Having a BLAST with SLAM

Meeting 4, CSCI 5535, Fall 2013

IF ANYONE HITS ME WITH A SNOWBALL, I'LL HIT *HIM* WITH 250 SNOWBALLS!
WHAT IF SOMEBODY HITS *YOU* WITH 250 SNOWBALLS?
..SIGHHH..

Announcements

- Homework 0 due Sat
  - Questions?
- Move Tue office hours to 4-5pm

Software Model Checking via Counterexample Guided Abstraction Refinement

There are easily dozens of papers.

We will skim.

SLAM Overview/Review

- **Input**:

  Specification – property to check : "no deadlocks"
  "program uses lock correctly"
  not "I am webserver"

  program / (programs , device drivers)

- **Output**:

  ✓ verified = program has no deadlocks

  × counterexample = path that may result in error
  that violates the spec

SLAM Overview

- **Input**: Program **and** Specification
  - Standard C Program (pointers, procedures)
  - Specification = Partial Correctness
    - Given as a finite state machine (typestate)
    - "I use locks correctly", not "I am a webserver"
- **Output**: Verified **or** Counterexample
  - Verified = program does not violate spec
    - Can come with proof!
  - Counterexample = concrete bug instance
    - A path through the program that violates the spec

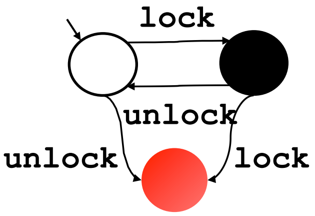
Take-Home Message

- **SLAM** is a **software model checker**. It **abstracts** C programs to **boolean programs** and model-checks the boolean programs.
- No errors in the boolean program implies no errors in the original.

true bug false alarm

- An error in the boolean program **may** be a real bug. Or SLAM may **refine** the abstraction and start again.

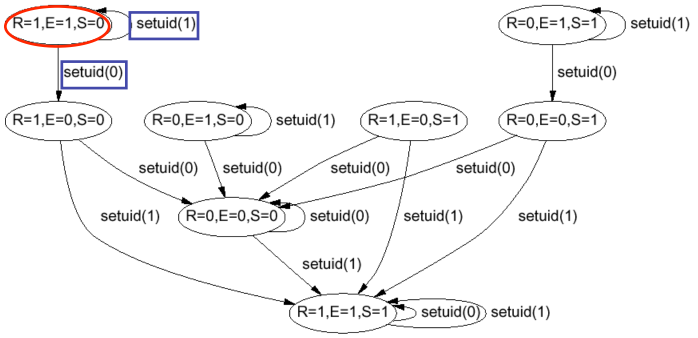
Property 1: Double Locking



"An attempt to re-acquire an acquired lock or release a released lock will cause a **deadlock**."

Calls to **lock** and **unlock** must **alternate**.

Property 2: Drop Root Privilege

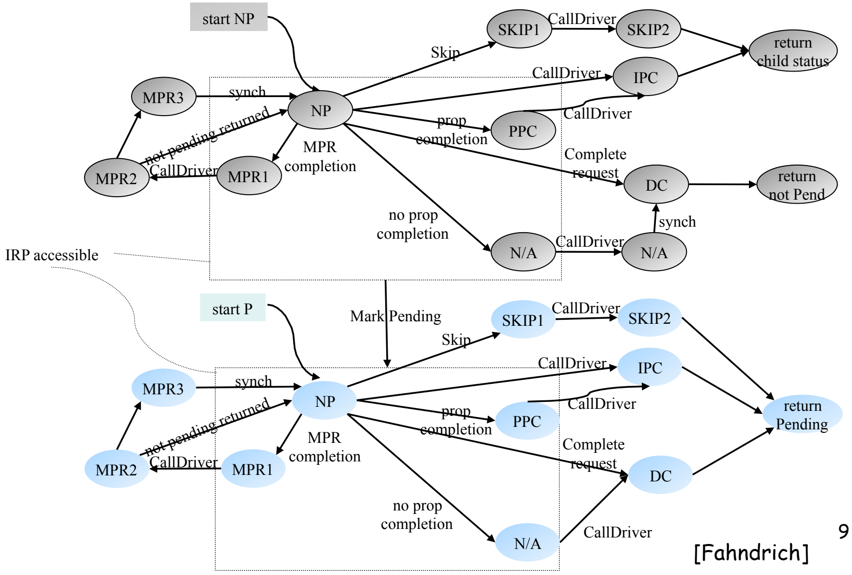


[Chen-Wagner-Dean '02]

"User applications must not run with root privilege"

When **execv** is called, must have **suid ≠ 0**

Property 3 : IRP Handler



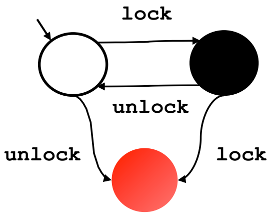
9

[Fahndrich]

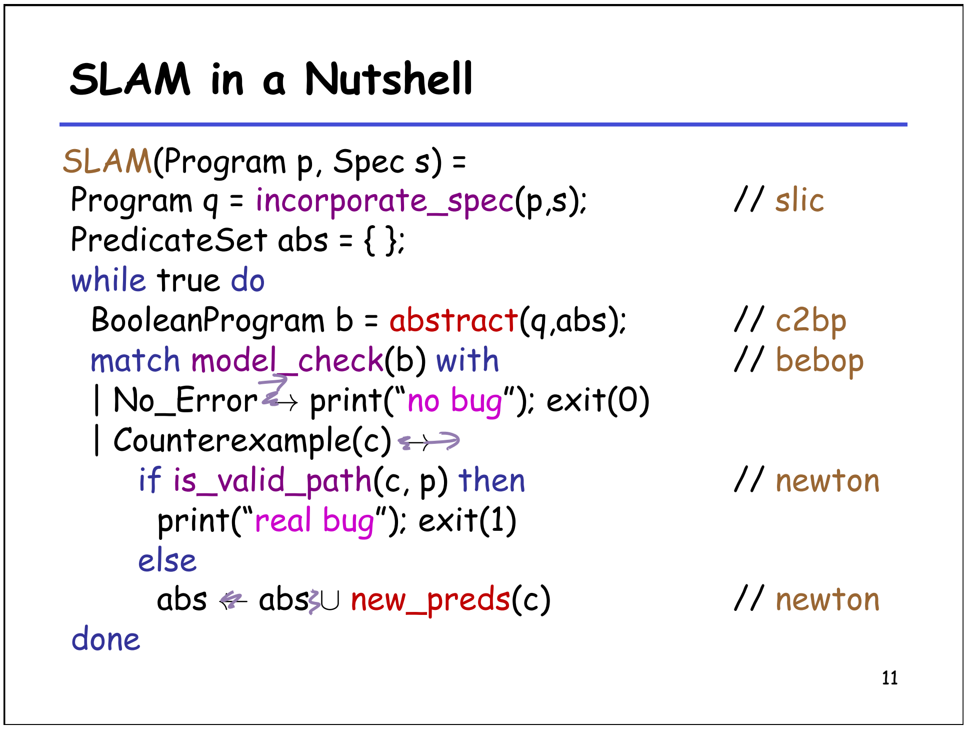
Example SLAM Input

```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:         q->data = new;

       unlock();
           new ++;
       }
4: } while(new != old);
5: unlock();
   return;
}
```



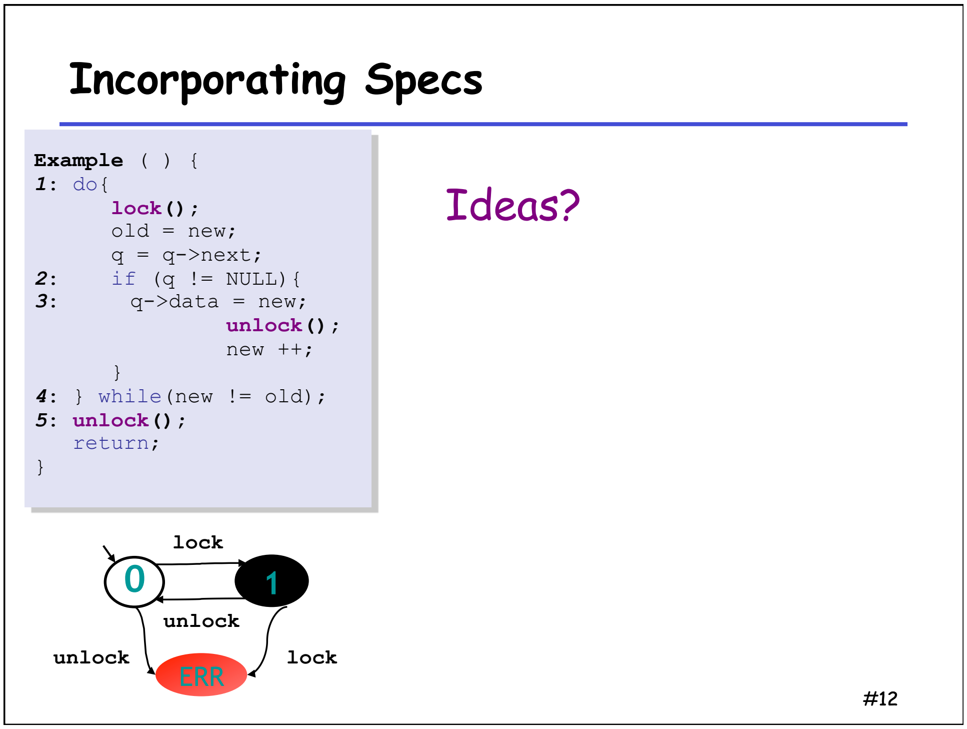
10



SLAM in a Nutshell

```
SLAM(Program p, Spec s) =
 Program q = incorporate_spec(p,s);              // slic
 PredicateSet abs = { };
 while true do
   BooleanProgram b = abstract(q,abs);           // c2bp
   match model_check(b) with                     // bebop
   | No_Error → print("no bug"); exit(0)
   | Counterexample(c) →
       if is_valid_path(c, p) then               // newton
         print("real bug"); exit(1)
       else
         abs ← abs ∪ new_preds(c)                // newton
 done
```

11



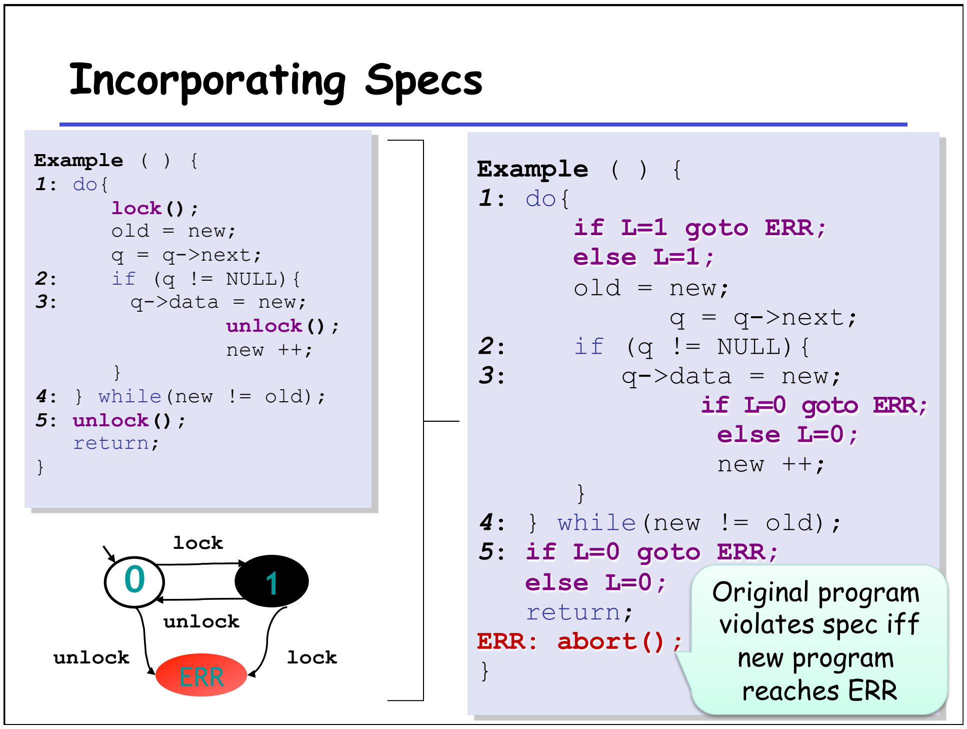
Incorporating Specs

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:     q->data = new;
            unlock();
            new ++;
     }
4: } while(new != old);
5: unlock();
   return;
}
```

Ideas?

lock
0
1
unlock
unlock
lock
ERR

#12

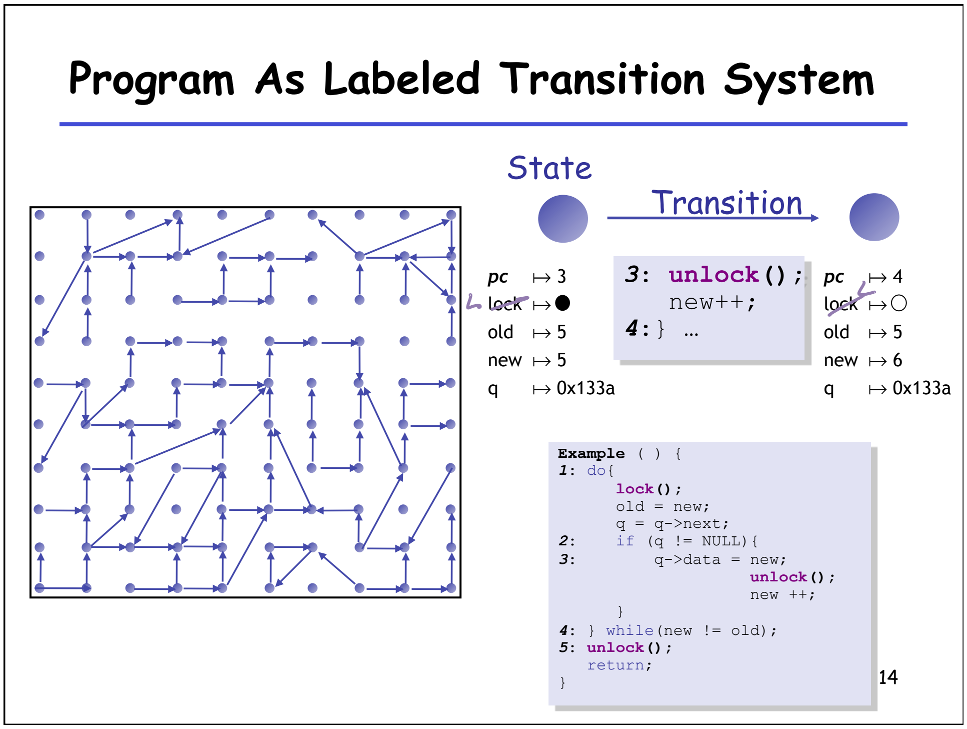


Incorporating Specs

```
Example ( ) {
1: do{
        lock();
        old = new;
        q = q->next;
2:      if (q != NULL){
3:        q->data = new;
                unlock();
                new ++;
        }
4: } while(new != old);
5: unlock();
   return;
}
```

```
Example ( ) {
1: do{
        if L=1 goto ERR;
        else L=1;
        old = new;
              q = q->next;
2:      if (q != NULL){
3:           q->data = new;
                  if L=0 goto ERR;
                   else L=0;
                   new ++;
        }
4: } while(new != old);
5: if L=0 goto ERR;
   else L=0;
   return;
ERR: abort();
}
```

Original program violates spec iff new program reaches ERR



Program As Labeled Transition System

State

Transition

pc ↦ 3
lock ↦ ●
old ↦ 5
new ↦ 5
q ↦ 0x133a

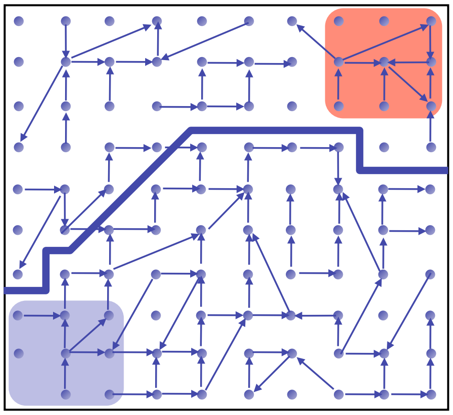
```
3: unlock();
   new++;
4:} …
```

pc ↦ 4
lock ↦ ○
old ↦ 5
new ↦ 6
q ↦ 0x133a

```
Example ( ) {
1: do{
        lock();
        old = new;
        q = q->next;
2:      if (q != NULL){
3:        q->data = new;
                unlock();
                new ++;
        }
4: } while(new != old);
5: unlock();
   return;
}
```

14

The Safety Verification Problem



Error

(e.g., states with PC = Err)

Safe States

(never reach Error)

Initial

Is there a path from an initial to an error state ?

Problem? Infinite state graph (old=1, old=2, old=…)

Solution? Set of states ≃ logical formula

15

Representing [Sets of States] as Formulas

$[x<5] = \{x \mapsto 0, x \mapsto 1, x \mapsto 2, x \mapsto -1, x \mapsto 3, x \mapsto 4, \ldots\}$

$\boxed{x<5}$

$[F]$ states satisfying F $\{\underline{s} \mid s \models F\}$	F FO formula over program vars
$[F_1] \cap [F_2]$	$F_1 \wedge F_2$
$[F_1] \cup [F_2]$	$F_1 \vee F_2$
$\overline{[F]}$	$\neg F$
$[F_1] \subseteq [F_2]$	

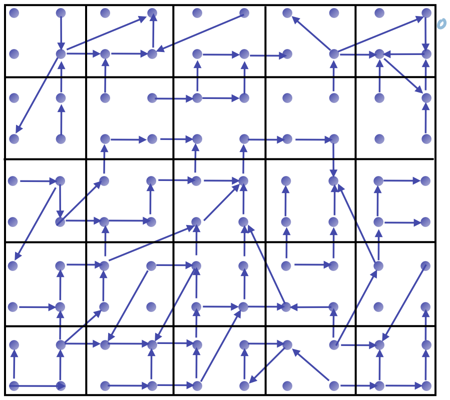
i.e. $F_1 \wedge \neg F_2$ unsatisfiable

Representing [Sets of States] as Formulas

$[F]$ states satisfying F $\{s \mid s \models F\}$	F FO formula over program vars
$[F_1] \cap [F_2]$	$F_1 \wedge F_2$
$[F_1] \cup [F_2]$	$F_1 \vee F_2$
$\overline{[F]}$	$\neg F$
$[F_1] \subseteq [F_2]$	$F_1 \Rightarrow F_2$

i.e. $F_1 \wedge \neg F_2$ unsatisfiable

Idea 1: Predicate Abstraction



- **Predicates** on program state:

  0 0 1 1 *lock* *(i.e., lock=true)*

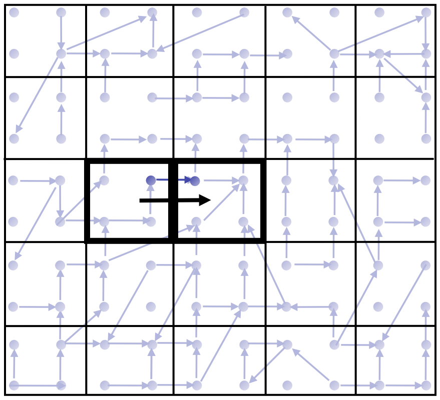
  0 1 0 1 *old = new*

- States satisfying **same** predicates are **equivalent**
  - **Merged** into one **abstract state**

Why?

- Num of abstract states is **finite**
  - **Thus model-checking the abstraction will be feasible!**

Abstract States and Transitions



State

Transition

pc	↦ 3	
lock	↦ ●	
old	↦ 5	
new	↦ 5	
q	↦ 0x133a	

```
3: unlock();
   new++;
4:} ...
```

pc	↦ 4
lock	↦ ○
old	↦ 5
new	↦ 6
q	↦ 0x133a

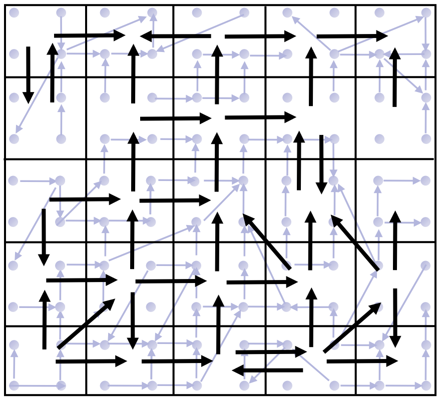
Theorem Prover

lock
old=new

¬ ¬ *lock*
¬ ¬ *old=new*

19

Abstraction



State

Transition

c_1 c_2

pc	↦ 3
lock	↦ ●
old	↦ 5
new	↦ 5
q	↦ 0x133a

```
3: unlock();
   new++;
4:} ...
```

pc	↦ 4
lock	↦ ○
old	↦ 5
new	↦ 6
q	↦ 0x133a

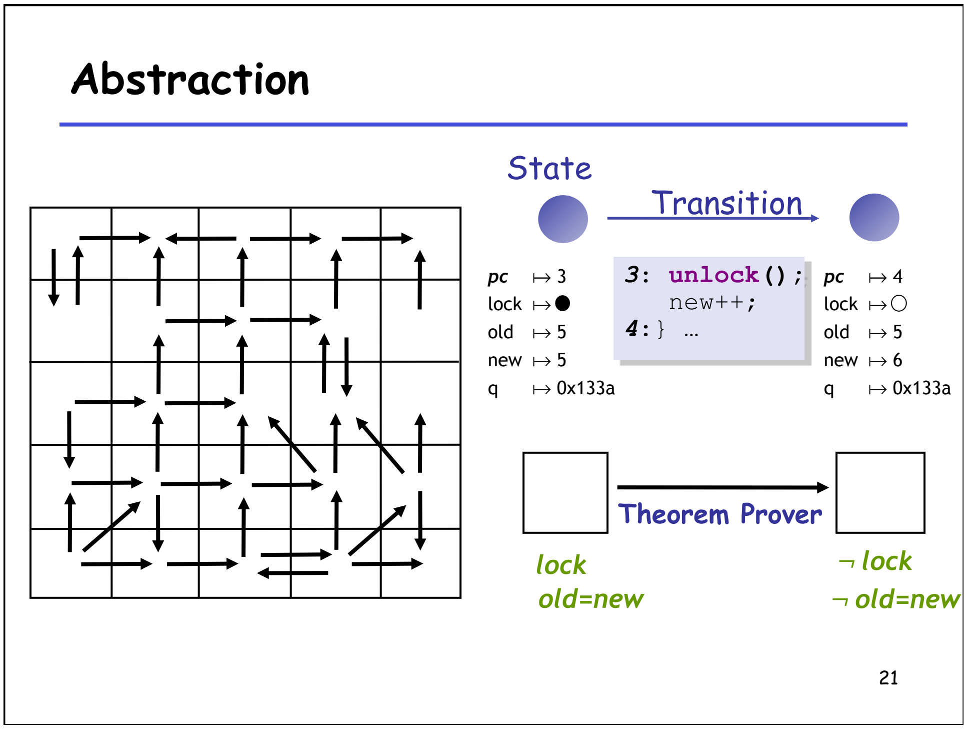
A_1 Theorem Prover A_2

lock
old=new

¬ *lock*
¬ *old=new*

Existential Lifting
(i.e., $A_1 \overset{\to}{\mapsto} A_2$ iff $\overset{\exists}{}\,\exists \overset{G}{c_1} \in A_1\overset{\exists}{.}\ \exists \overset{G}{c_2} \in A_2.\ \overset{\to}{c_1} \to c_2$)

20



Abstraction

State

Transition

pc $\mapsto$ 3
lock $\mapsto$ ●
old $\mapsto$ 5
new $\mapsto$ 5
q $\mapsto$ 0x133a

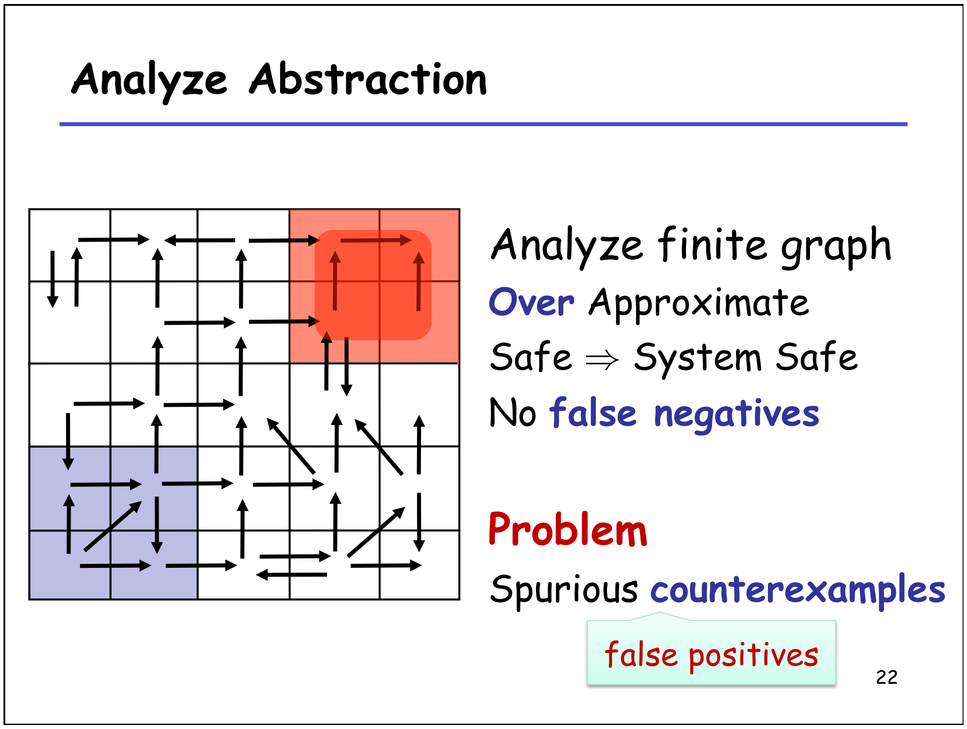
```
3: unlock();
   new++;
4: } ...
```

pc $\mapsto$ 4
lock $\mapsto$ ○
old $\mapsto$ 5
new $\mapsto$ 6
q $\mapsto$ 0x133a

Theorem Prover

lock
old=new

¬ lock
¬ old=new



Analyze Abstraction

Analyze finite graph

Over Approximate

Safe ⇒ System Safe

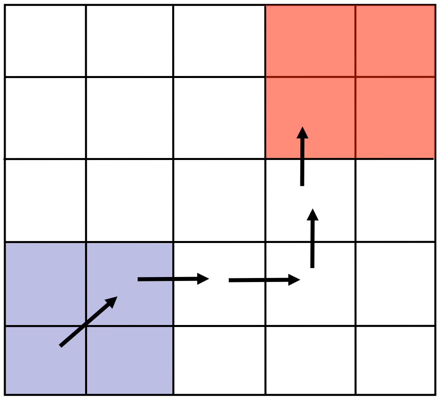
No **false negatives**

Problem

Spurious **counterexamples**

false positives

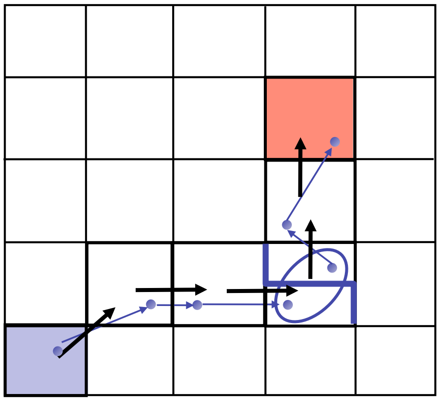
Idea 2: Counterexample-Guided Refinement



Solution

Use spurious **counterexamples** to **refine** abstraction!

Idea 2: Counterexample-Guided Refinement



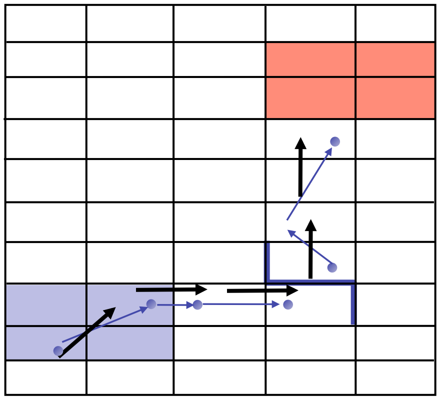
Solution

Use spurious **counterexamples** to **refine** abstraction!

1. **Add predicates** to distinguish states across **cut**
2. Build **refined** abstraction

Imprecision due to **merge**

Iterative Abstraction-Refinement



[Kurshan et al 93] [Clarke et al 00]
[Ball-Rajamani 01]

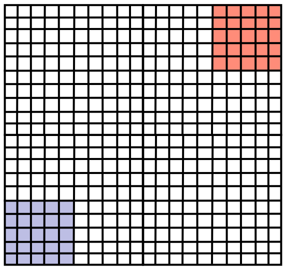
Solution

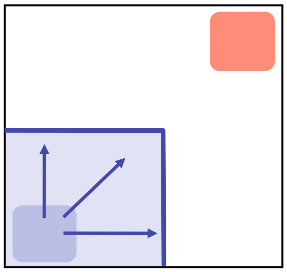
Use spurious **counterexamples** to **refine** abstraction!

1. **Add predicates** to distinguish states across **cut**
2. Build **refined** abstraction
   - eliminates counterexample
3. **Repeat** search until real counterexample or system proved safe

Problem: Abstraction is Expensive

Why?





Reachable

Problem

#abstract states = $2^{\#predicates}$

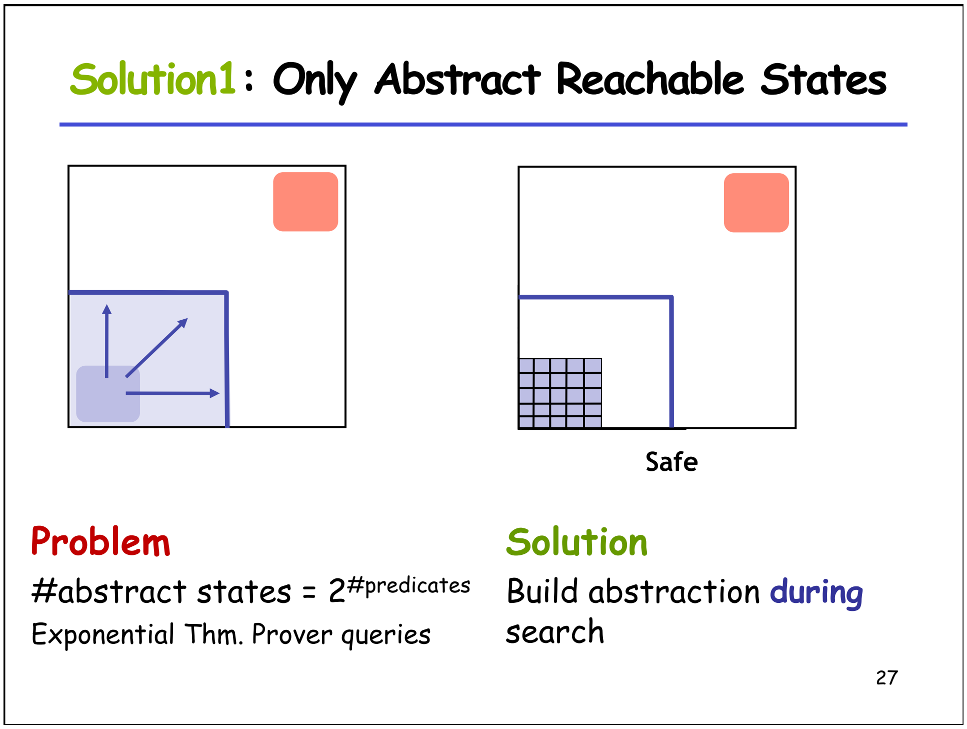
Exponential Thm. Prover queries

Observe

Fraction of state space reachable

#Preds ~ 100's, #States ~ 2^{100} ,

#Reach ~ 1000's



Solution1: Only Abstract Reachable States

Problem

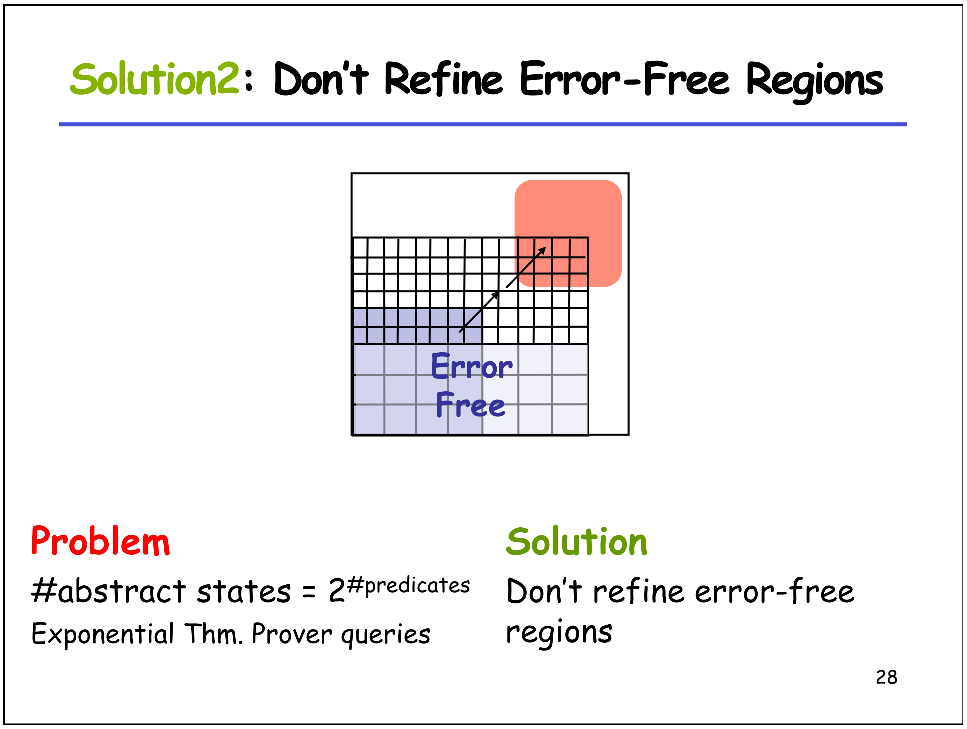
#abstract states = $2^{\#predicates}$

Exponential Thm. Prover queries

Solution

Build abstraction **during** search

27



Solution2: Don't Refine Error-Free Regions

Problem

#abstract states = $2^{\#predicates}$

Exponential Thm. Prover queries

Solution

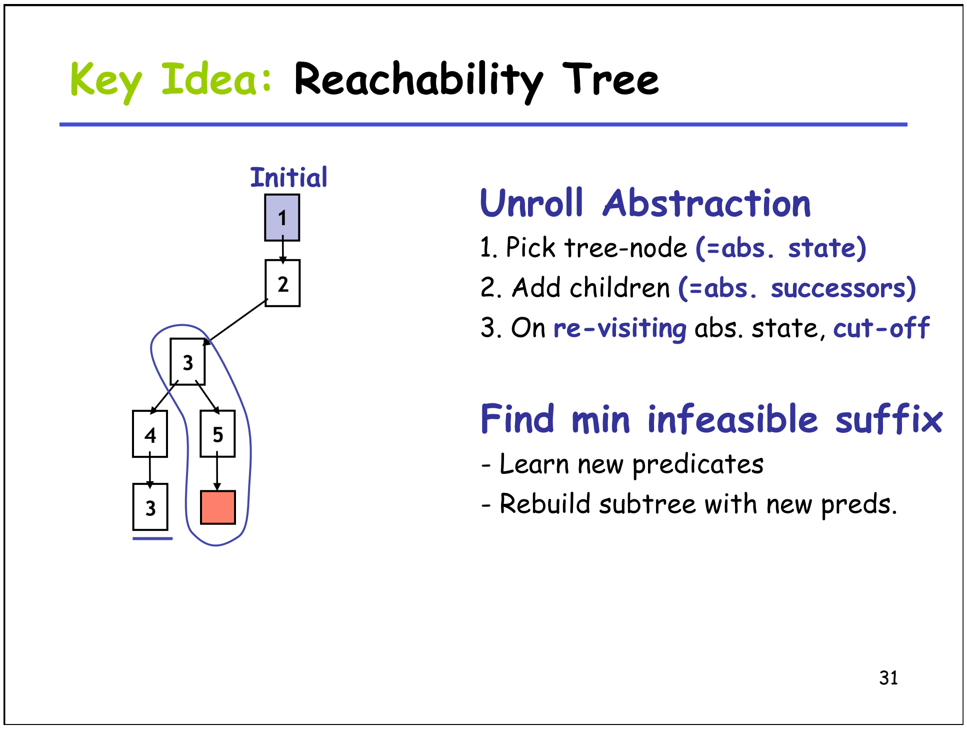
Don't refine error-free regions

28

29

Key Idea for Solutions?

30



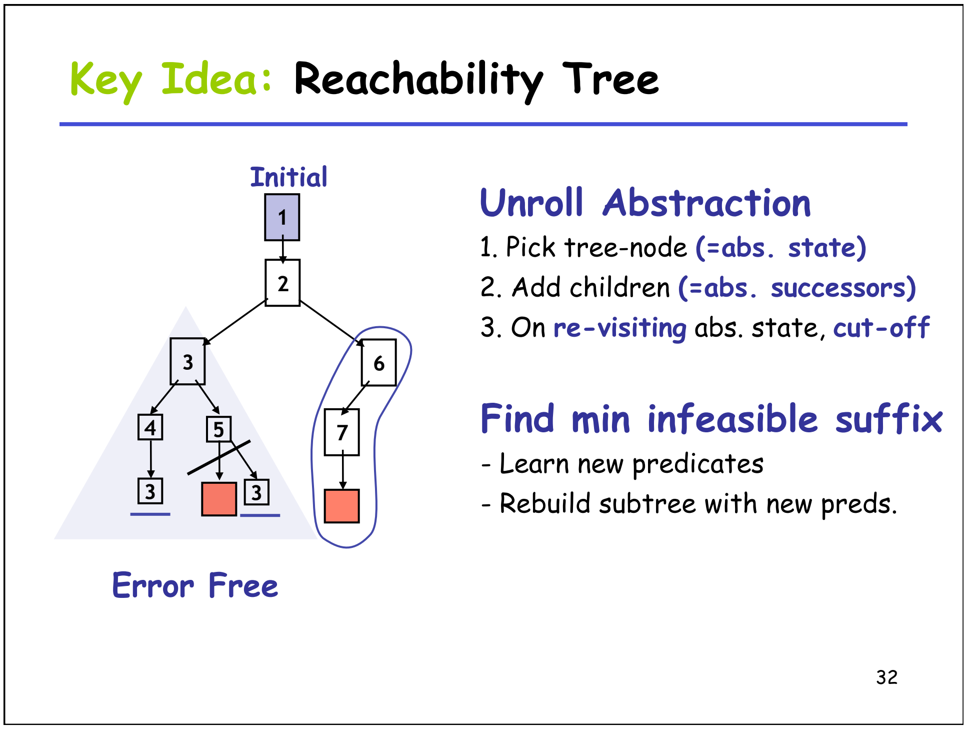
Key Idea: Reachability Tree

Unroll Abstraction

1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

- Learn new predicates
- Rebuild subtree with new preds.



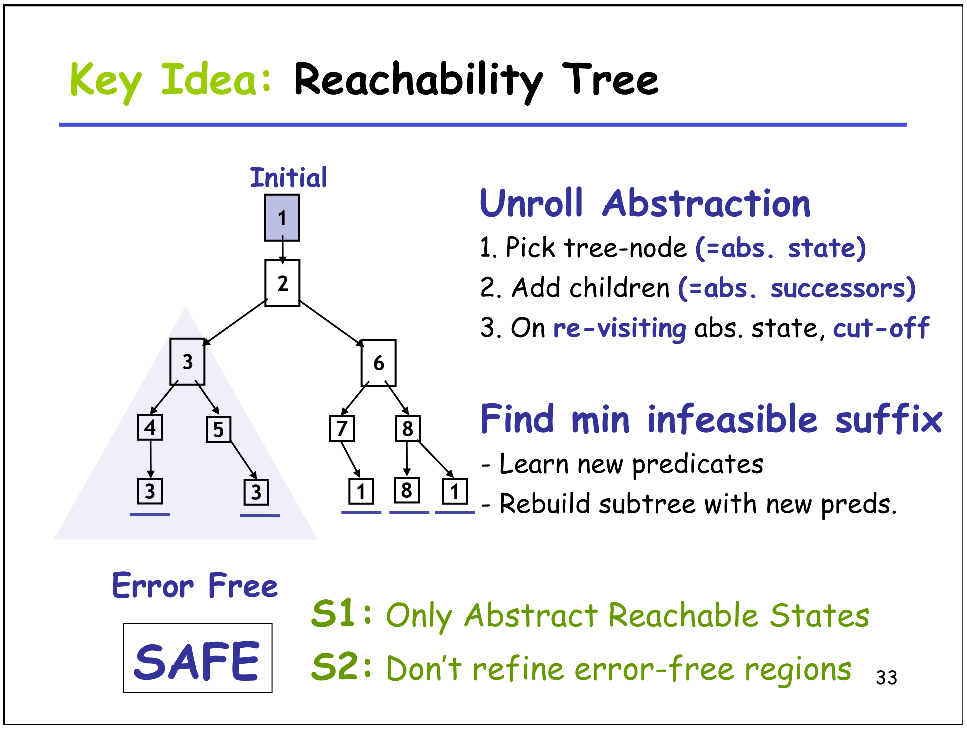
Key Idea: Reachability Tree

Unroll Abstraction

1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

- Learn new predicates
- Rebuild subtree with new preds.



Key Idea: Reachability Tree

Unroll Abstraction

1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

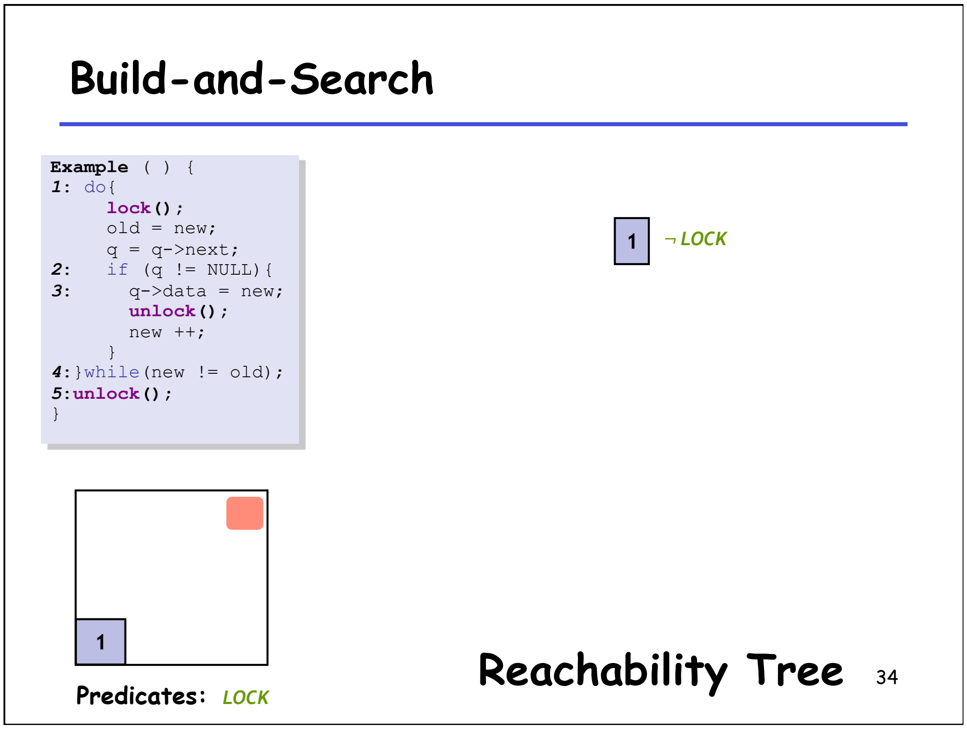
- Learn new predicates
- Rebuild subtree with new preds.

Error Free

SAFE

S1: Only Abstract Reachable States

S2: Don't refine error-free regions

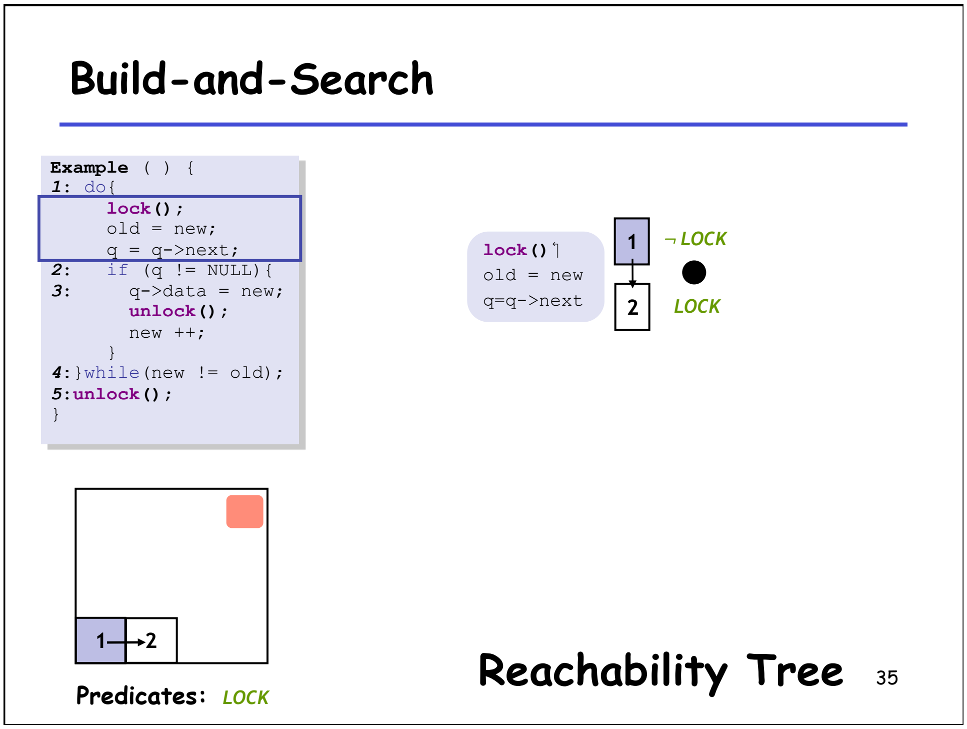


Build-and-Search

```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock();
}
```

Predicates: LOCK

Reachability Tree

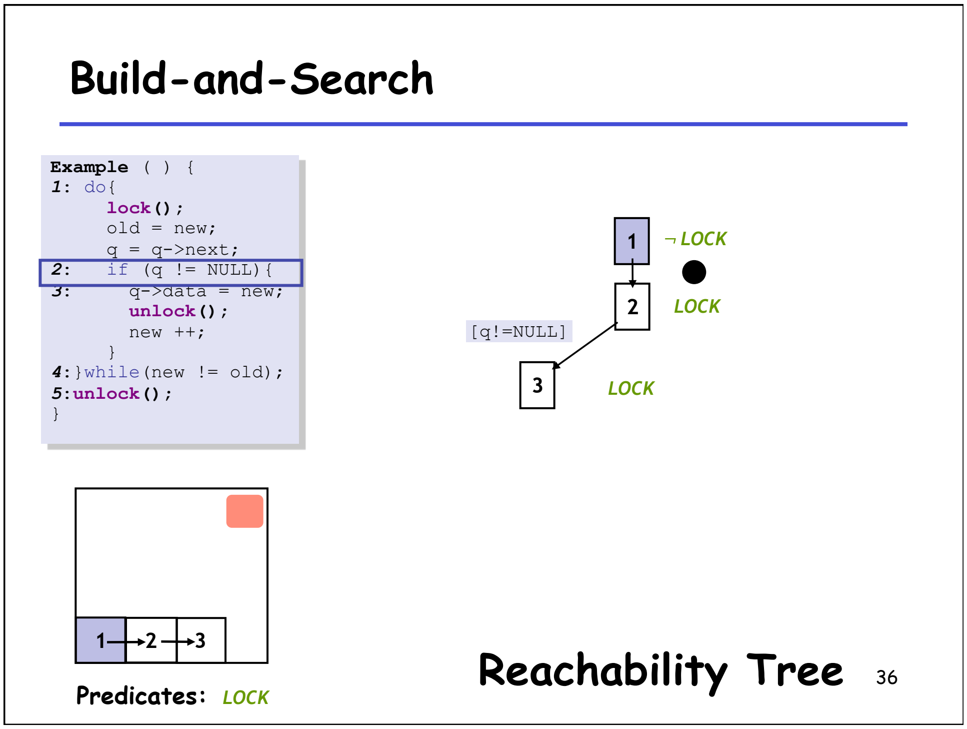


Build-and-Search

```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock();
}
```

Reachability Tree 35

Predicates: LOCK



Build-and-Search

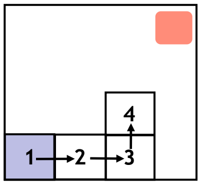
```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock();
}
```

Reachability Tree 36

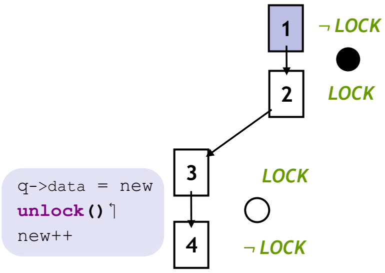
Predicates: LOCK

Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```



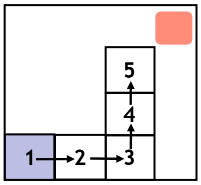
Predicates: LOCK



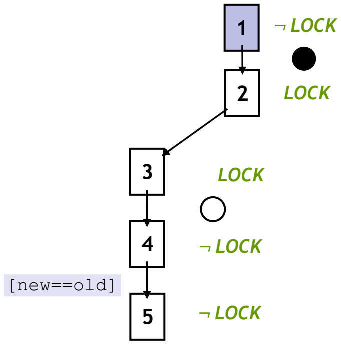
Reachability Tree

Build-and-Search

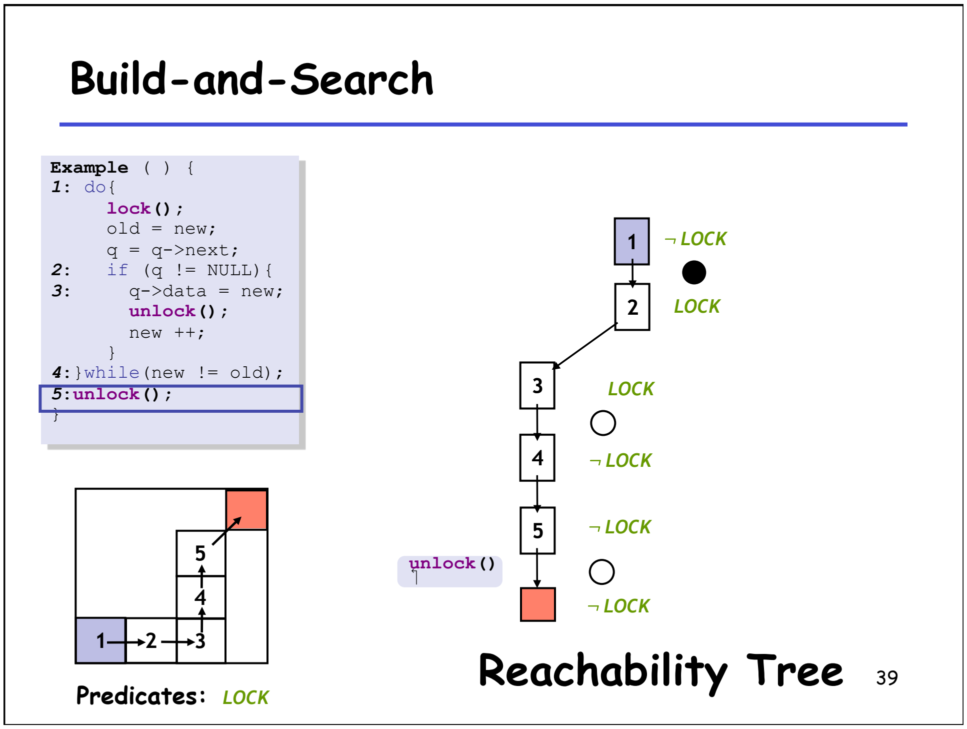
```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```



Predicates: LOCK



Reachability Tree

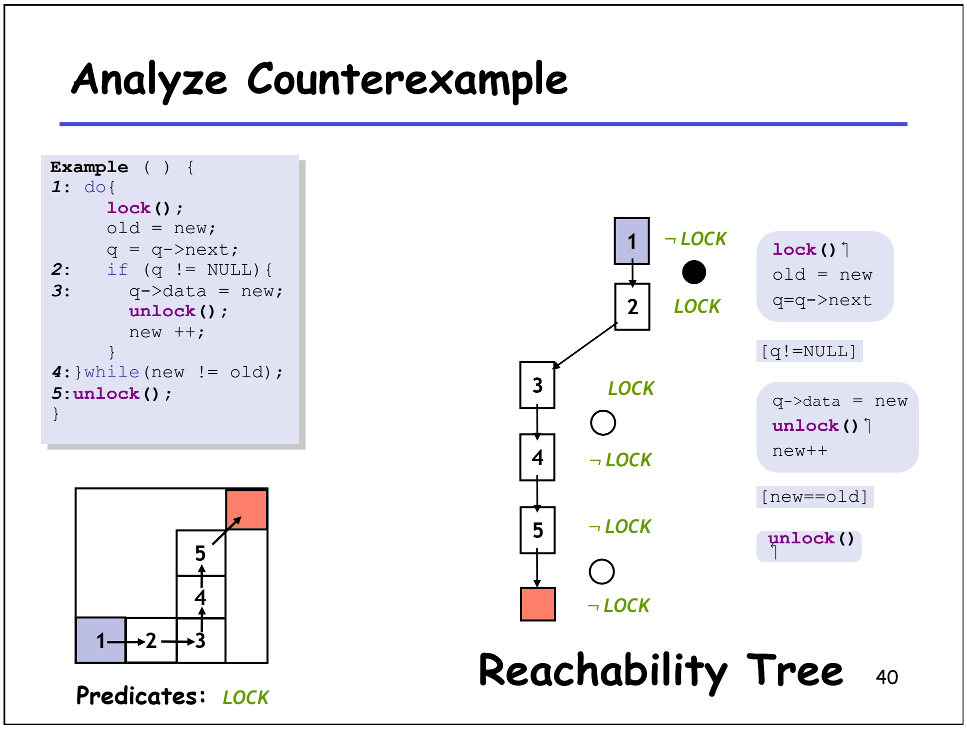


Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```

Predicates: *LOCK*

Reachability Tree



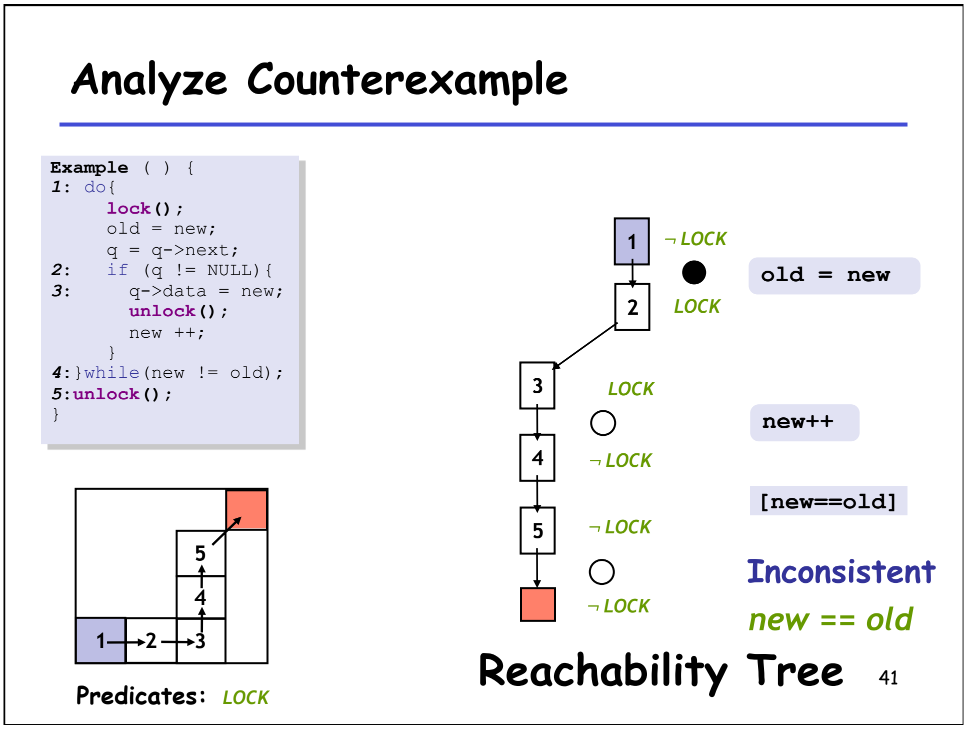
Analyze Counterexample

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```

```
lock() ⌉
old = new
q=q->next
[q!=NULL]
q->data = new
unlock() ⌉
new++
[new==old]
unlock() ⌉
```

Predicates: *LOCK*

Reachability Tree



Analyze Counterexample

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```

Predicates: *LOCK*

old = new

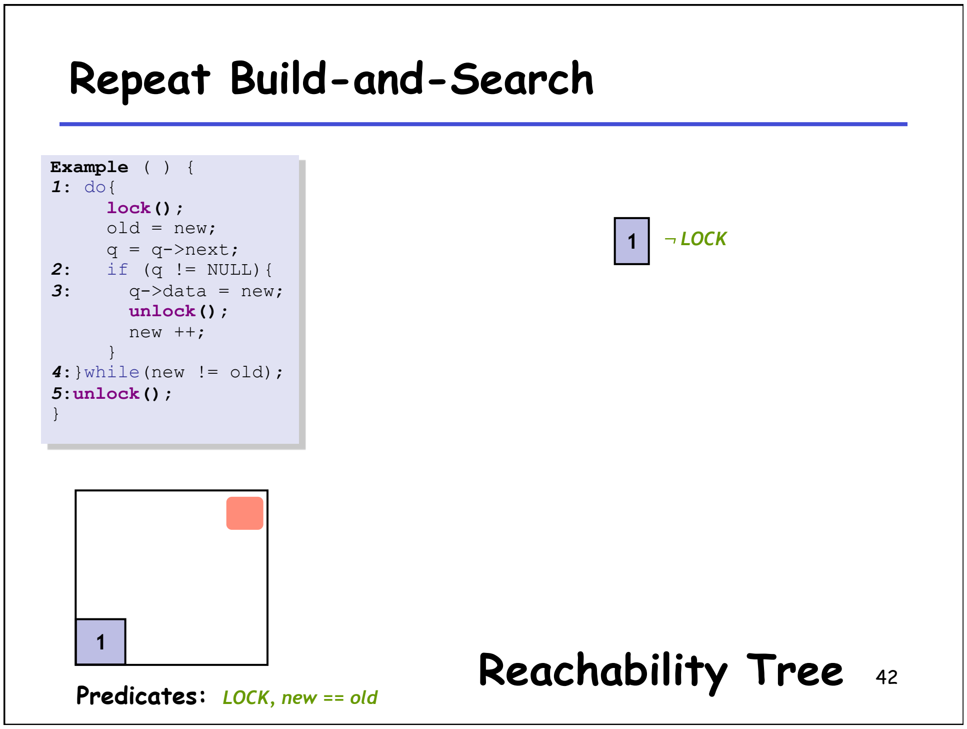
new++

[new==old]

Inconsistent

new == old

Reachability Tree 41

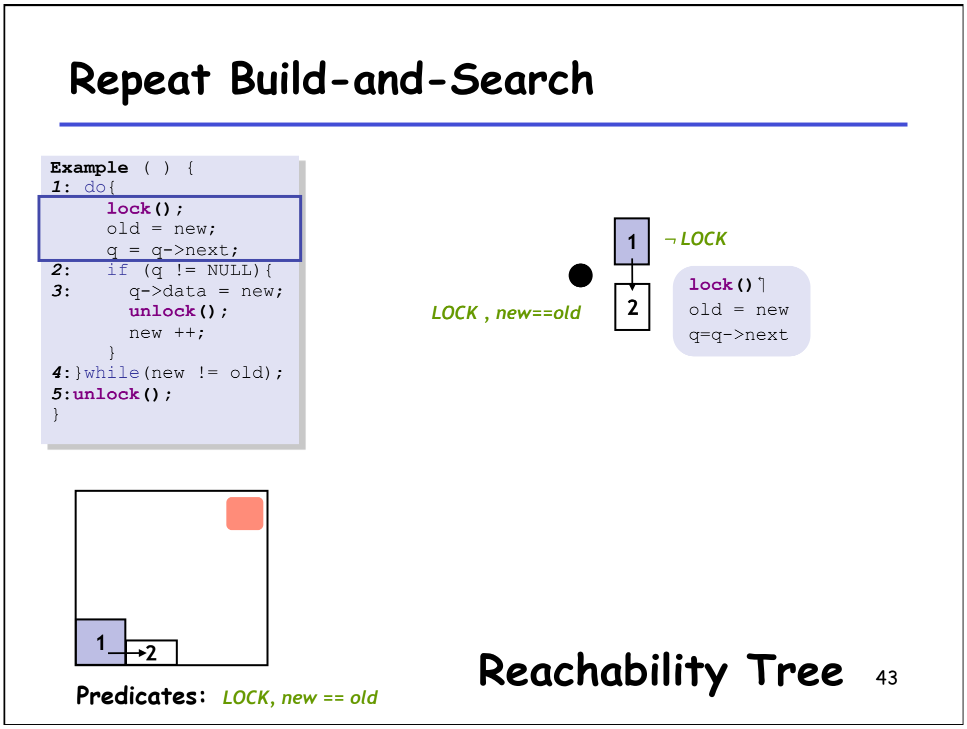


Repeat Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```

Predicates: *LOCK, new == old*

Reachability Tree 42



Repeat Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:     q->data = new;
       unlock();
       new ++;
     }
4:}while(new != old);
5:unlock();
}
```

¬ LOCK

LOCK , new==old

1

2

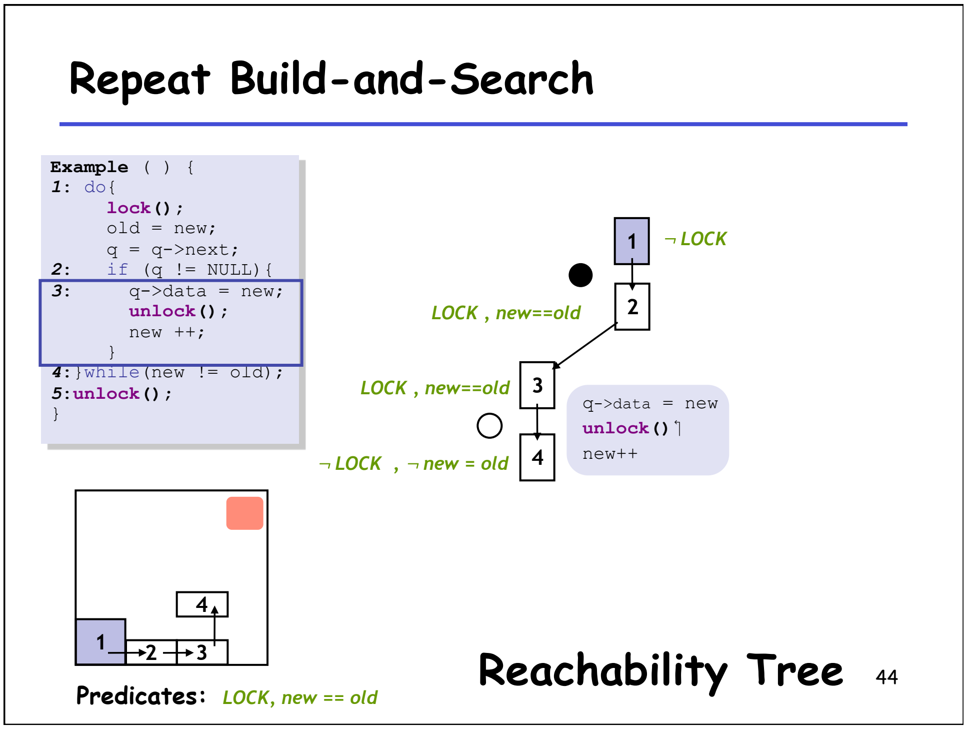
lock() ⌉
old = new
q=q->next

1

2

Predicates: *LOCK, new == old*

Reachability Tree 43



Repeat Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:     q->data = new;
       unlock();
       new ++;
     }
4:}while(new != old);
5:unlock();
}
```

¬ LOCK

LOCK , new==old

LOCK , new==old

¬ LOCK , ¬ new = old

1

2

3

4

q->data = new
unlock() ⌉
new++

1

2

3

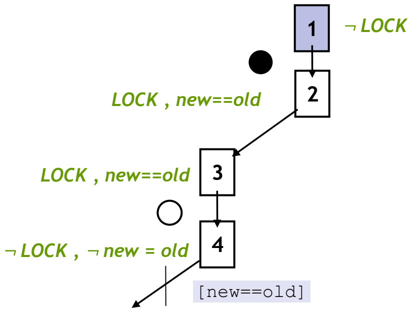
4

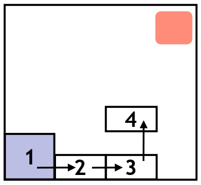
Predicates: *LOCK, new == old*

Reachability Tree 44

Repeat Build-and-Search

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```



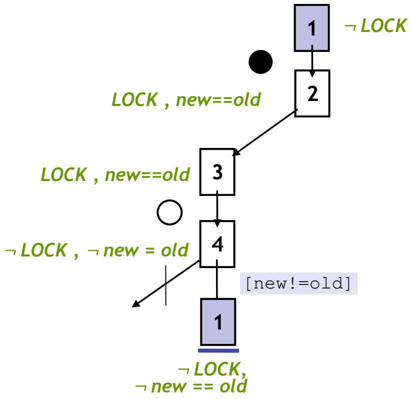


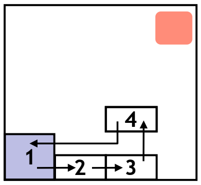
Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

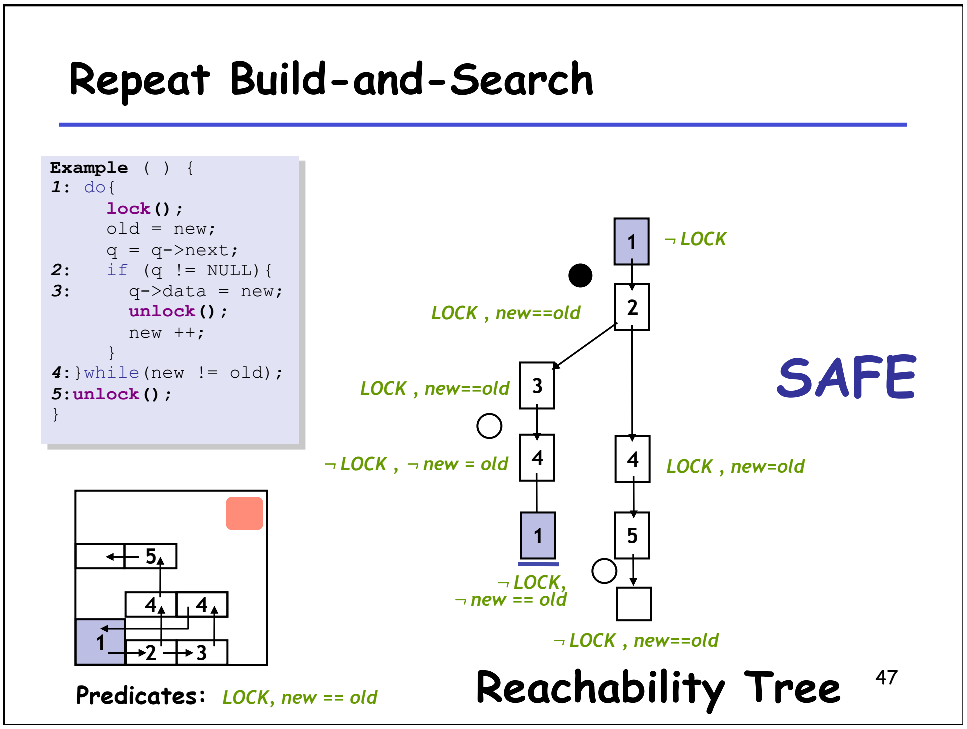
```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```





Predicates: LOCK, new == old

Reachability Tree



Repeat Build-and-Search

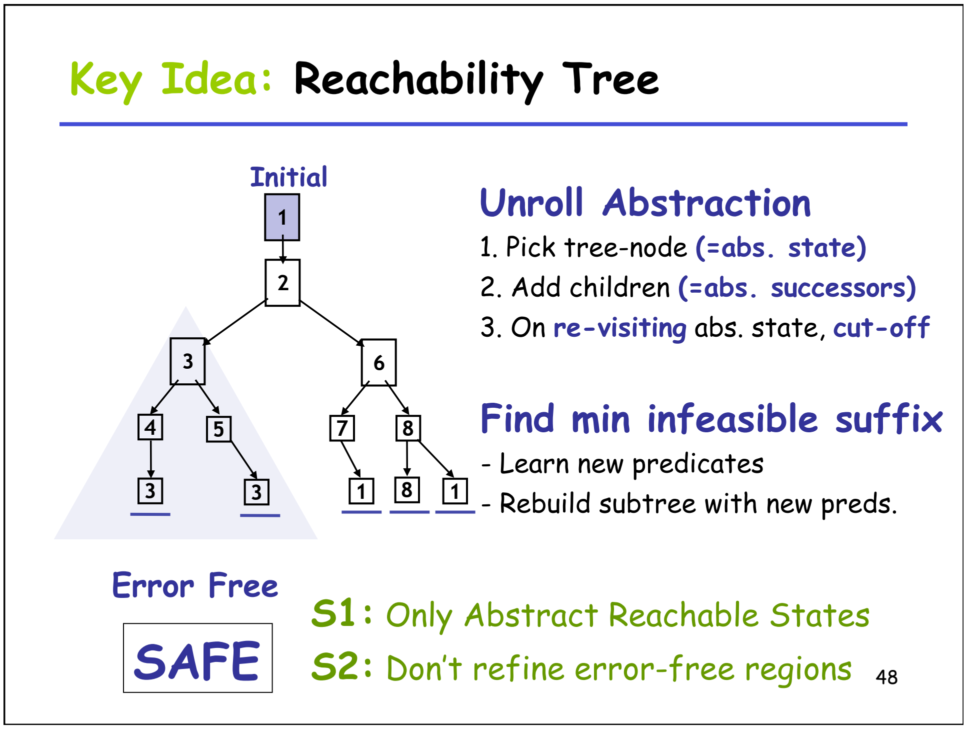
```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock();
}
```

SAFE

Predicates: LOCK, new == old

Reachability Tree

47



Key Idea: Reachability Tree

Unroll Abstraction

1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

- Learn new predicates
- Rebuild subtree with new preds.

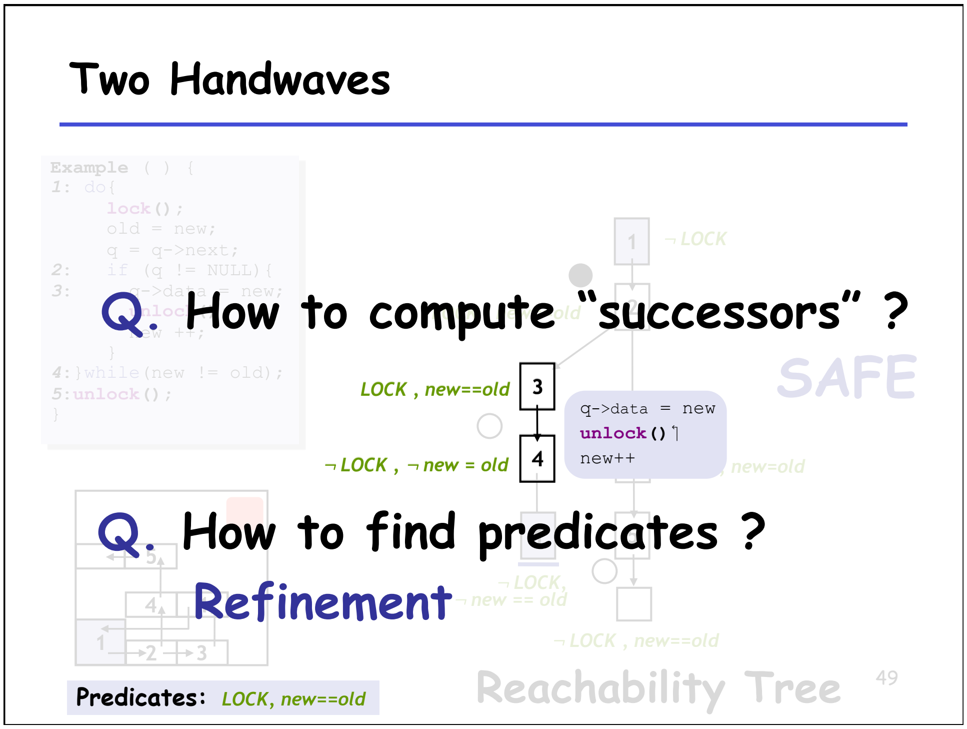
Error Free

SAFE

S1: Only Abstract Reachable States

S2: Don't refine error-free regions

48



Two Handwaves

```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock ();
}
```

Q. How to compute "successors" ?

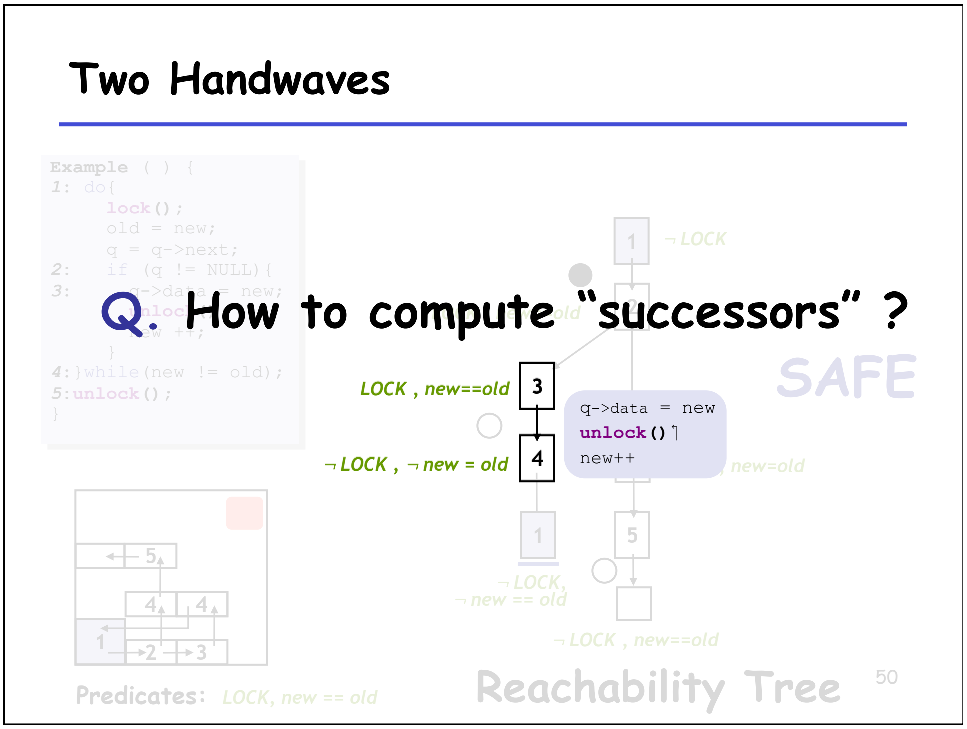
Q. How to find predicates ?

Refinement

Predicates: LOCK, new==old

Reachability Tree

49



Two Handwaves

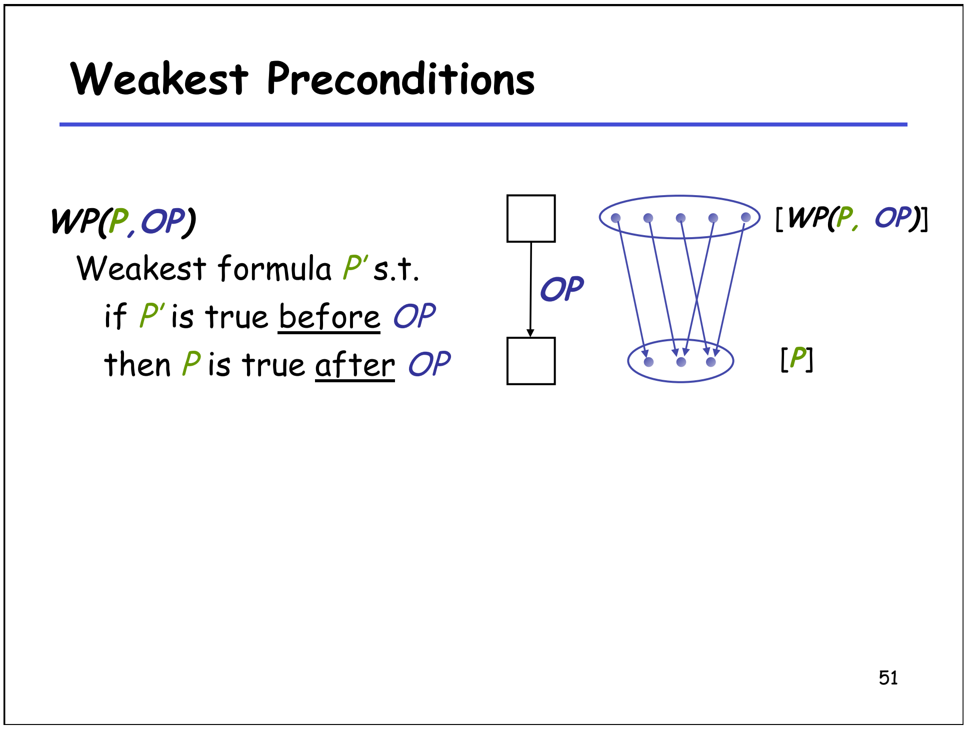
```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
      }
4:}while(new != old);
5:unlock ();
}
```

Q. How to compute "successors" ?

Predicates: LOCK, new == old

Reachability Tree

50



Weakest Preconditions

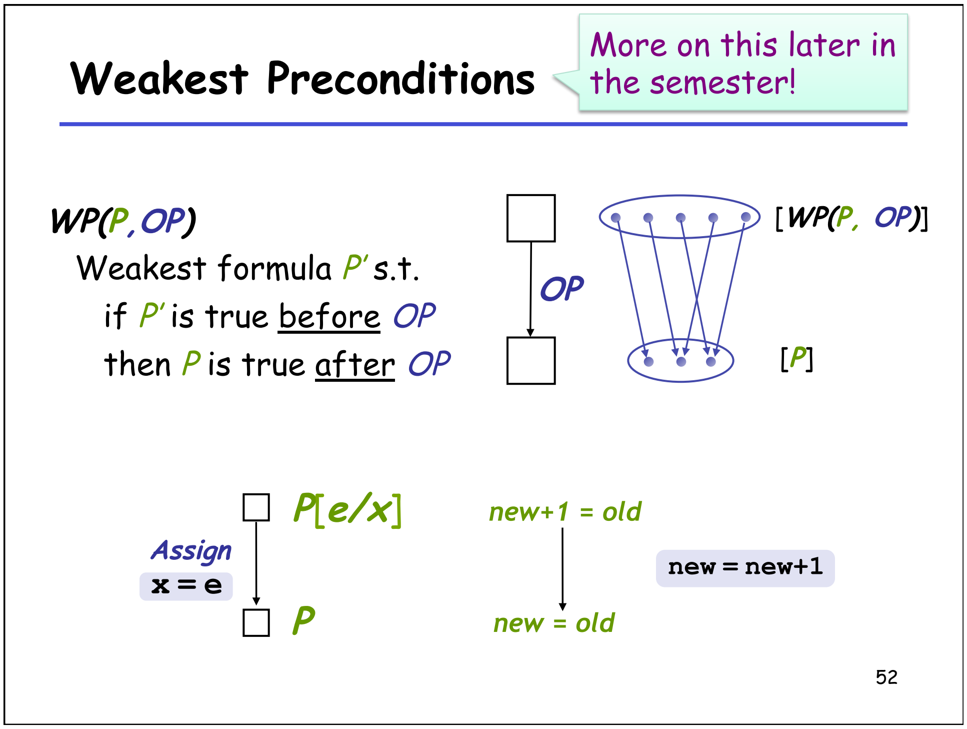
WP(P,OP)

Weakest formula *P'* s.t.

if *P'* is true <u>before</u> *OP*

then *P* is true <u>after</u> *OP*

51



Weakest Preconditions

More on this later in the semester!

WP(P,OP)

Weakest formula *P'* s.t.

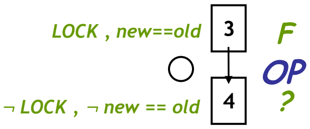
if *P'* is true <u>before</u> *OP*

then *P* is true <u>after</u> *OP*

52

How to compute successor?

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```



For each *p*

- Check if ***p*** is true (or false) after ***OP***

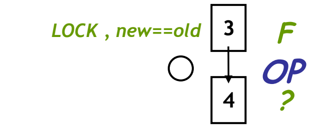
Q: When is ***p*** true <u>after</u> ***OP*** ?

- If ***WP(p, OP)*** is true <u>before</u> ***OP*** !
- We know ***F*** is true <u>before</u> ***OP***
- Thm. Pvr. Query: ***F*** ⇒ ***WP(p, OP)***

Predicates: *LOCK, new == old*

How to compute successor?

```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
     }
4:}while(new != old);
5:unlock();
}
```



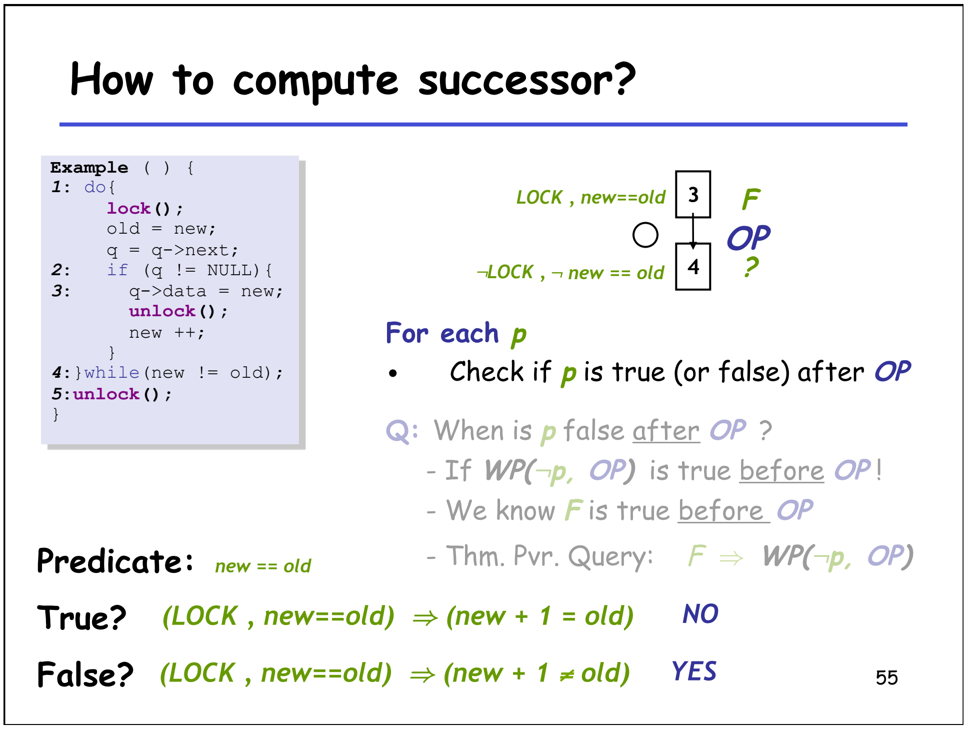
For each *p*

- Check if ***p*** is true (or false) after ***OP***

Q: When is ***p*** false <u>after</u> ***OP*** ?

- If ***WP(¬p, OP)*** is true <u>before</u> ***OP*** !
- We know ***F*** is true <u>before</u> ***OP***
- Thm. Pvr. Query: ***F*** ⇒ ***WP(¬p, OP)***

Predicates: *LOCK, new == old*



How to compute successor?

```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:        q->data = new;
          unlock();
          new ++;
       }
4:}while(new != old);
5:unlock();
}
```

LOCK , new==old
3
¬LOCK , ¬ new == old
4
F
OP
?

For each *p*

- Check if ***p*** is true (or false) after ***OP***

Q: When is *p* false after *OP* ?

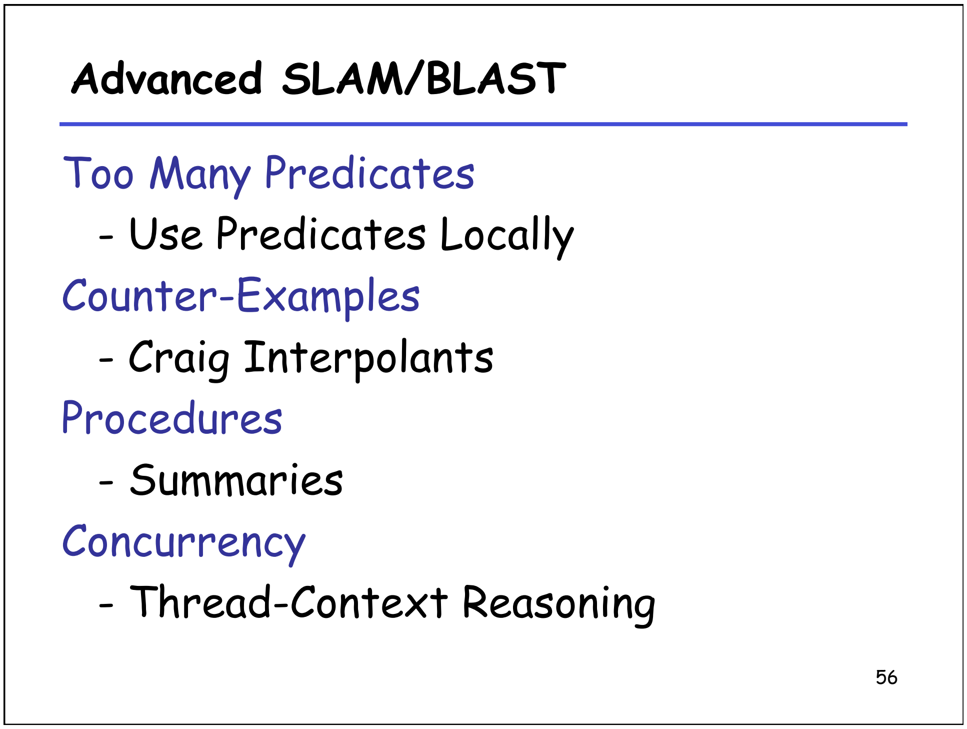
- If *WP(¬p, OP)* is true before *OP* !
- We know *F* is true before *OP*
- Thm. Pvr. Query: *F* ⇒ *WP(¬p, OP)*

Predicate: *new == old*

True? *(LOCK , new==old)* ⇒ *(new + 1 = old)* NO

False? *(LOCK , new==old)* ⇒ *(new + 1 ≠ old)* YES

55



Advanced SLAM/BLAST

Too Many Predicates

- Use Predicates Locally

Counter-Examples

- Craig Interpolants

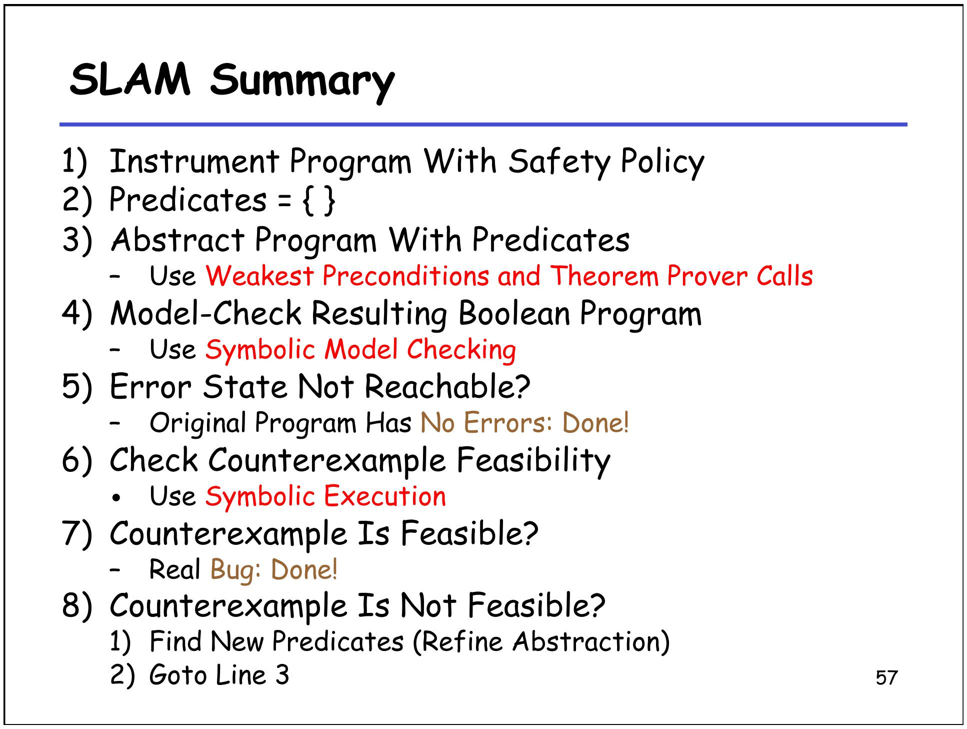
Procedures

- Summaries

Concurrency

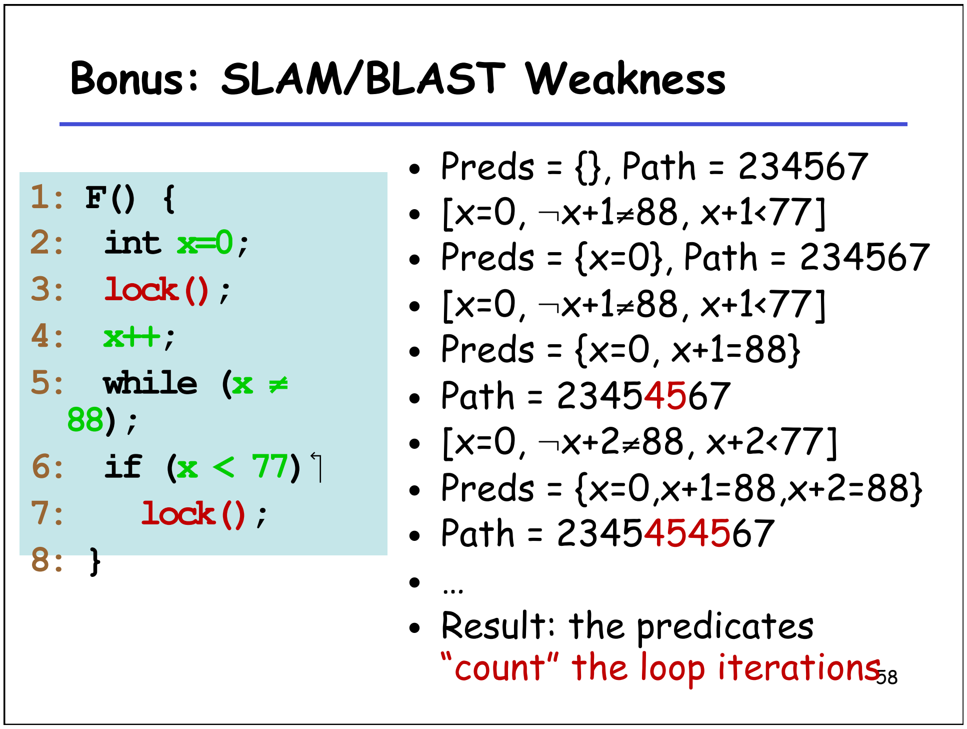
- Thread-Context Reasoning

56



SLAM Summary

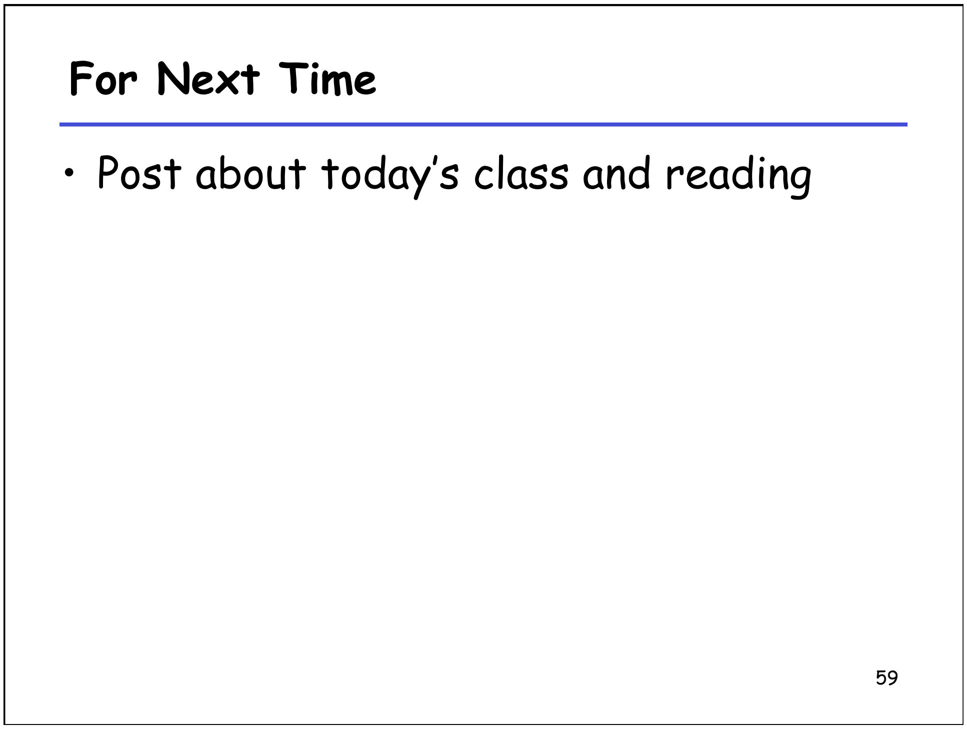
1) Instrument Program With Safety Policy
2) Predicates = { }
3) Abstract Program With Predicates
   - Use Weakest Preconditions and Theorem Prover Calls
4) Model-Check Resulting Boolean Program
   - Use Symbolic Model Checking
5) Error State Not Reachable?
   - Original Program Has No Errors: Done!
6) Check Counterexample Feasibility
   - Use Symbolic Execution
7) Counterexample Is Feasible?
   - Real Bug: Done!
8) Counterexample Is Not Feasible?
   1) Find New Predicates (Refine Abstraction)
   2) Goto Line 3



Bonus: SLAM/BLAST Weakness

```
1: F() {
2:   int x=0;
3:   lock();
4:   x++;
5:   while (x ≠
  88);
6:   if (x < 77)↰
7:      lock();
8: }
```

- Preds = {}, Path = 234567
- [x=0, ¬x+1≠88, x+1<77]
- Preds = {x=0}, Path = 234567
- [x=0, ¬x+1≠88, x+1<77]
- Preds = {x=0, x+1=88}
- Path = 23454567
- [x=0, ¬x+2≠88, x+2<77]
- Preds = {x=0,x+1=88,x+2=88}
- Path = 2345454567
- …
- Result: the predicates "count" the loop iterations



For Next Time

- Post about today’s class and reading