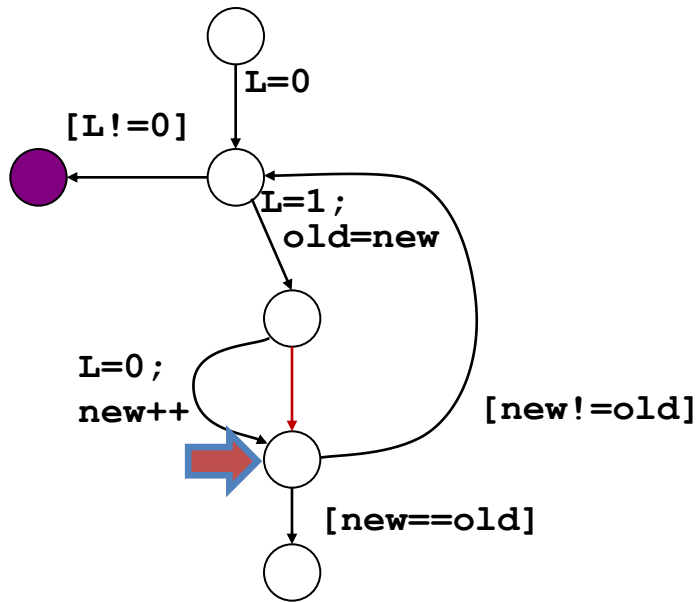
( $L=0 \wedge \text{new}=\text{old}$ )  $\rightarrow$  ( $L!=0$ ) **Not Passed**

( $L=0 \wedge \text{new}=\text{old}$ )  $\rightarrow$  ( $L=0$ ) **Pass**

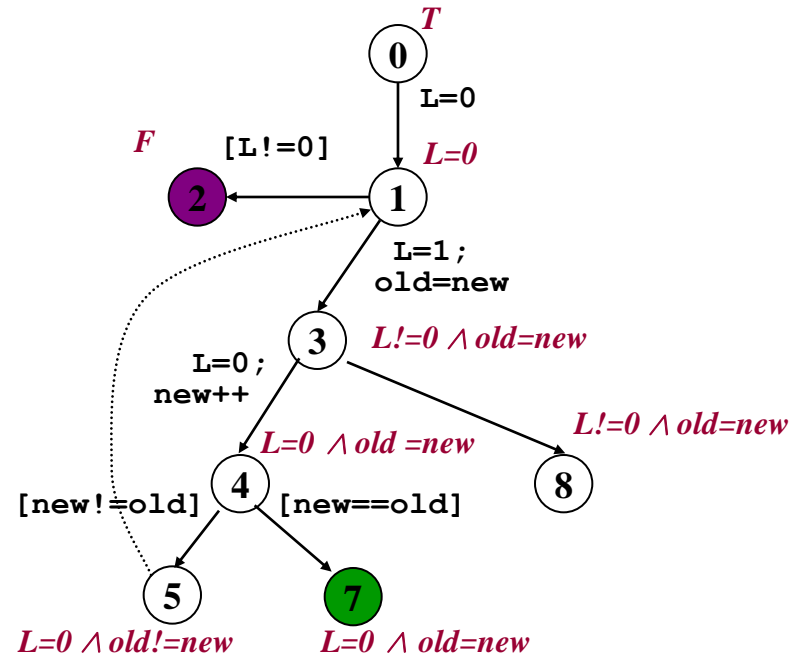
( $L=0 \wedge \text{new}=\text{old}$ )  $\rightarrow$  ( $\text{new}!=\text{old}$ ) **Not Passed**

( $L=0 \wedge \text{new}=\text{old}$ )  $\rightarrow$  ( $\text{new}=\text{old}$ ) **Pass**

# Unwinding the CFG

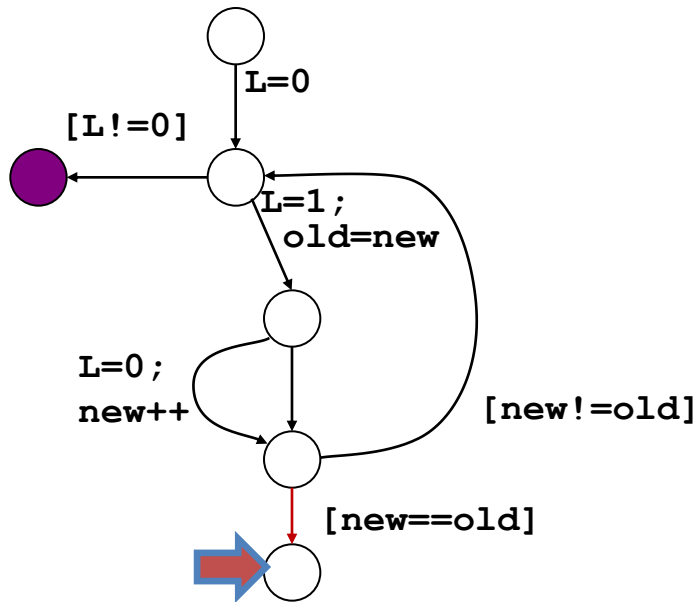


control-flow graph

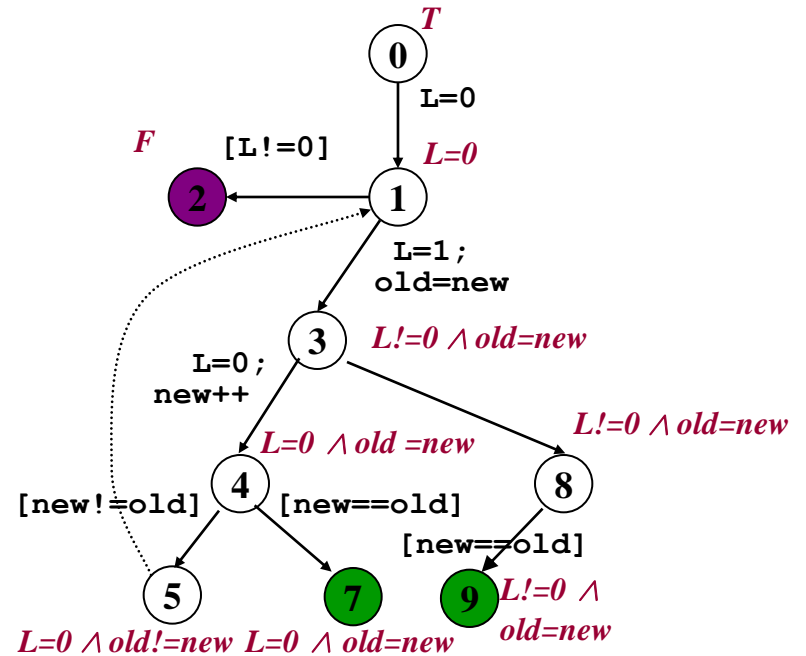


No Actions

# Unwinding the CFG



control-flow graph



Compute Post ( $L!=0 \wedge new=old$ ,  $[new==old]$ )

= ( $L!=0 \wedge new=old$ )

Make Abstraction

$(L!=0 \wedge new=old) \rightarrow (L!=0)$

Pass

$(L!=0 \wedge new=old) \rightarrow (L=0)$

Not Passed

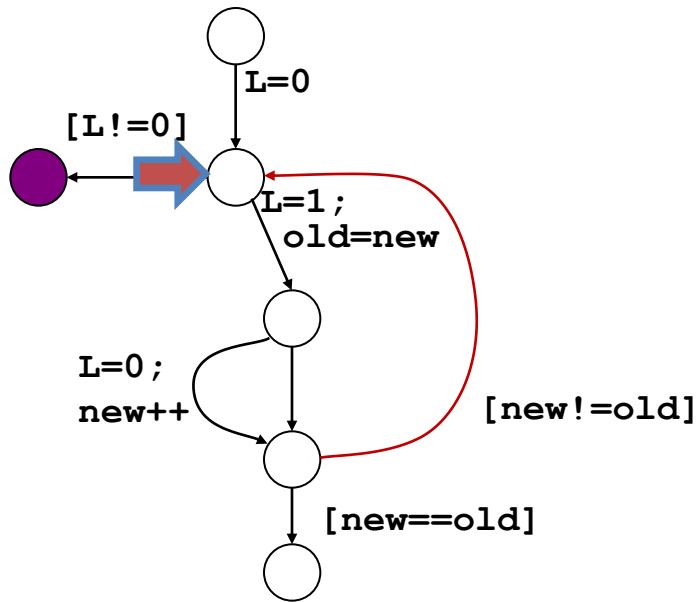
$(L!=0 \wedge new=old) \rightarrow (old=new)$

Pass

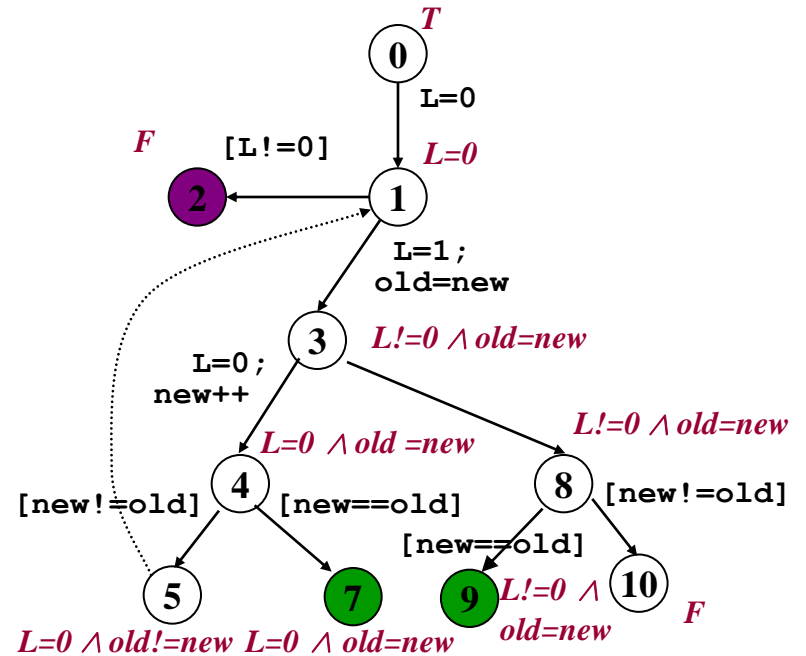
$(L!=0 \wedge new=old) \rightarrow (old!=new)$

Not Passed

# Unwinding the CFG



control-flow graph



Compute Post ( $L \neq 0 \wedge \text{new} = \text{old}, [\text{new} \neq \text{old}]$ )  
 $= (L \neq 0 \wedge \text{new} = \text{old} \wedge \text{new} \neq \text{old})$   
 $= \text{false}$



# Another Approach: The IMPACT method

Kenneth L. McMillan: Lazy Abstraction with Interpolants. CAV 2006: 123-136

# Interpolation Lemma

(Craig,57)

# Interpolation Lemma (Craig,57)

- Notation:  $\mathcal{L}(\phi)$  is the set of FO formulas over the symbols of  $\phi$

# Interpolation Lemma (Craig,57)

- Notation:  $\mathcal{L}(\phi)$  is the set of FO formulas over the symbols of  $\phi$
- If  $A \wedge B = \text{false}$ , there exists an *interpolant*  $A'$  for  $(A,B)$  such that:

$$A \Rightarrow A'$$

$$A' \wedge B = \text{false}$$

$$A' \in \mathcal{L}(A) \cap \mathcal{L}(B)$$

# Interpolation Lemma (Craig,57)

- Notation:  $\mathcal{L}(\phi)$  is the set of FO formulas over the symbols of  $\phi$
- If  $A \wedge B = \text{false}$ , there exists an *interpolant*  $A'$  for  $(A,B)$  such that:

$$A \Rightarrow A'$$

$$A' \wedge B = \text{false}$$

$$A' \in \mathcal{L}(A) \cap \mathcal{L}(B)$$

- Example:
  - $A = p \wedge q, \quad B = \neg q \wedge r, \quad A' = q$

# Interpolation Lemma (Craig,57)

- Notation:  $\mathcal{L}(\phi)$  is the set of FO formulas over the symbols of  $\phi$
- If  $A \wedge B = \text{false}$ , there exists an *interpolant*  $A'$  for  $(A,B)$  such that:

$$\begin{aligned} A &\Rightarrow A' \\ A' \wedge B &= \text{false} \\ A' &\in \mathcal{L}(A) \cap \mathcal{L}(B) \end{aligned}$$

- Example:
  - $A = p \wedge q, \quad B = \neg q \wedge r, \quad A' = q$
- Interpolants from proofs
  - in certain quantifier-free theories, we can obtain an interpolant for a pair  $A,B$  from a refutation in linear time. [McMillan 05]
  - in particular, we can have linear arithmetic, uninterpreted functions, and restricted use of arrays

# Interpolants for sequences

# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas



# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when

# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$

# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$

# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$
  - $A'_n = \text{False}$

# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$
  - $A'_n = \text{False}$
  - and finally,  $A'_i \in \mathcal{L}(A_1 \dots A_i) \cap \mathcal{L}(A_{i+1} \dots A_n)$

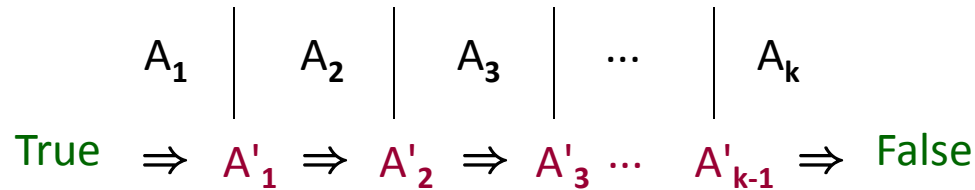
# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$
  - $A'_n = \text{False}$
  - and finally,  $A'_i \in \mathcal{L}(A_1 \dots A_i) \cap \mathcal{L}(A_{i+1} \dots A_n)$

$A_1 \quad A_2 \quad A_3 \quad \dots \quad A_k$

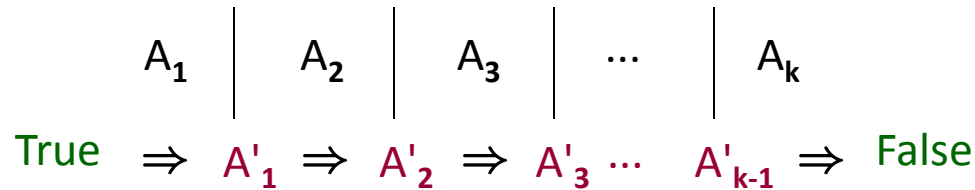
# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$
  - $A_n = \text{False}$
  - and finally,  $A'_i \in \mathcal{L}(A_1 \dots A_i) \cap \mathcal{L}(A_{i+1} \dots A_n)$



# Interpolants for sequences

- Let  $A_1 \dots A_n$  be a sequence of formulas
- A sequence  $A'_0 \dots A'_n$  is an interpolant for  $A_1 \dots A_n$  when
  - $A'_0 = \text{True}$
  - $A'_{i-1} \wedge A_i \Rightarrow A'_i$ , for  $i = 1..n$
  - $A_n = \text{False}$
  - and finally,  $A'_i \in \mathcal{L}(A_1 \dots A_i) \cap \mathcal{L}(A_{i+1} \dots A_n)$

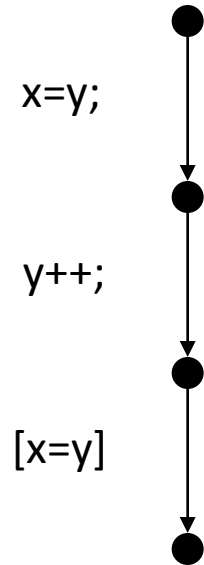


In other words, the interpolant is a structured refutation of  $A_1 \dots A_n$

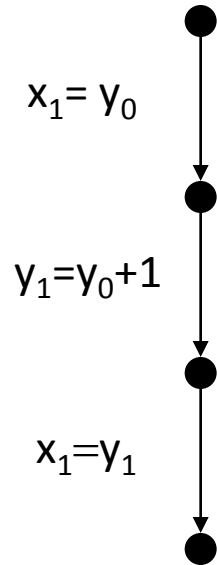


# Interpolants as Floyd-Hoare proofs

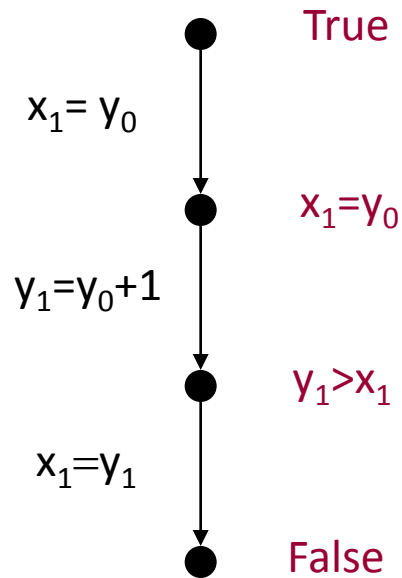
# Interpolants as Floyd-Hoare proofs



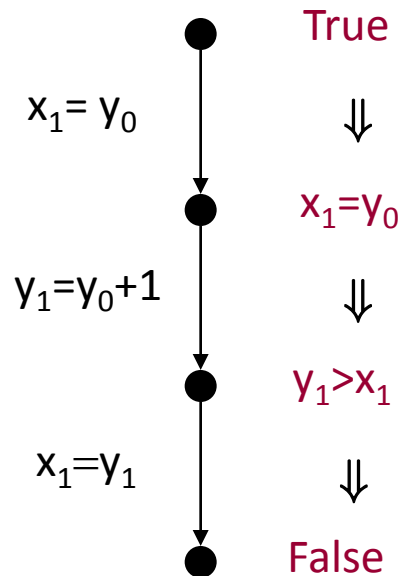
# Interpolants as Floyd-Hoare proofs



# Interpolants as Floyd-Hoare proofs

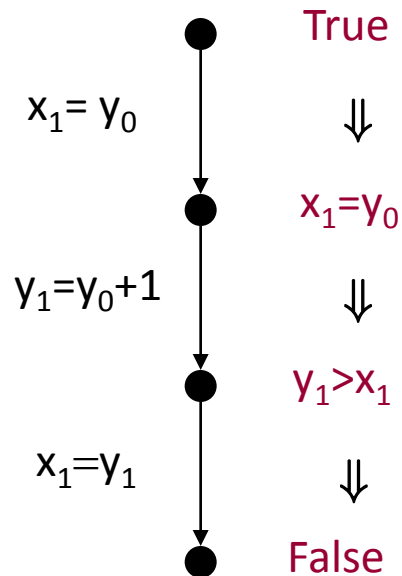


# Interpolants as Floyd-Hoare proofs



1. Each formula implies the next

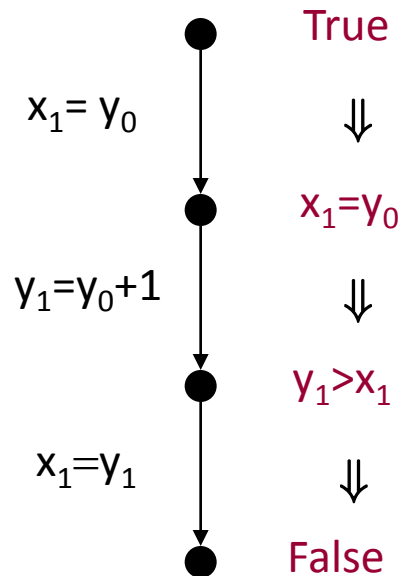
# Interpolants as Floyd-Hoare proofs



1. Each formula implies the next

2. Each is over common symbols of prefix and suffix

# Interpolants as Floyd-Hoare proofs

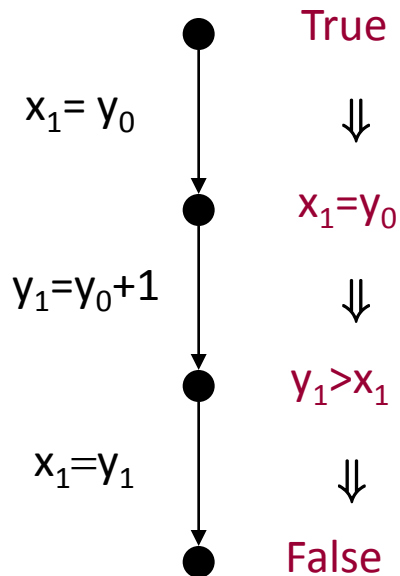


1. Each formula implies the next

2. Each is over common symbols of prefix and suffix

3. Begins with true, ends with false

# Interpolants as Floyd-Hoare proofs

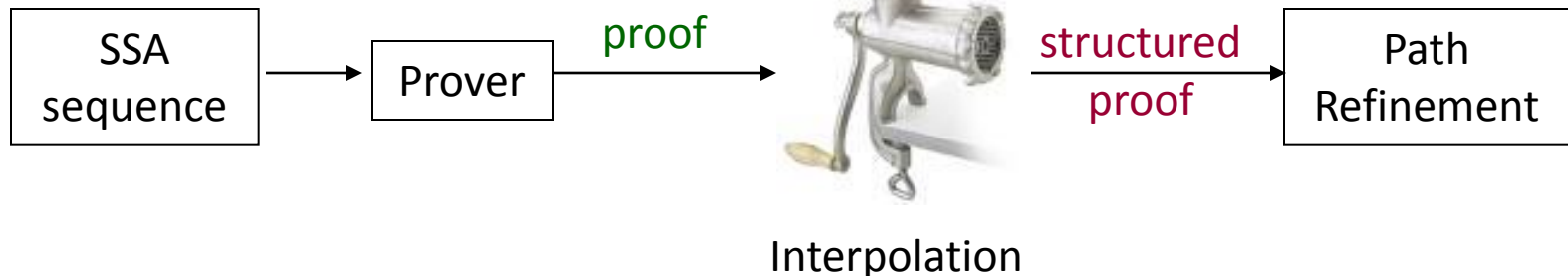


1. Each formula implies the next

2. Each is over common symbols of prefix and suffix

3. Begins with true, ends with false

Path refinement procedure





# Lazy abstraction -- an example

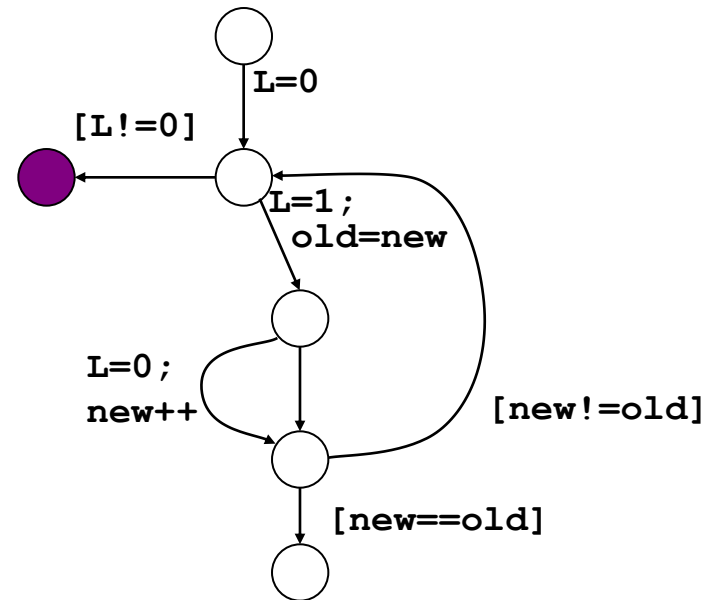
```
do{  
    lock() ;  
    old = new;  
    if(*){  
        unlock;  
        new++;  
    }  
} while (new != old);
```

program fragment

# Lazy abstraction -- an example

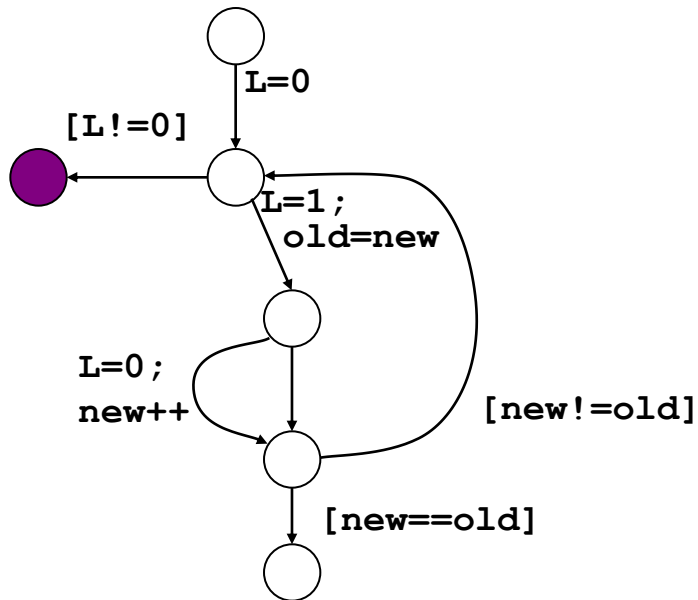
```
do{  
    lock();  
    old = new;  
    if(*){  
        unlock;  
        new++;  
    }  
} while (new != old);
```

program fragment



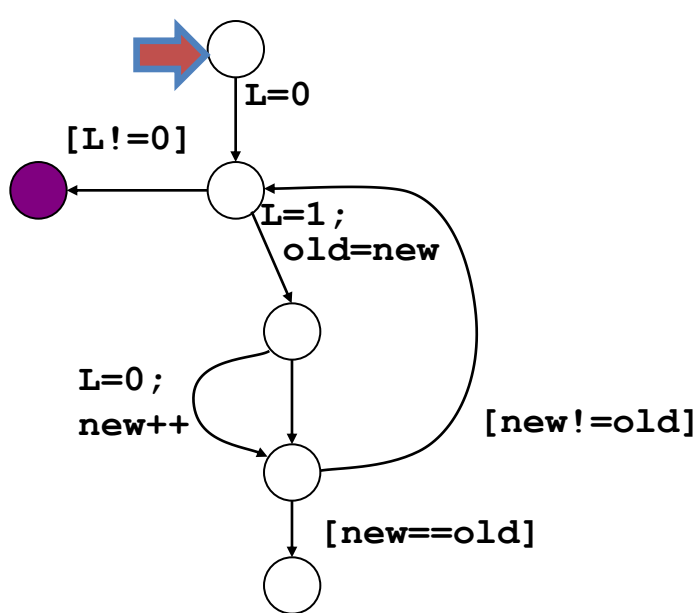
control-flow graph

# Unwinding the CFG



control-flow graph

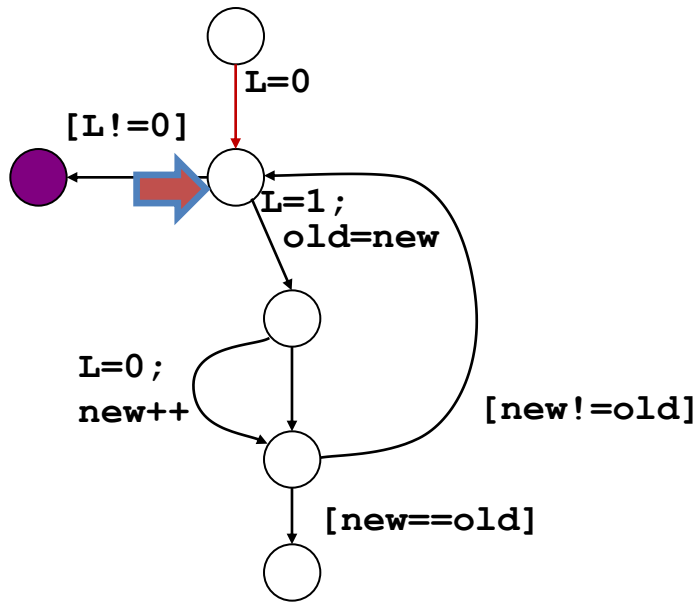
# Unwinding the CFG



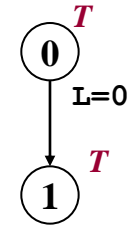
control-flow graph

$\textcircled{0}^T$

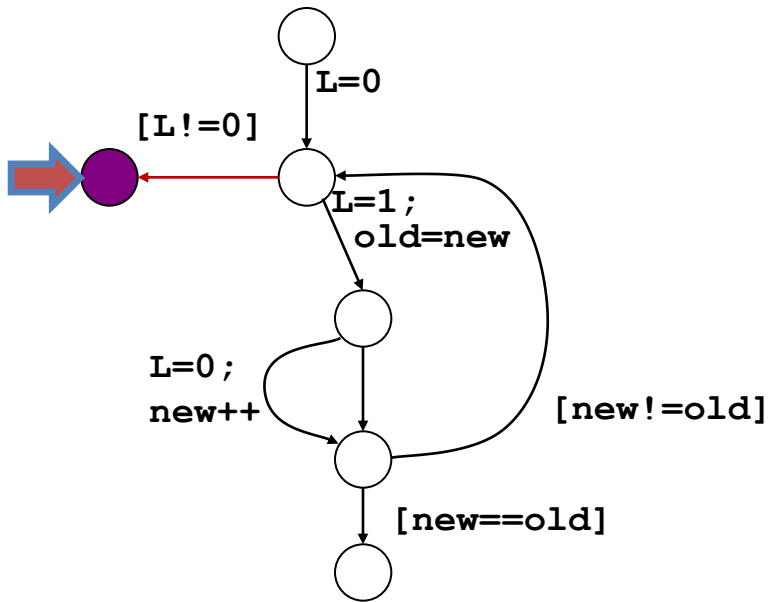
# Unwinding the CFG



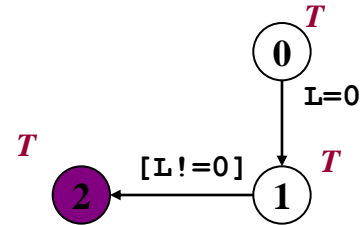
control-flow graph



# Unwinding the CFG

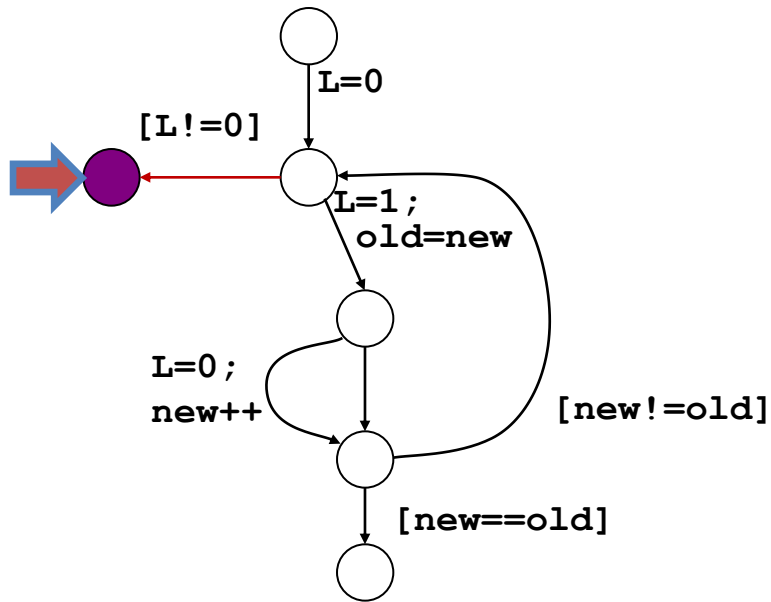


control-flow graph

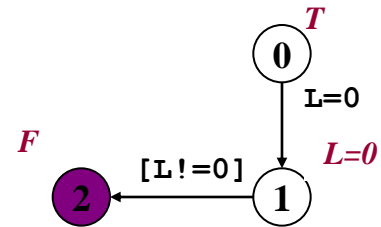


Interpolant for  $(L_0 = 0) \wedge (L_0 \neq 0) = ?$

# Unwinding the CFG



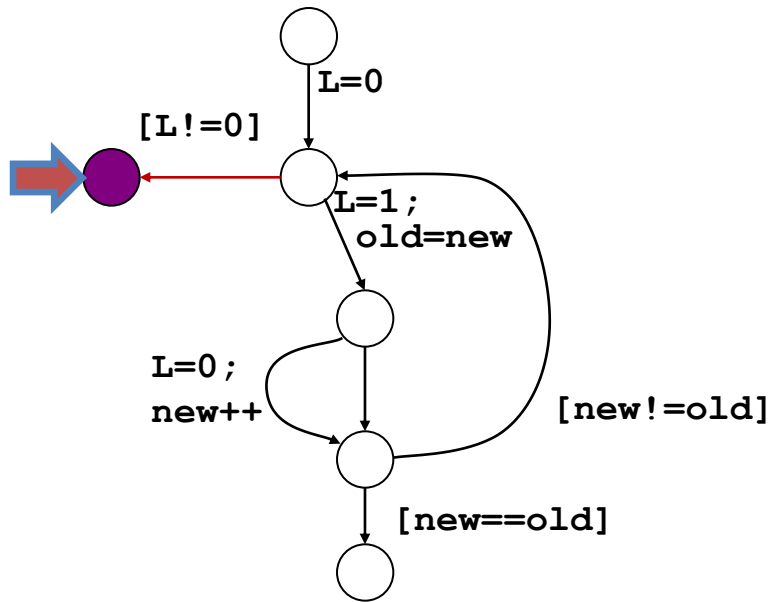
control-flow graph



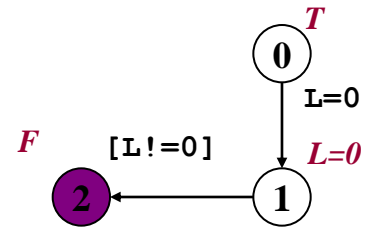
Label error state with false, by refining labels on path

Interpolant for  $(L_0 = 0) \wedge (L_0 \neq 0) = ?$   
 $T \quad L_0 = 0 \quad F$

# Unwinding the CFG

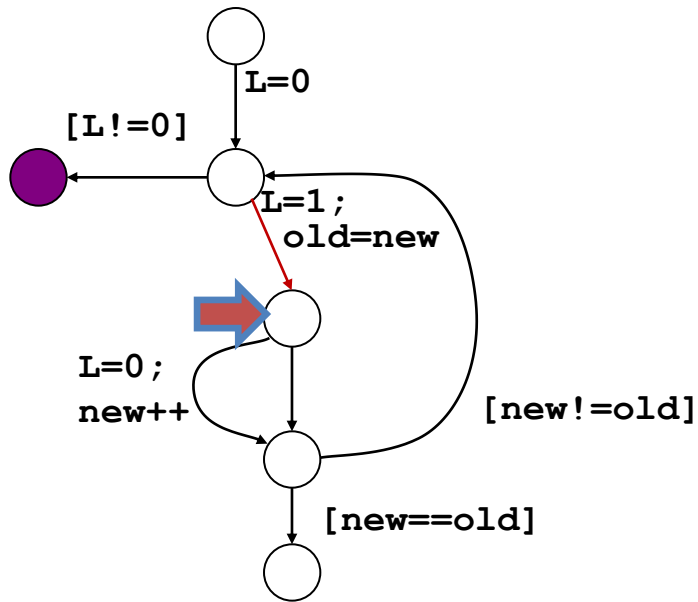


control-flow graph

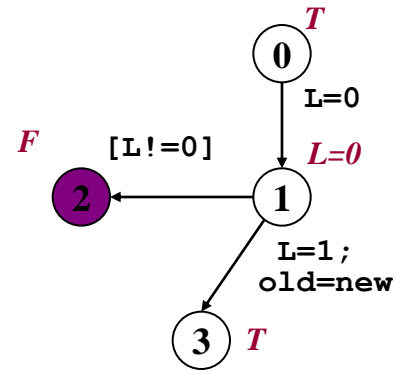




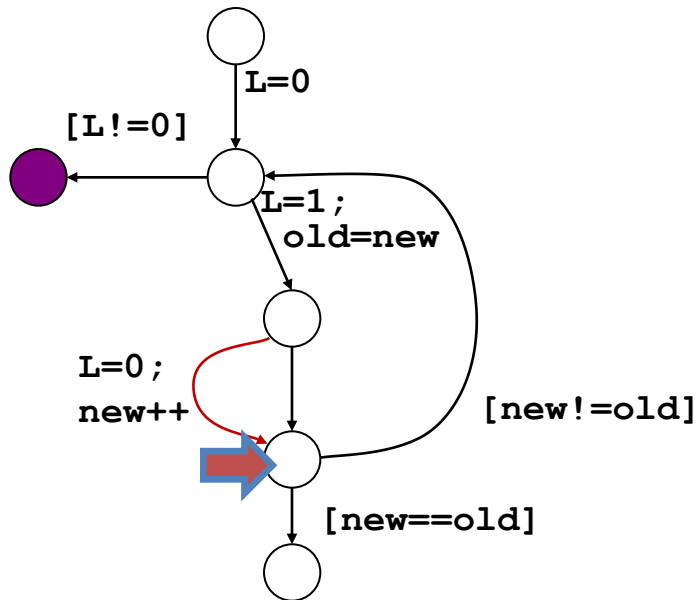
# Unwinding the CFG



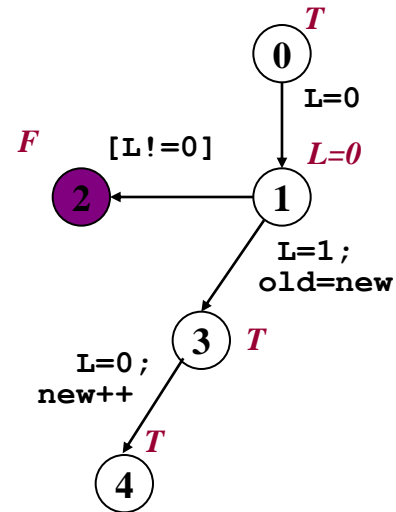
control-flow graph



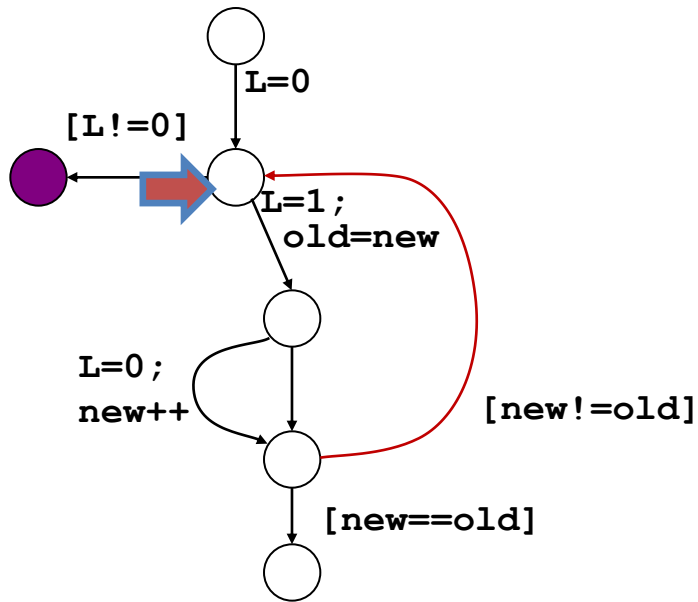
# Unwinding the CFG



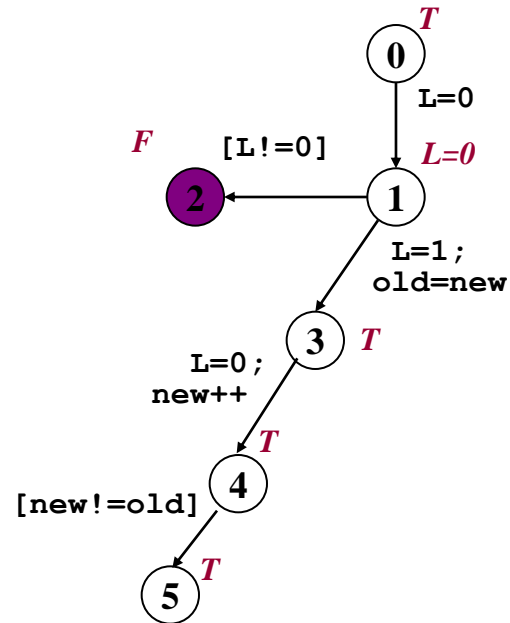
control-flow graph



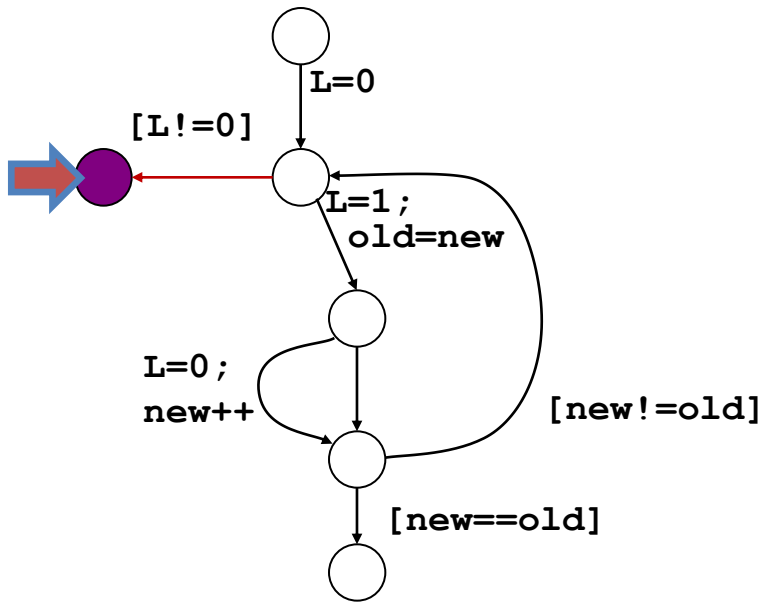
# Unwinding the CFG



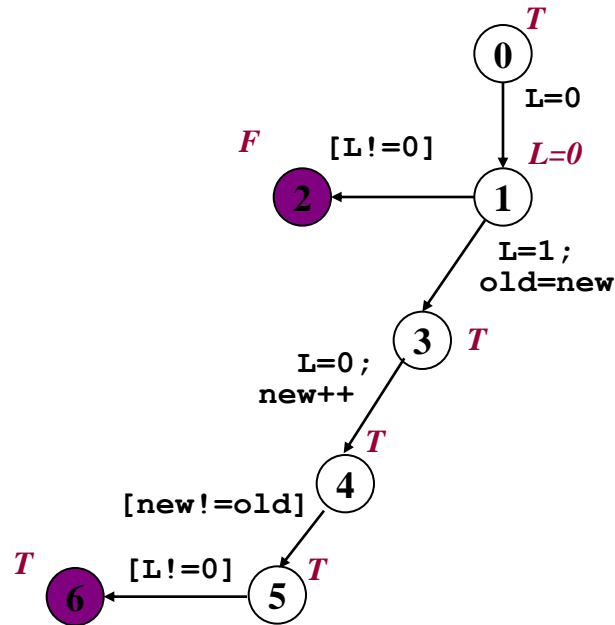
control-flow graph



# Unwinding the CFG

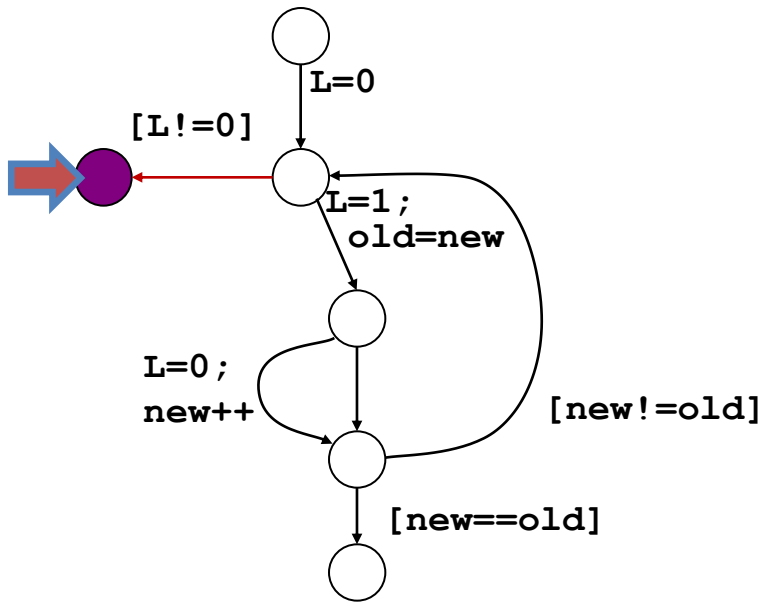


control-flow graph

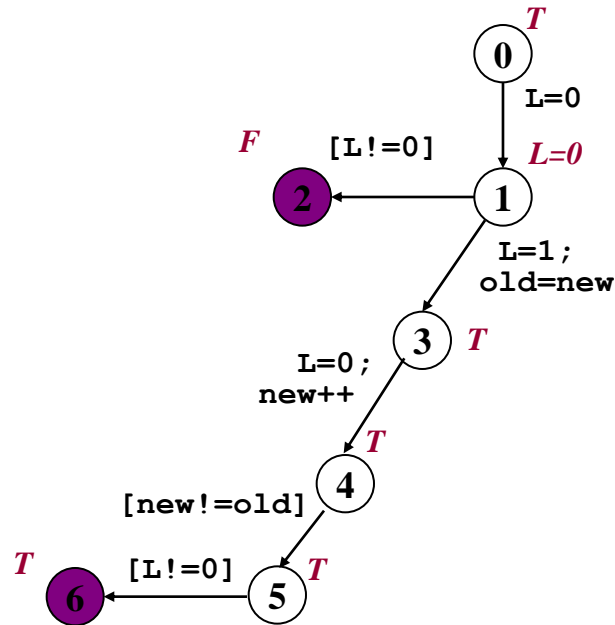


Interpolant for  $(L_0 = 0) \wedge (L_1 = 1 \wedge \text{old}_0 = \text{new}_0) \wedge (L_2 = 0 \wedge \text{new}_1 = \text{new}_0 + 1) \wedge (\text{new}_1 \neq \text{old}_0) \wedge (L_2 \neq 0) = ?$

# Unwinding the CFG



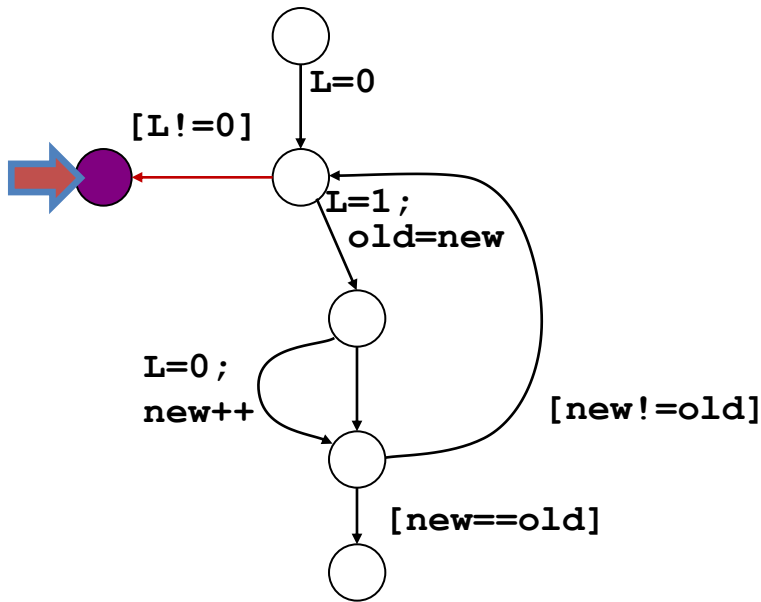
control-flow graph



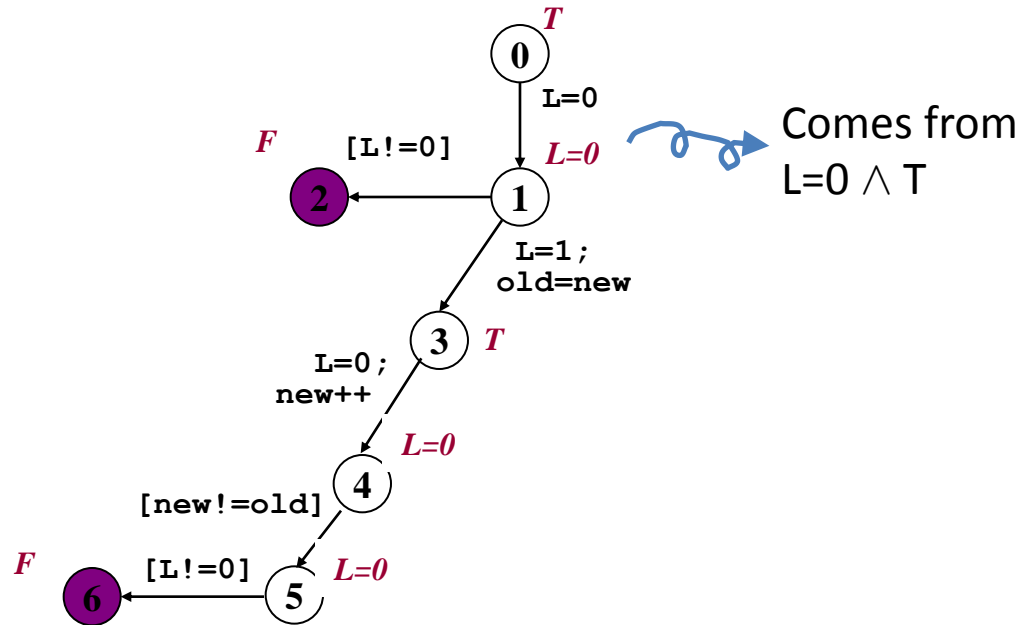
Interpolant for  $(L_0 = 0) \wedge (L_1 = 1 \wedge \text{old}_0 = \text{new}_0) \wedge (L_2 = 0 \wedge \text{new}_1 = \text{new}_0 + 1) \wedge (\text{new}_1 \neq \text{old}_0) \wedge (L_2 \neq 0) = ?$

$T$ 
 $T$ 
 $T$ 
 $L_2 = 0$ 
 $L_2 = 0$ 
 $F$

# Unwinding the CFG



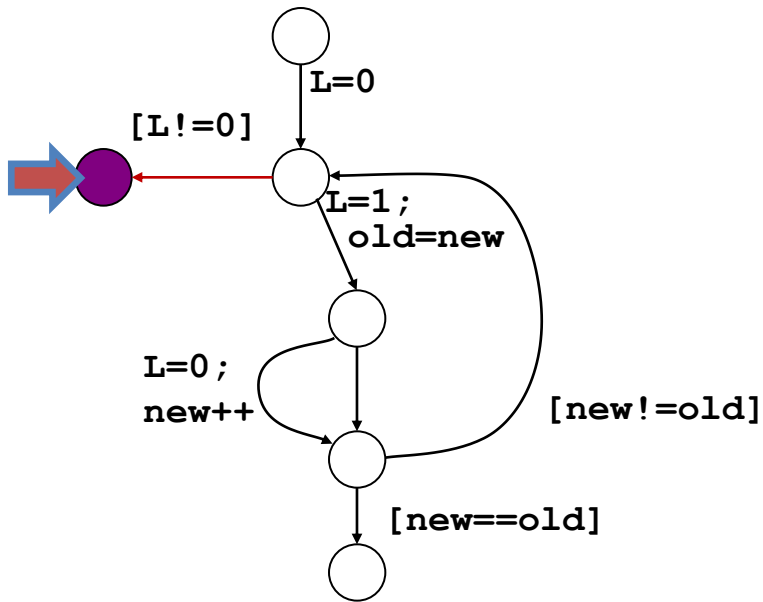
control-flow graph



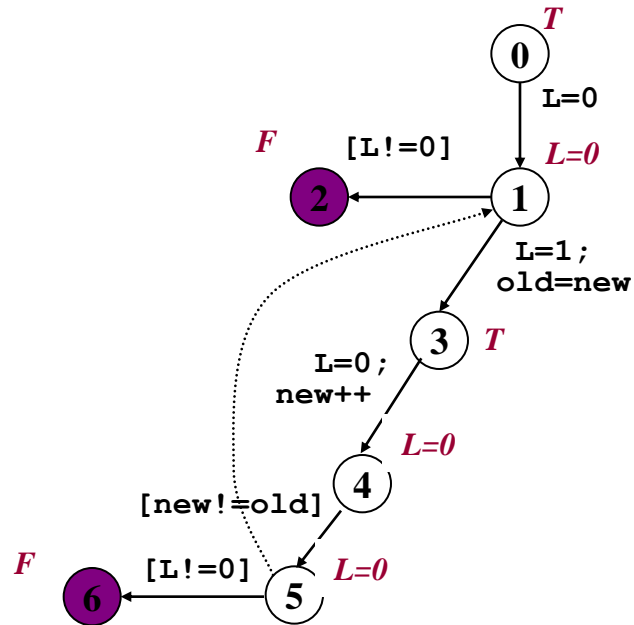
Interpolant for  $(L_0 = 0) \wedge (L_1 = 1 \wedge \text{old}_0 = \text{new}_0) \wedge (L_2 = 0 \wedge \text{new}_1 = \text{new}_0 + 1) \wedge (\text{new}_1 \neq \text{old}_0) \wedge (L_2 \neq 0) = ?$

$T$      $T$      $T$      $L_2 = 0$      $L_2 = 0$      $F$

# Unwinding the CFG

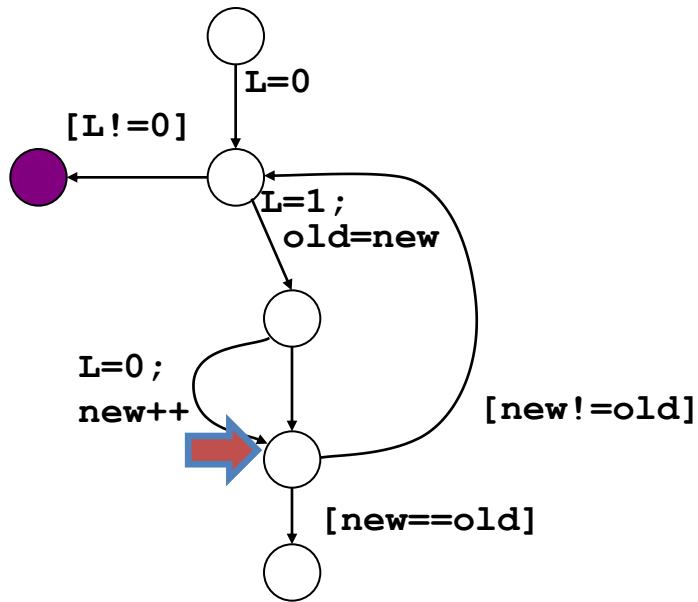


control-flow graph

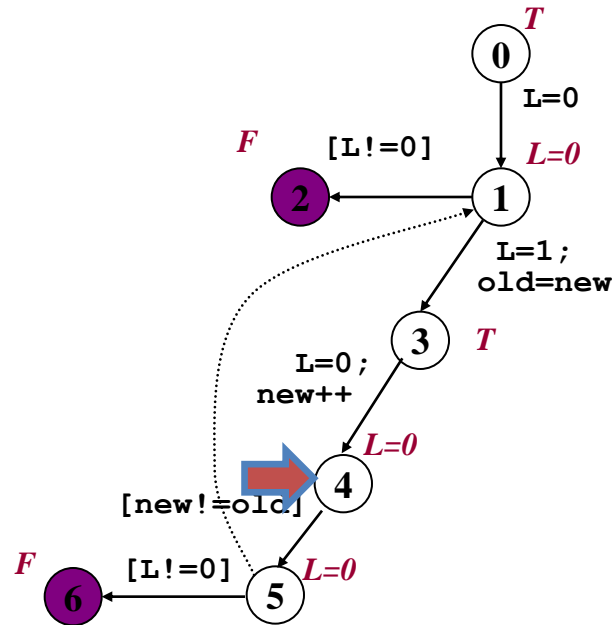


Covering: state 5 is subsumed by state 1.

# Unwinding the CFG

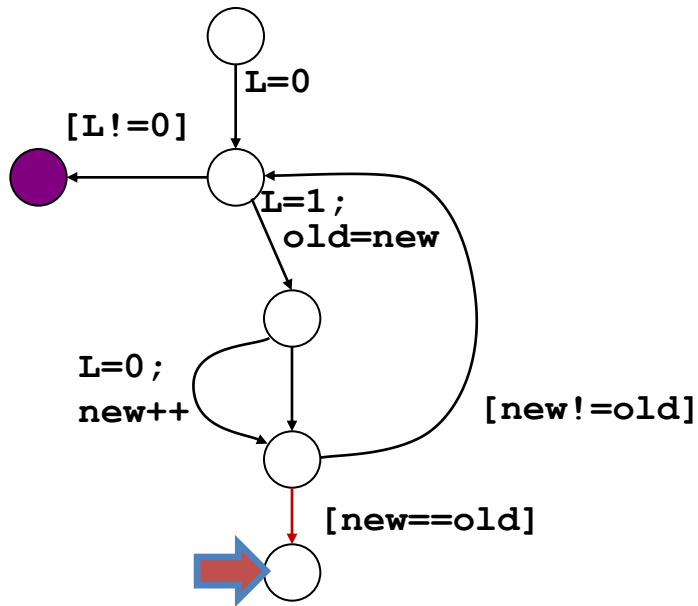


control-flow graph

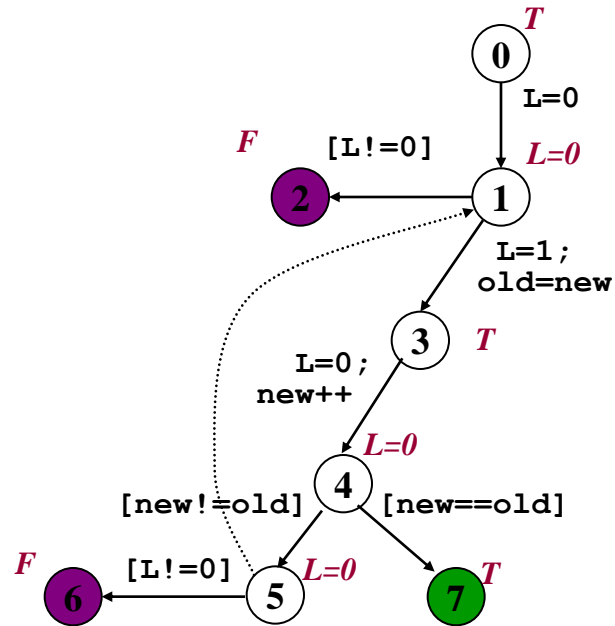




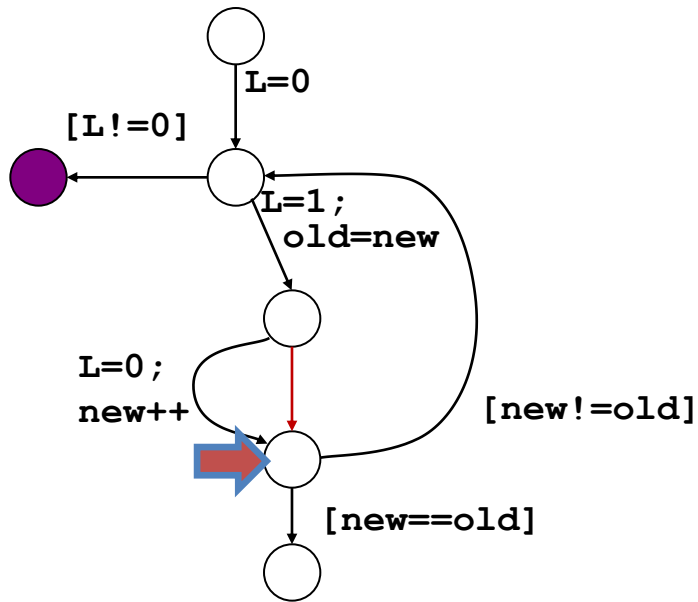
# Unwinding the CFG



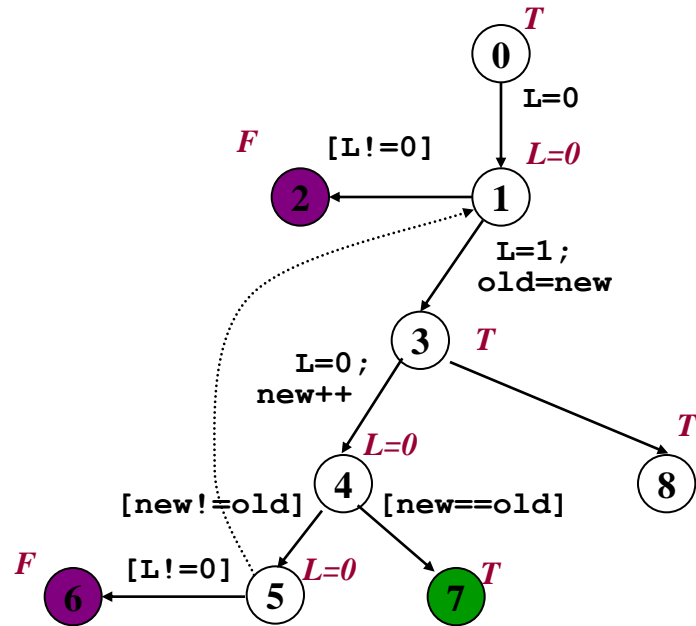
control-flow graph



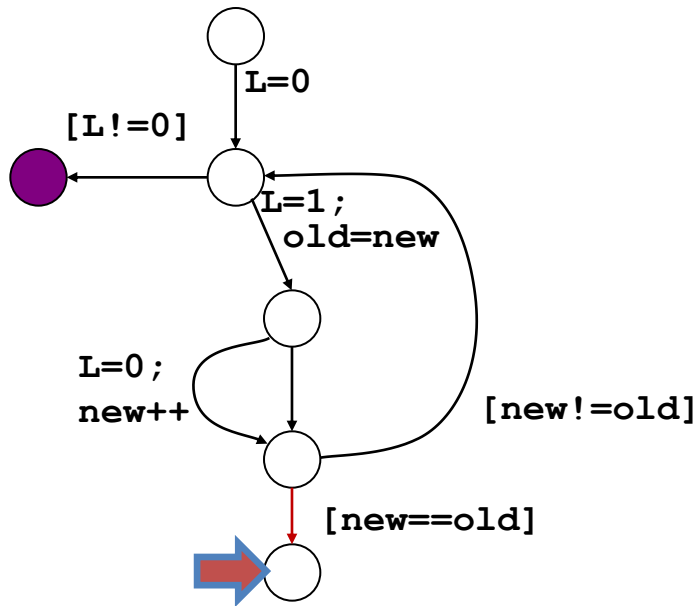
# Unwinding the CFG



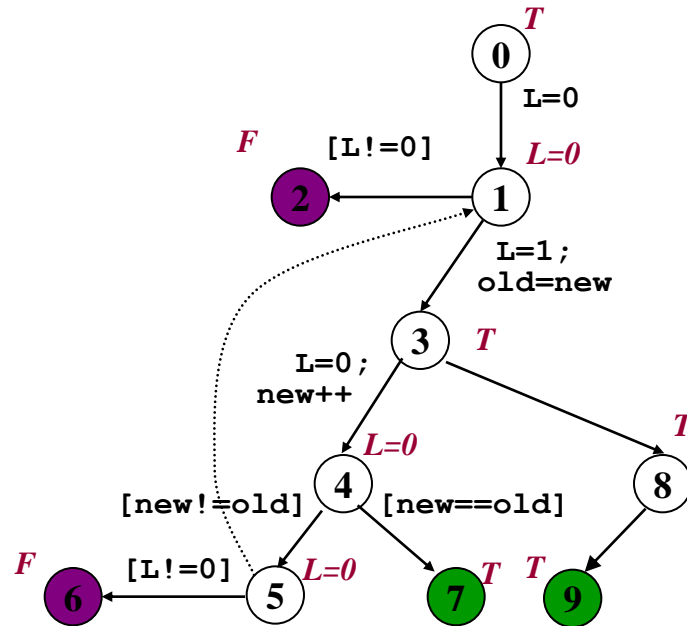
control-flow graph



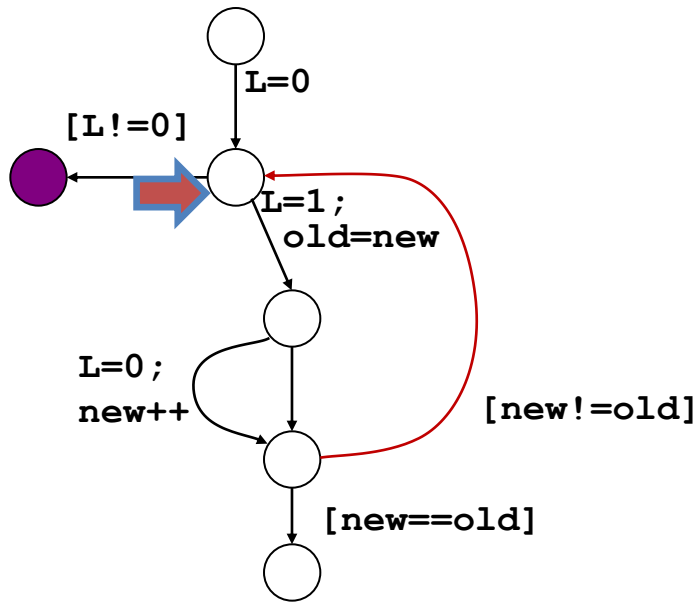
# Unwinding the CFG



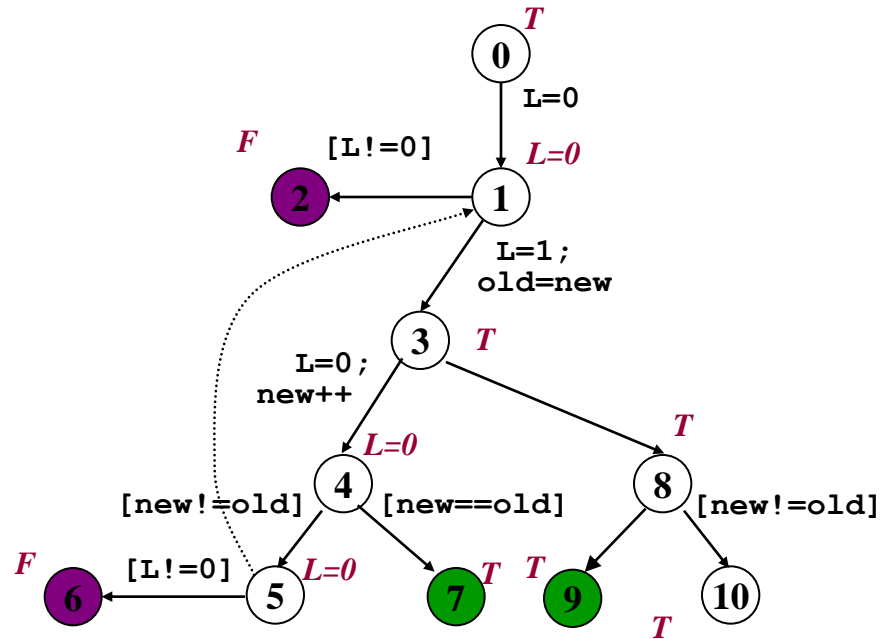
control-flow graph



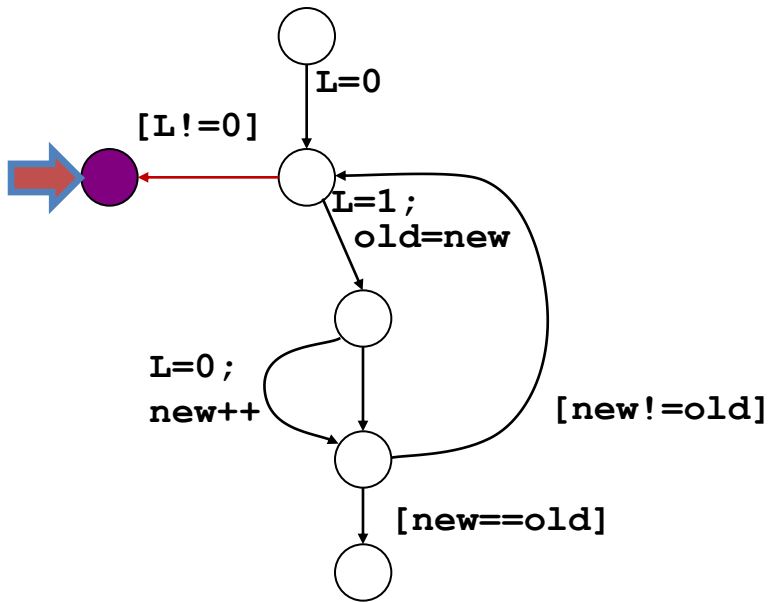
# Unwinding the CFG



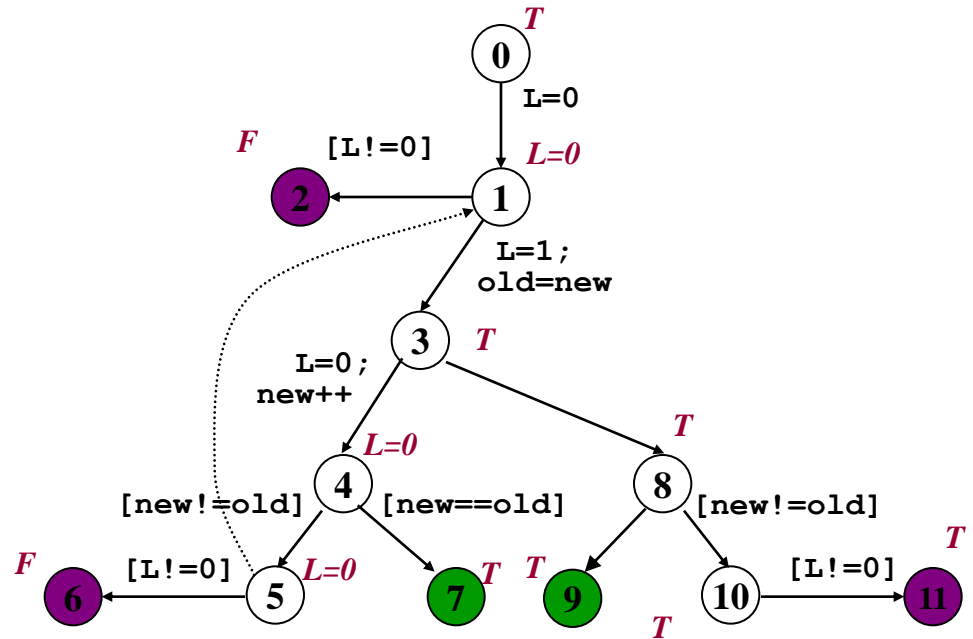
control-flow graph



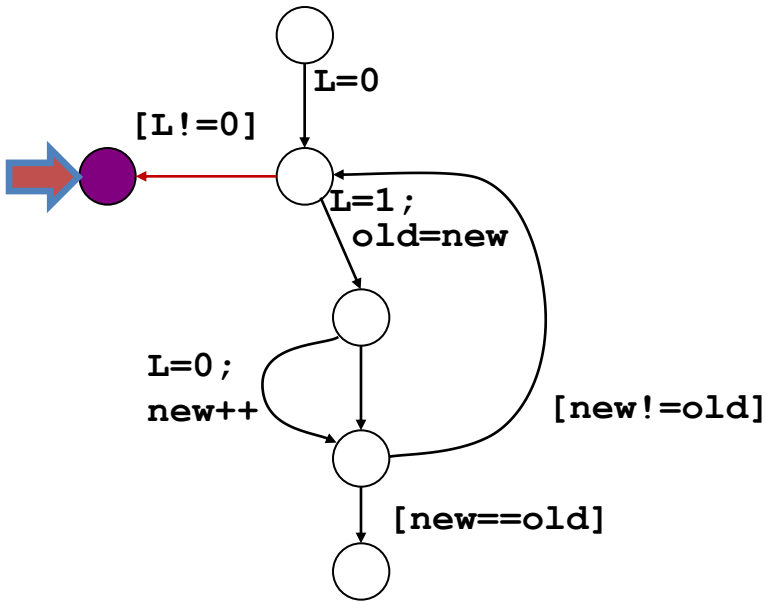
# Unwinding the CFG



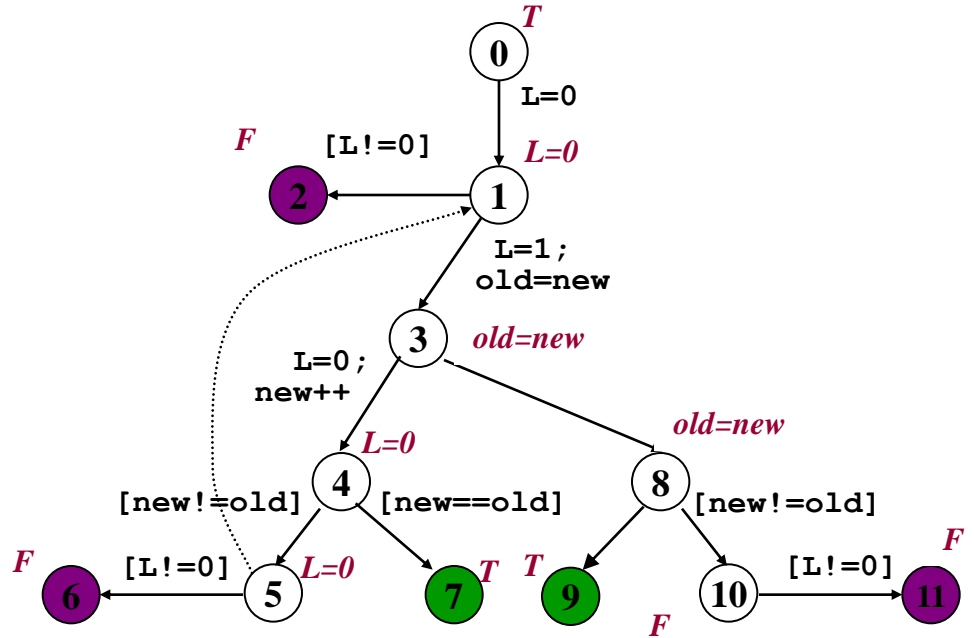
control-flow graph



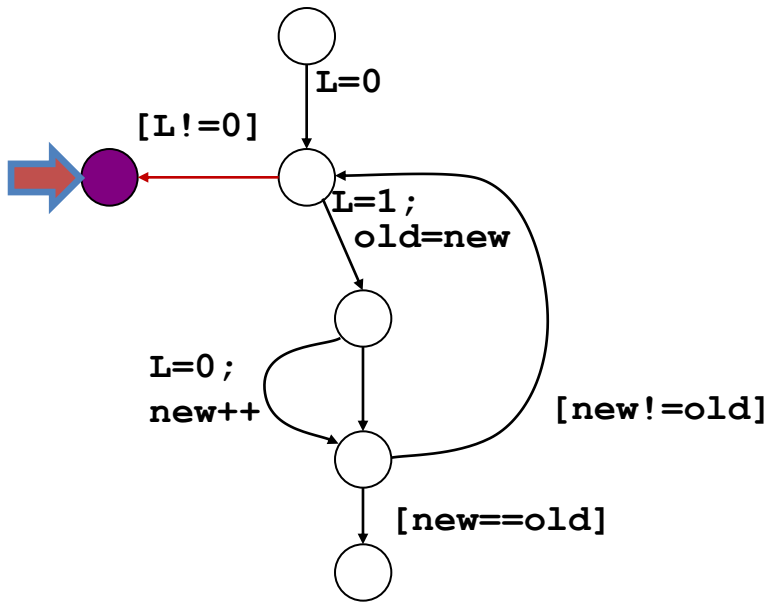
# Unwinding the CFG



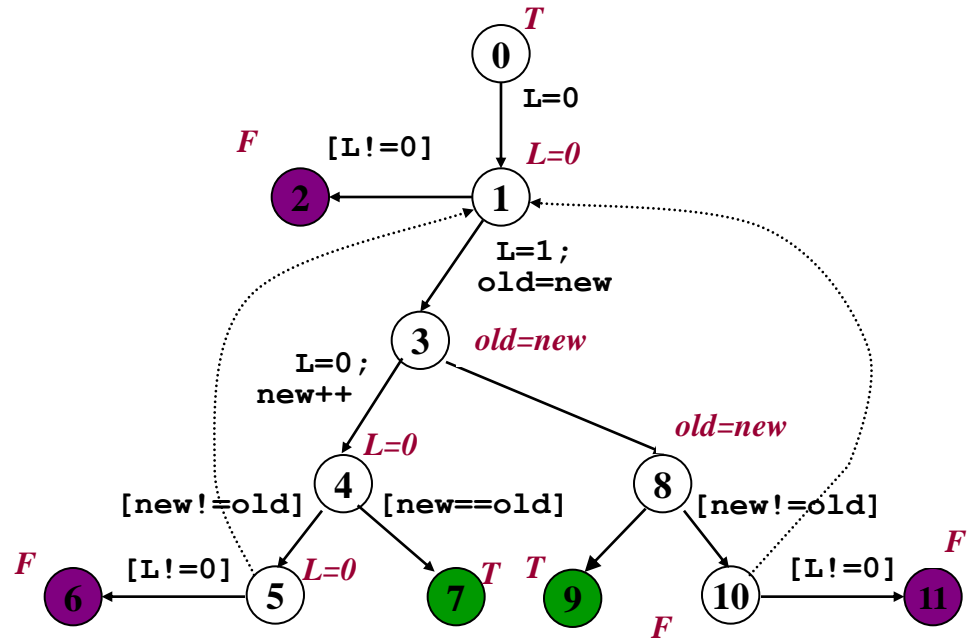
## control-flow graph



# Unwinding the CFG



control-flow graph



Another cover. Unwinding is now complete.

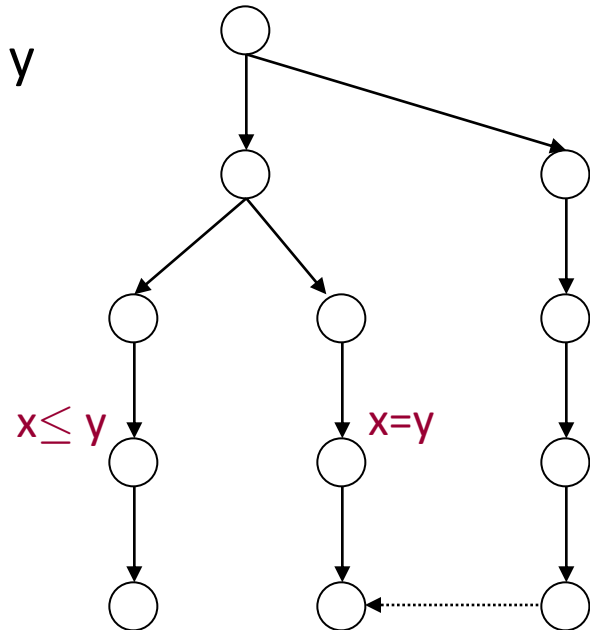
# Covering step

- If  $\psi(x) \Rightarrow \psi(y)$ ...
  - add covering arc  $x \triangleright y$
  - remove all  $z \triangleright w$  for  $w$  descendant of  $y$



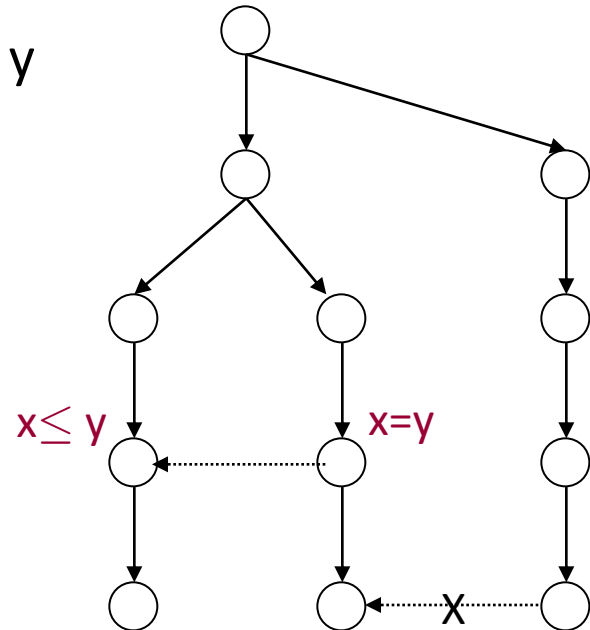
# Covering step

- If  $\psi(x) \Rightarrow \psi(y) \dots$ 
  - add covering arc  $x \triangleright y$
  - remove all  $z \triangleright w$  for  $w$  descendant of  $y$



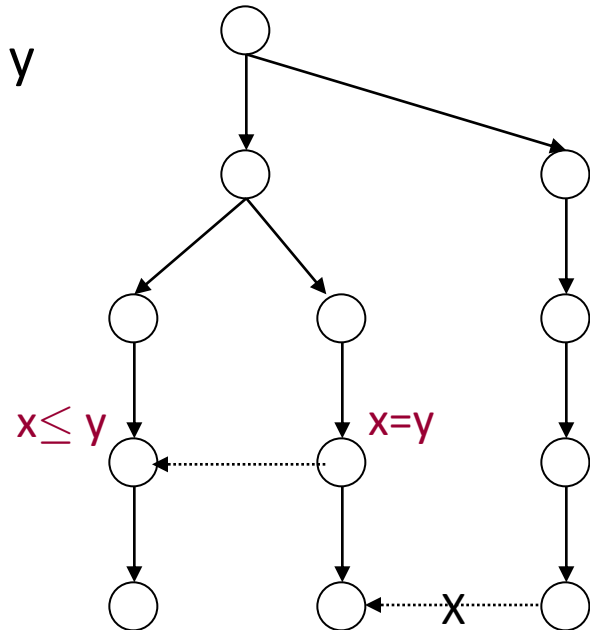
# Covering step

- If  $\psi(y) \Rightarrow \psi(x) \dots$ 
  - add covering arc  $x \triangleright y$
  - remove all  $w \triangleright z$  for  $w$  descendant of  $y$



# Covering step

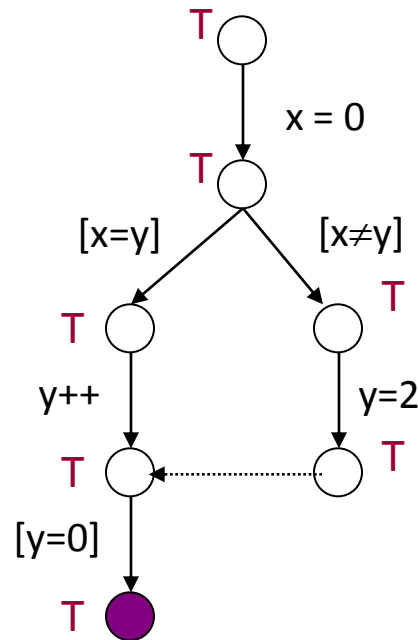
- If  $\psi(y) \Rightarrow \psi(x) \dots$ 
  - add covering arc  $x \triangleright y$
  - remove all  $w \triangleright z$  for  $w$  descendant of  $y$



We restrict covers to be descending in a suitable total order on vertices.  
This prevents covering from diverging.

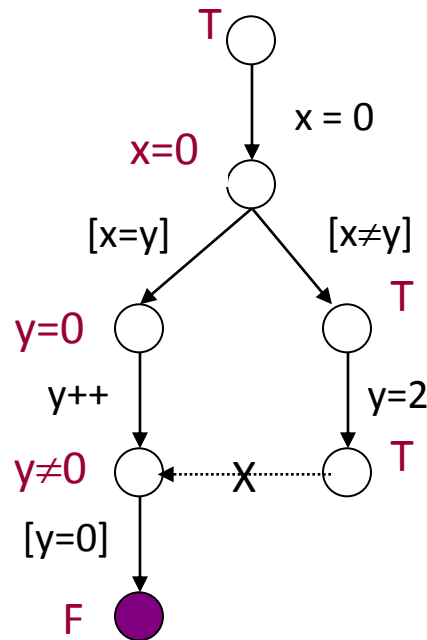
# Refinement step

- Label an error vertex False by refining the path to that vertex with an interpolant for that path.
- By refining with interpolants, we avoid predicate image computation.



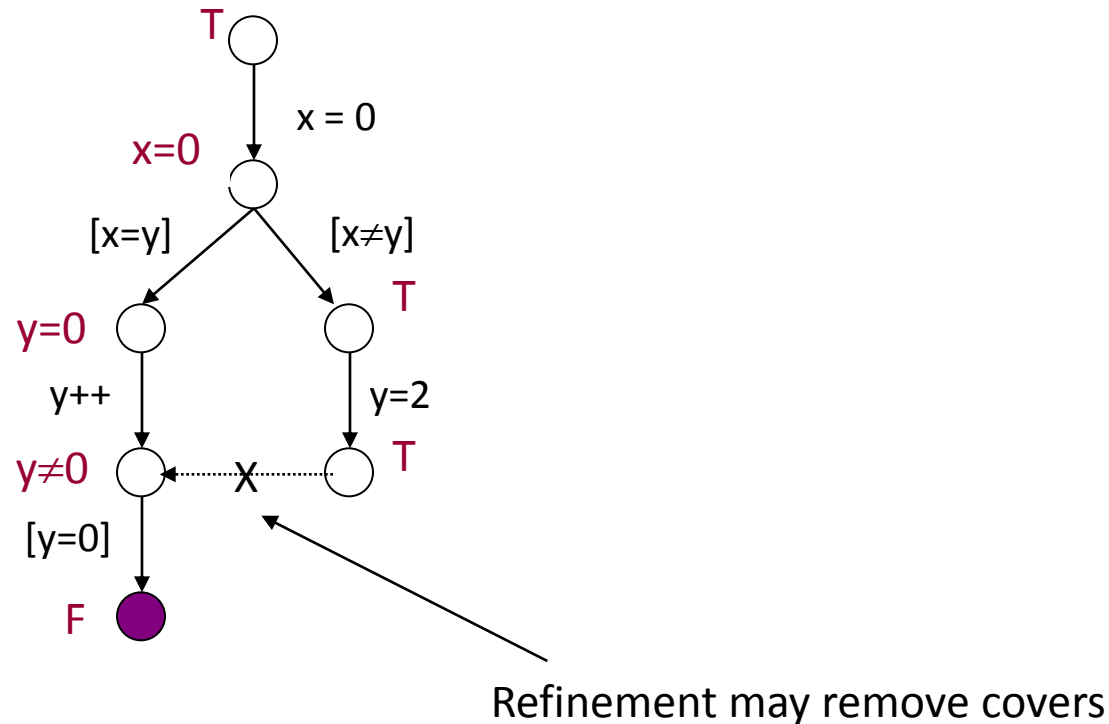
# Refinement step

- Label an error vertex False by refining the path to that vertex with an interpolant for that path.
- By refining with interpolants, we avoid predicate image computation.



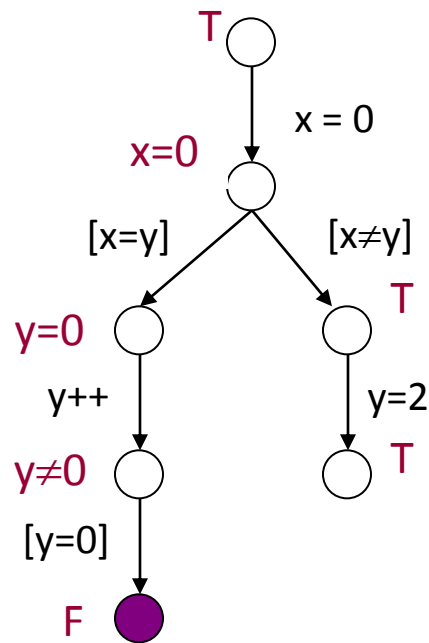
# Refinement step

- Label an error vertex False by refining the path to that vertex with an interpolant for that path.
- By refining with interpolants, we avoid predicate image computation.



# Forced cover

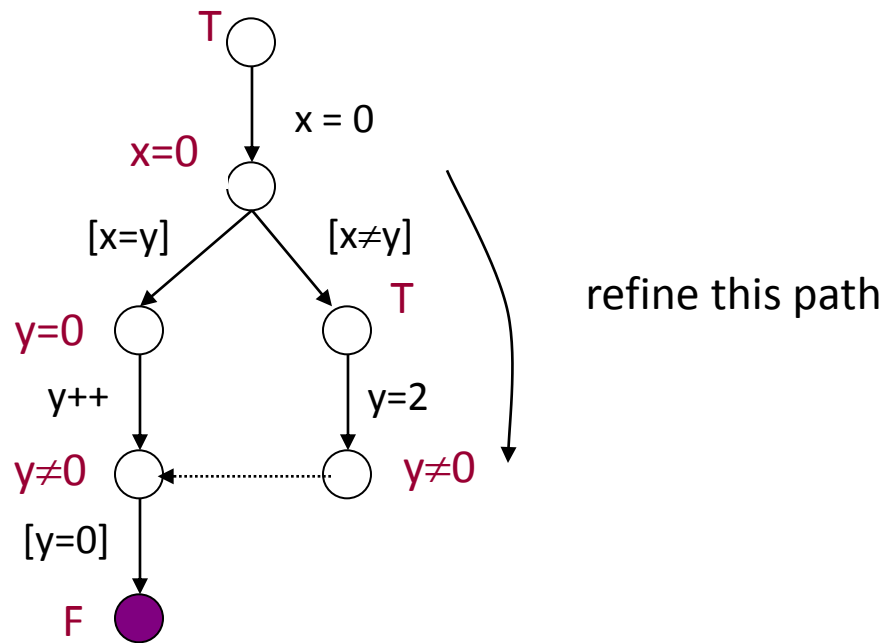
- Try to refine a sub-path to force a cover
  - show that path from nearest common ancestor of  $x, y$  proves  $\psi(x)$  at  $y$



Forced cover allow us to efficiently handle nested control structure

# Forced cover

- Try to refine a sub-path to force a cover
  - show that path from nearest common ancestor of  $x, y$  proves  $\psi(x)$  at  $y$

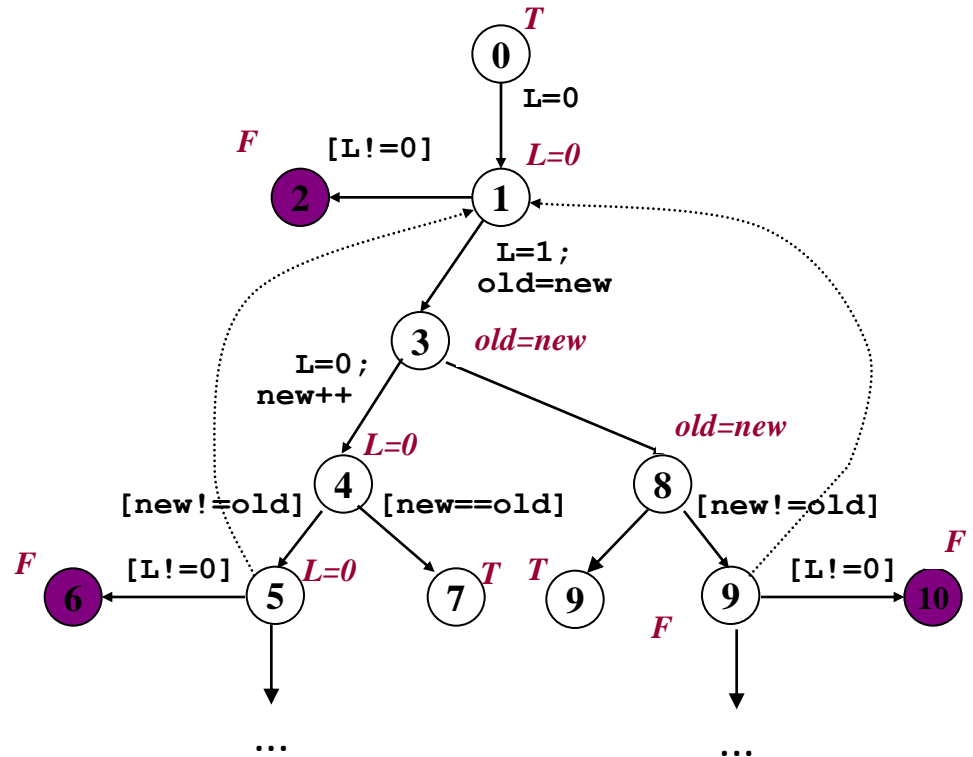


Forced cover allow us to efficiently handle nested control structure



# Safe and complete

- An unwinding is
  - *safe* if every error vertex is labeled False
  - *complete* if every nonterminal leaf is covered



Theorem: A CFG with a safe complete unwinding is safe.

# Unwinding steps

- Three basic operations:
  - Expand a nonterminal leaf
  - Cover: add a covering arc
  - Refine: strengthen labels along a path so error vertex labeled False

# Overall algorithm

1. Do as much covering as possible
2. If a leaf can't be covered, try forced covering
3. If the leaf still can't be covered, expand it
4. Label all error states False by refining with an interpolant
5. Continue until unwinding is safe and complete

# Interpolant Sequence in Princess

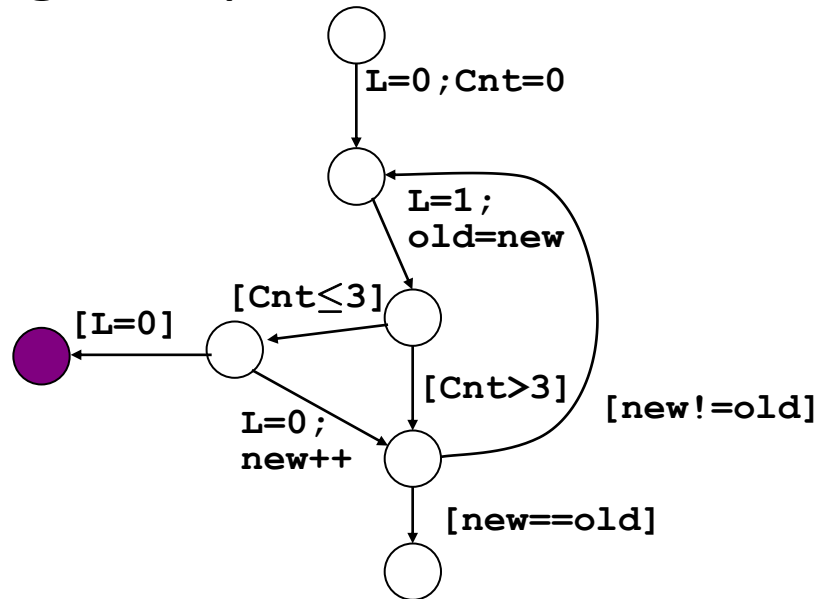
```
\functions {  
  int L0, L1, old0, new0, L2, new1;  
}  
\problem {  
  \part[p1]      (L0 =0) &  
  \part[p2]      (L1 =1 & old0=new0) &  
  \part[p3]      (L2=0 & new1 =new0+1) &  
  \part[p4]      (new1 != old0) &  
  \part[p5]      (L2!=0)  
  ->  
  false  
}
```

```
\interpolant {p1; p2, p3, p4, p5}  
\interpolant {p1, p2; p3, p4, p5}  
\interpolant {p1, p2, p3; p4, p5}  
\interpolant {p1, p2, p3, p4; p5}
```

Interpolant for  
 $(L_0 = 0) \wedge (L_1 = 1 \wedge \text{old}_0 = \text{new}_0)$   
 $\wedge (L_2 = 0 \wedge \text{new}_1 = \text{new}_0 + 1) \wedge (\text{new}_1 \neq \text{old}_0)$   
 $\wedge (L_2 \neq 0)$

# Homework

- Run the two versions of verification algorithms on the following control flow graph, using Princess for computing interpolants



control-flow graph