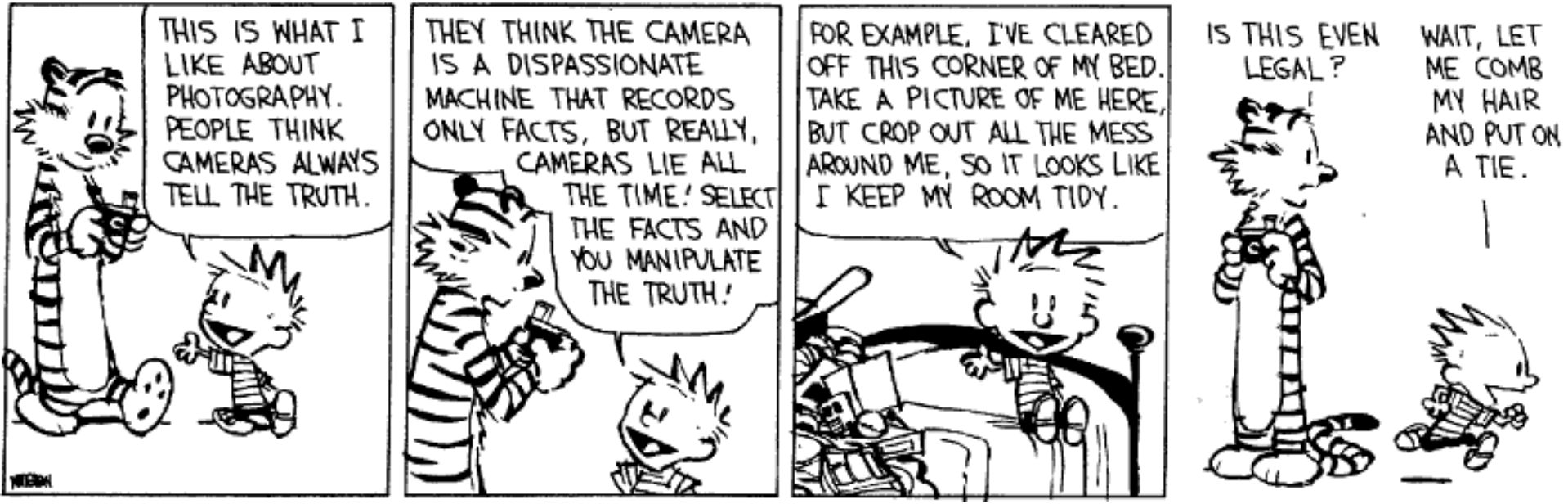


Having a BLAST with SLAM

Topic:
Software Model Checking via Counter-Example Guided Abstraction Refinement

- There are dozens of CEGAR papers; I will skim.

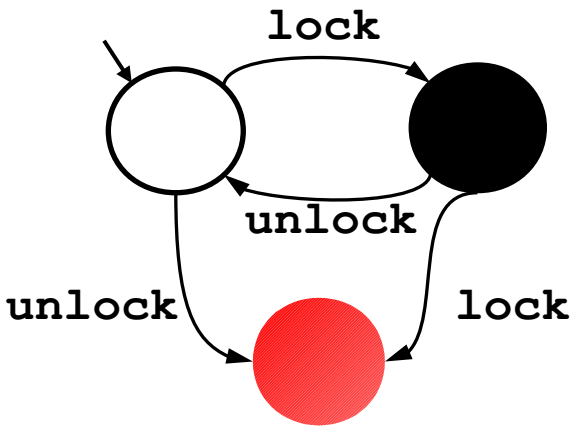
SLAM Overview

- 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. CEGAR is a common method.
- No errors in the boolean program implies no errors in the original.
- An error in the boolean program **may** be a real bug. Or SLAM may **refine** the abstraction and start again.

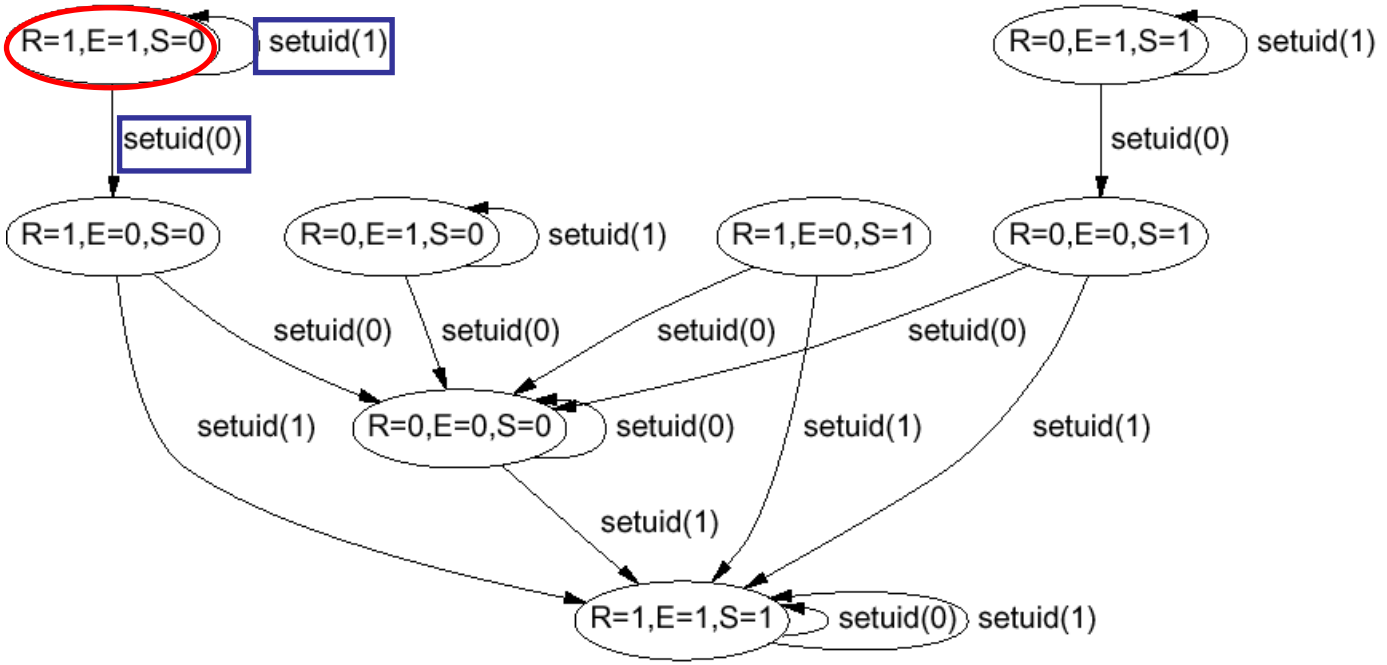
Property 1: Double Locking



*"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

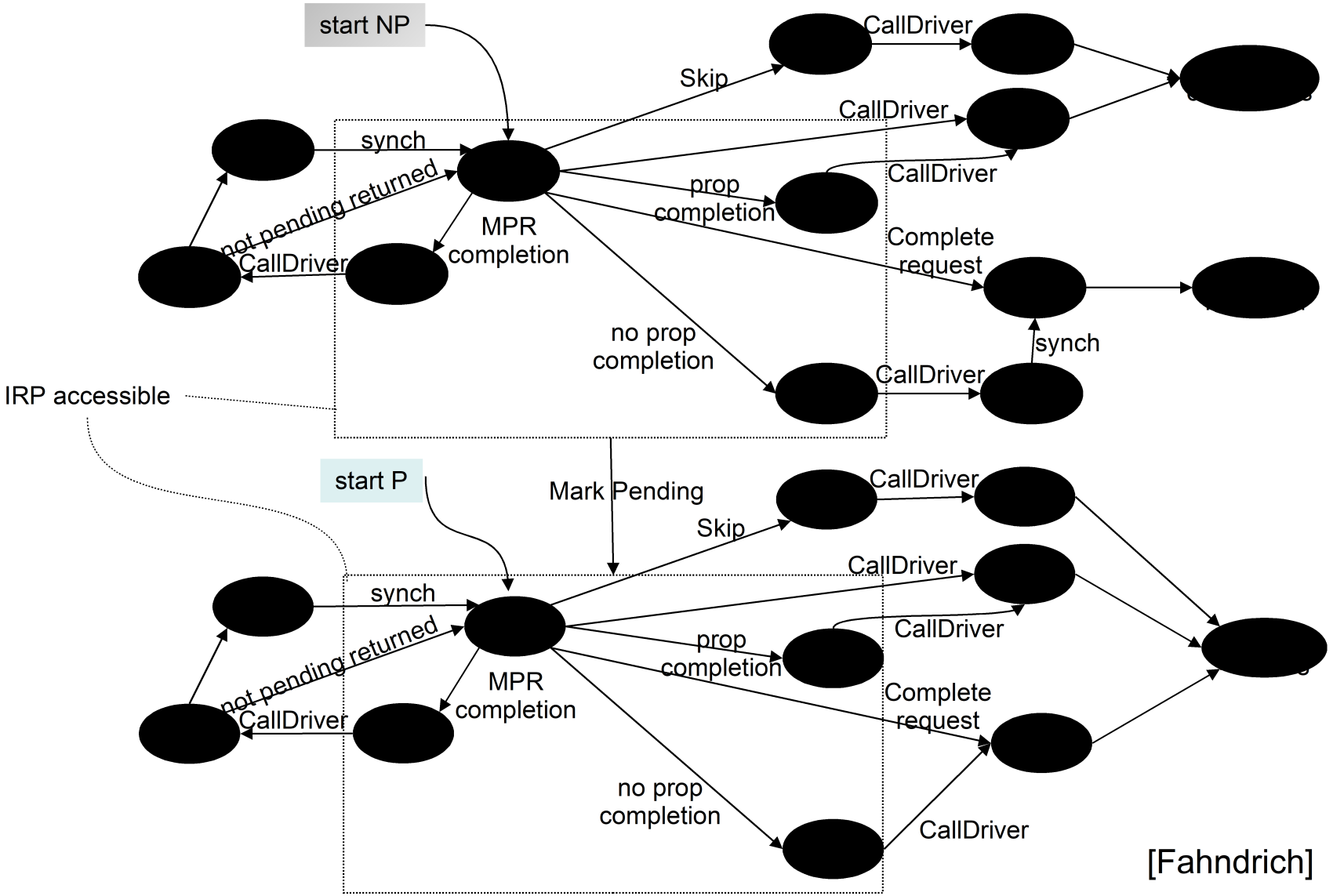


[Chen-Dean-Wagner '02]

"User applications must not run with root privilege"

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

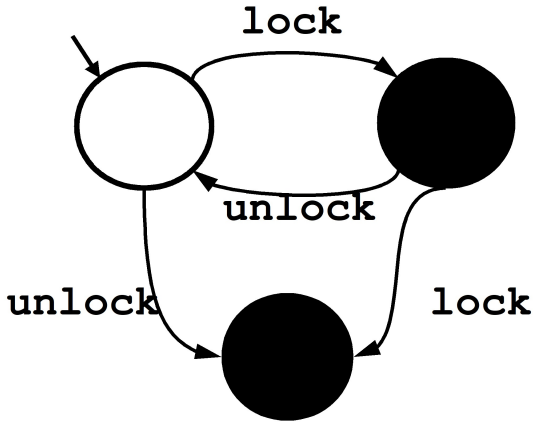
Property 3 : IRP Handler



[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;
}
```

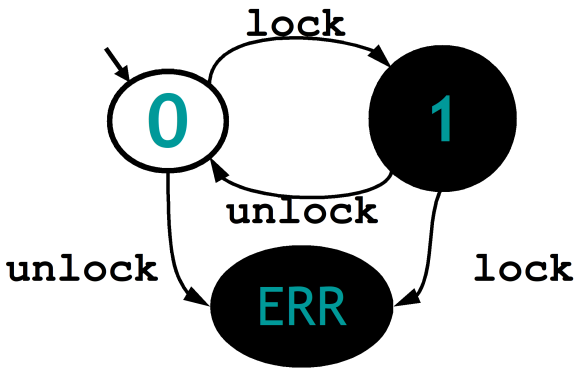


SLAM in a Nutshell

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

Incorporating Specs

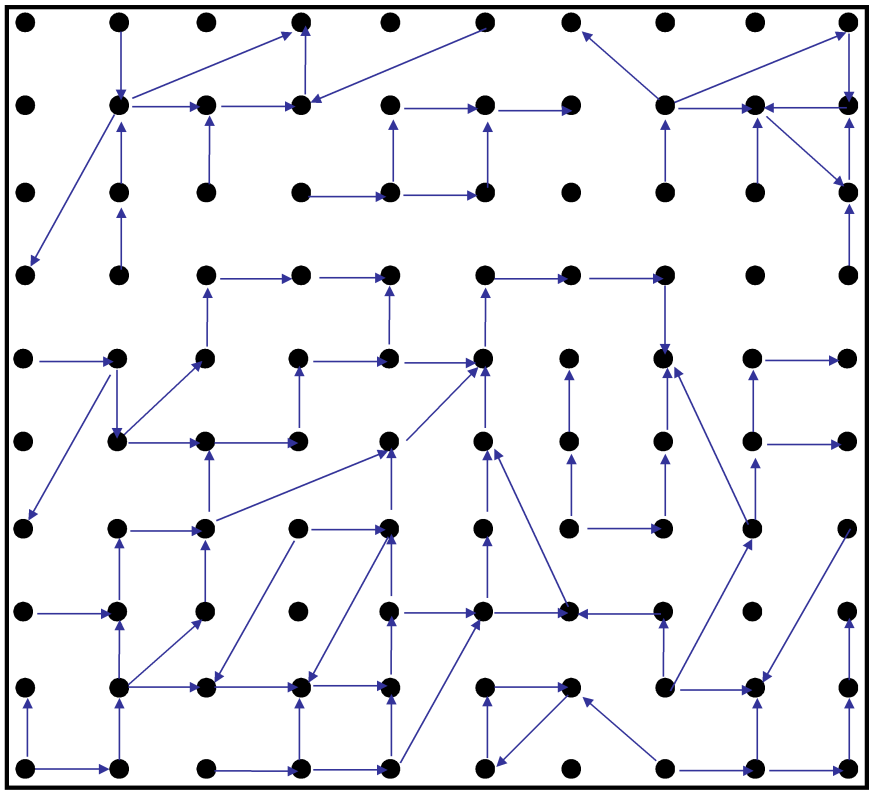
```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
       }
4: } while(new != old);
5:  unlock ();
    return;
}
```




```
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

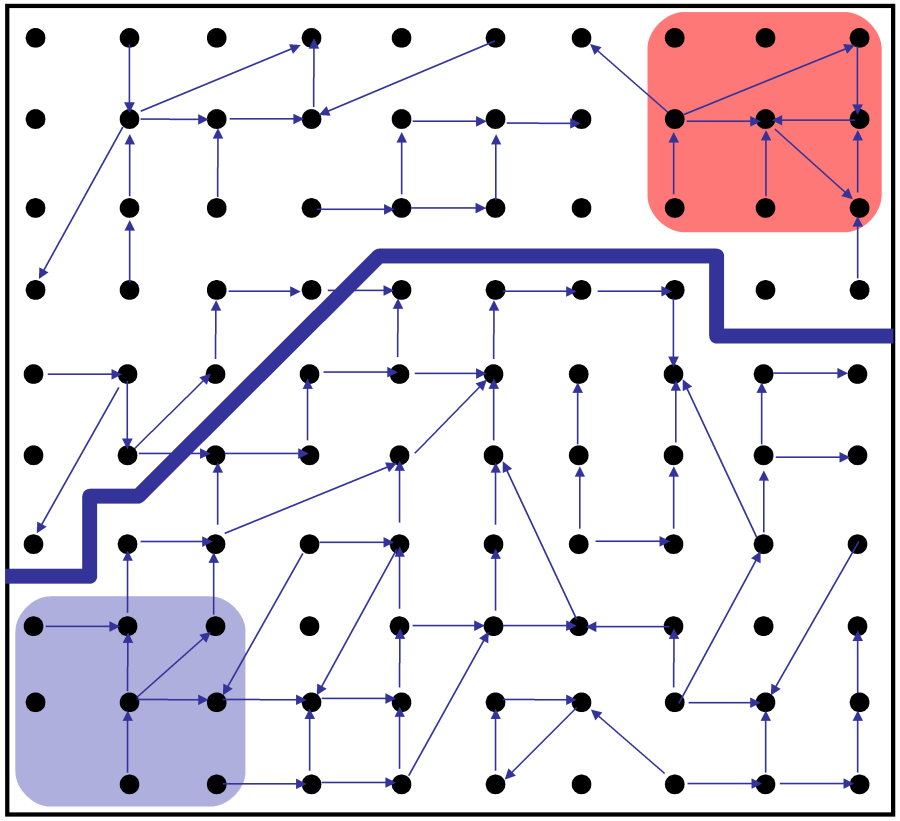
pc ↦ 3
lock ↦ ●
old ↦ 5
new ↦ 5
q ↦ 0x133a

```
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; }
```

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=…)

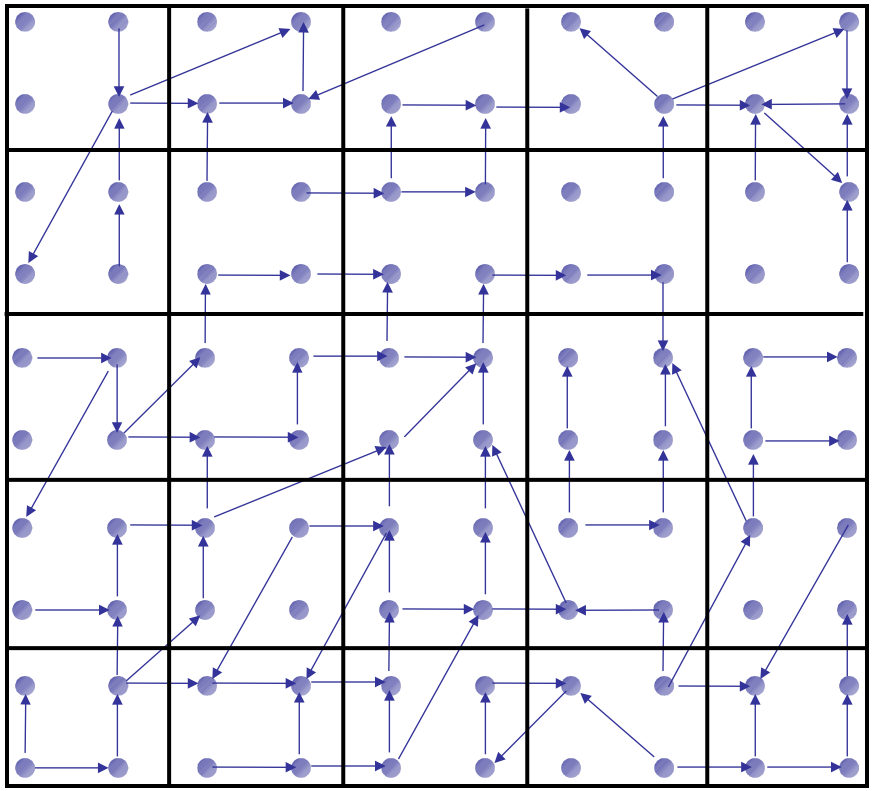
Solution : **Set** of states ≃ logical **formula**

Representing [Sets of States] as *Formulas*

$[F]$ states satisfying F $\{s \mid s \models F\}$	F FO fmla over prog. 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

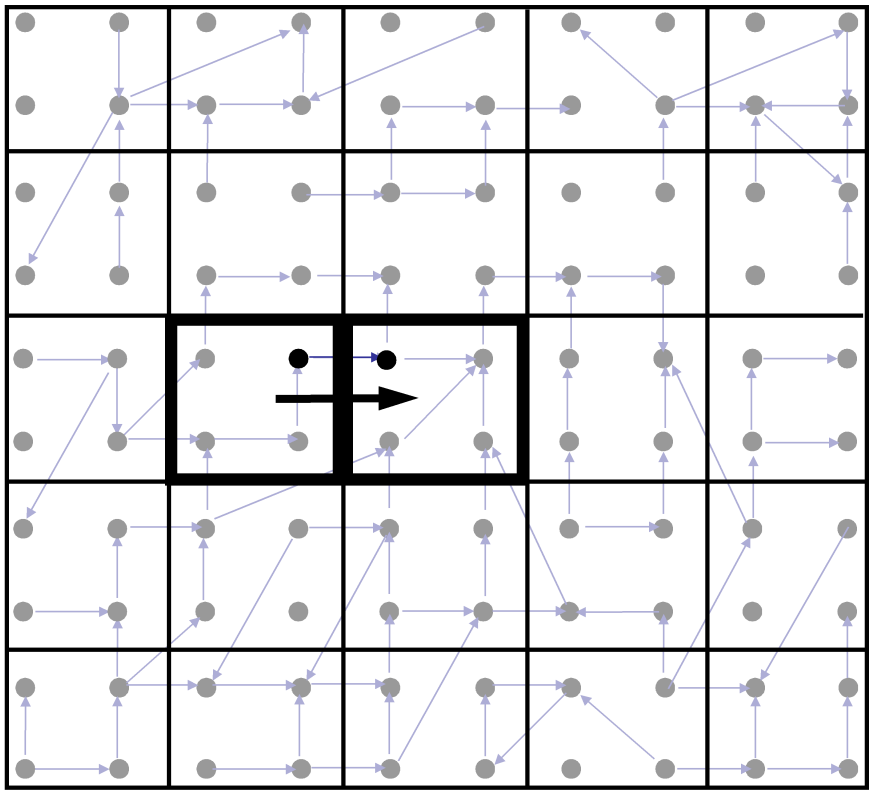


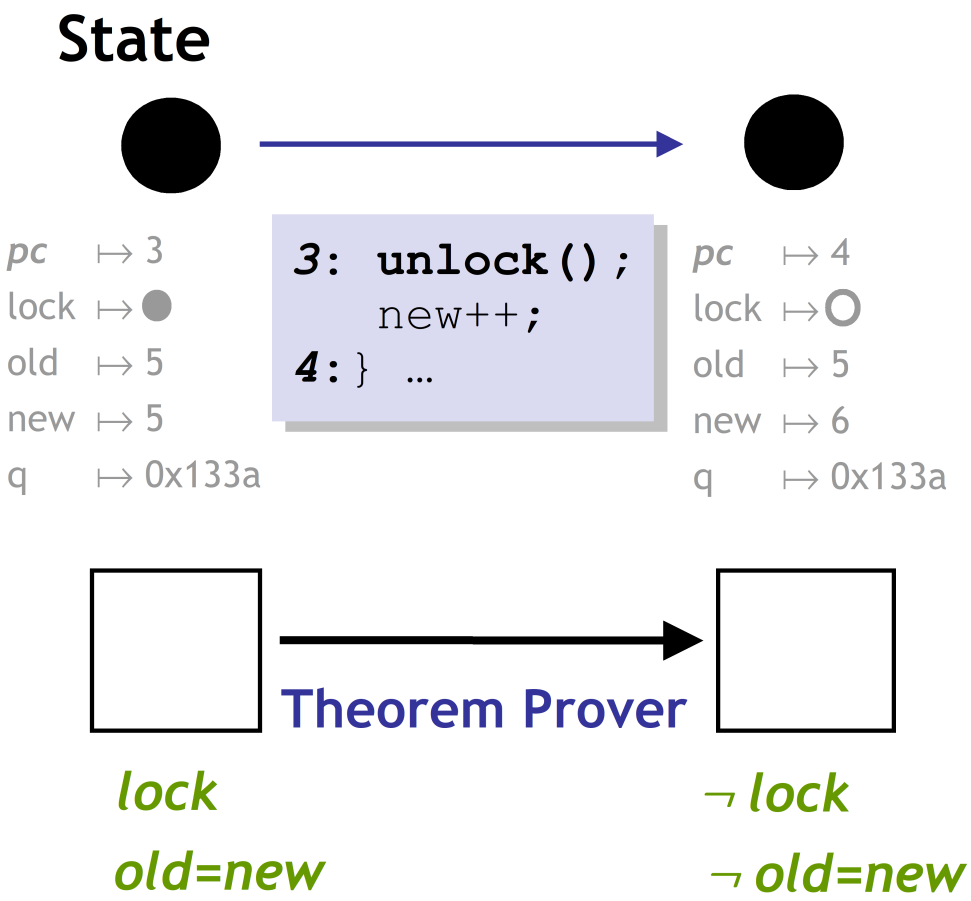
- **Predicates** on program state:
  lock *(i.e., lock=true)*
  old = new

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

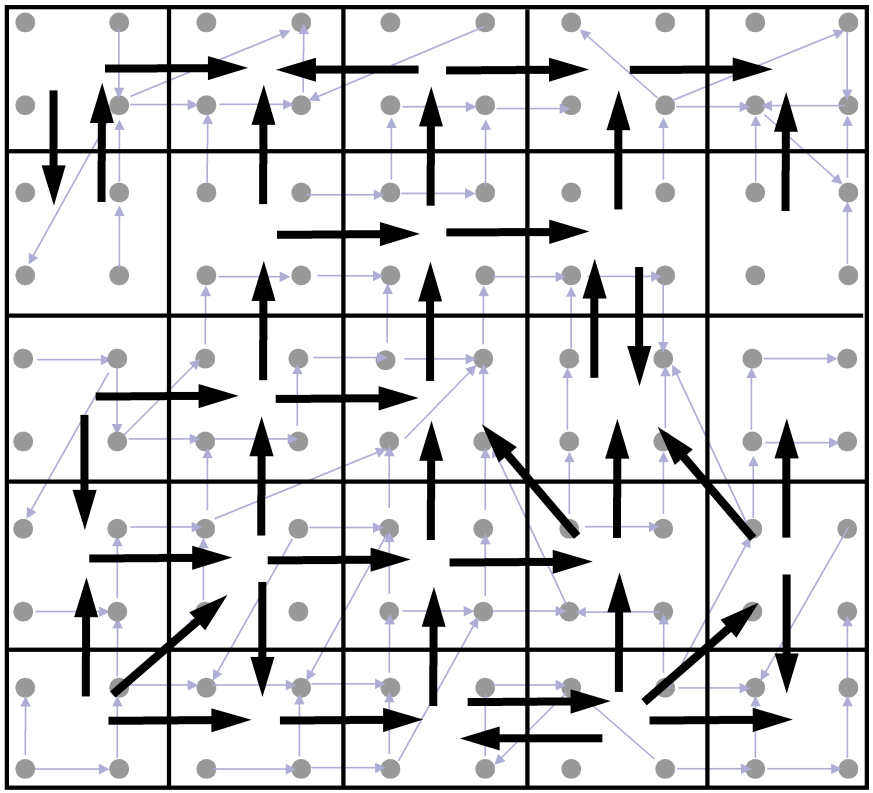
- #abstract states is **finite**
  - **Thus model-checking the abstraction will be feasible!**

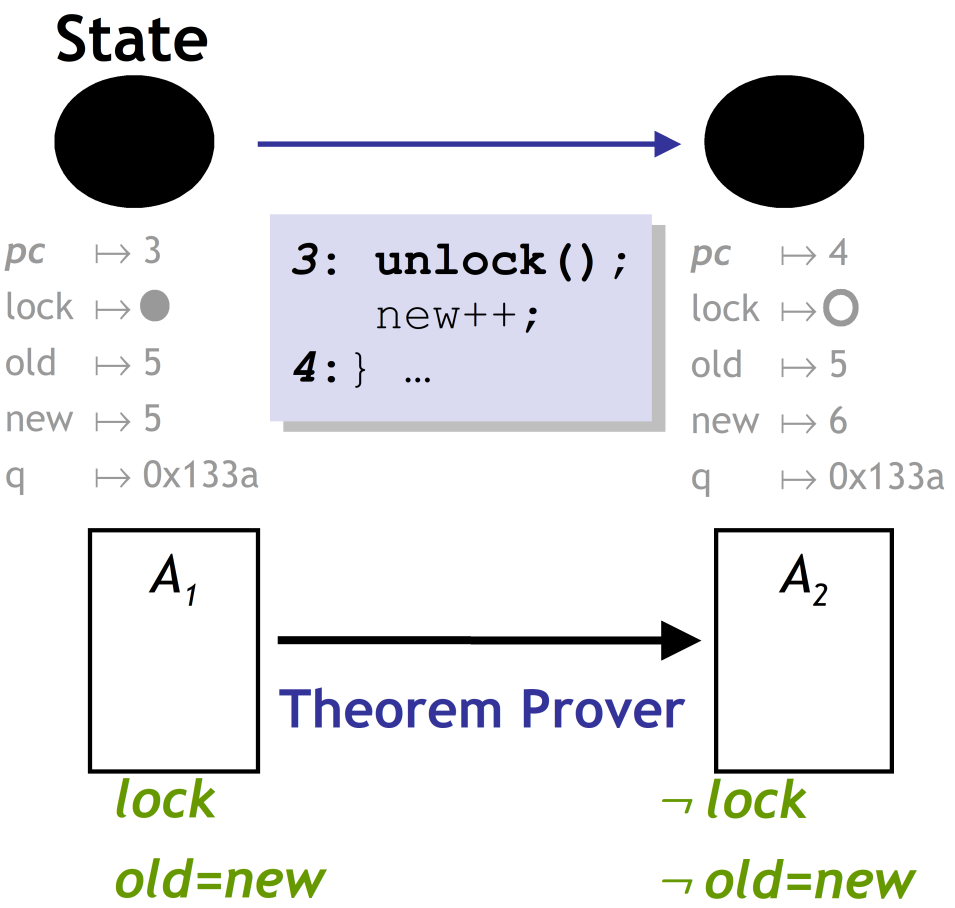
Abstract States and Transitions





Abstraction

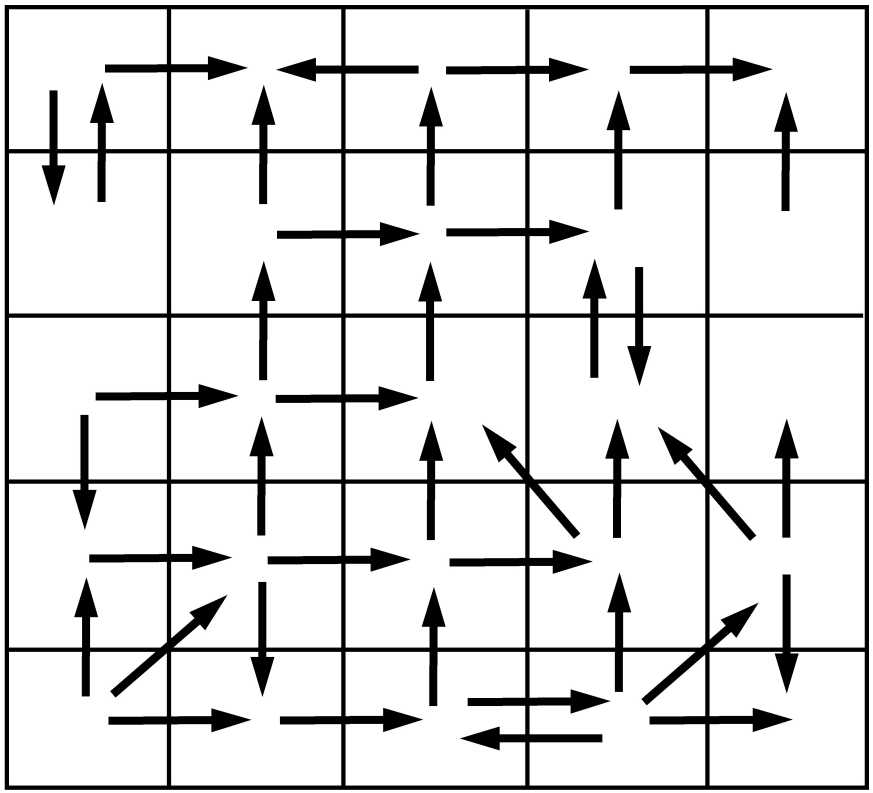




Existential Lifting

(i.e., $A_1 \rightarrow A_2$ iff $\exists c_1 \in A_1.\ \exists c_2 \in A_2.\ c_1 \rightarrow c_2$)

Abstraction

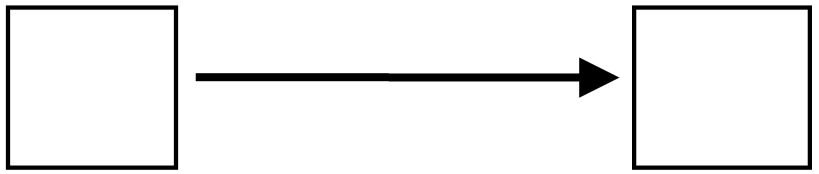


State

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

```
3: unlock();
   new++;
4:} …
```

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



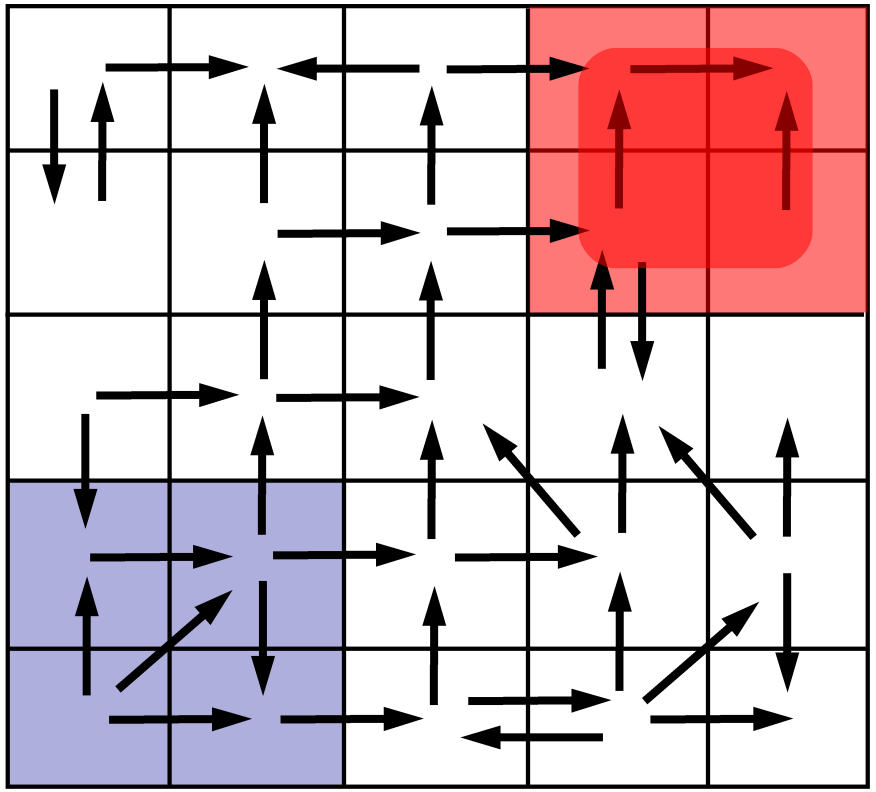
lock

old=new

¬ lock

¬ old=new

Analyze Abstraction



Analyze finite graph

Over Approximate:

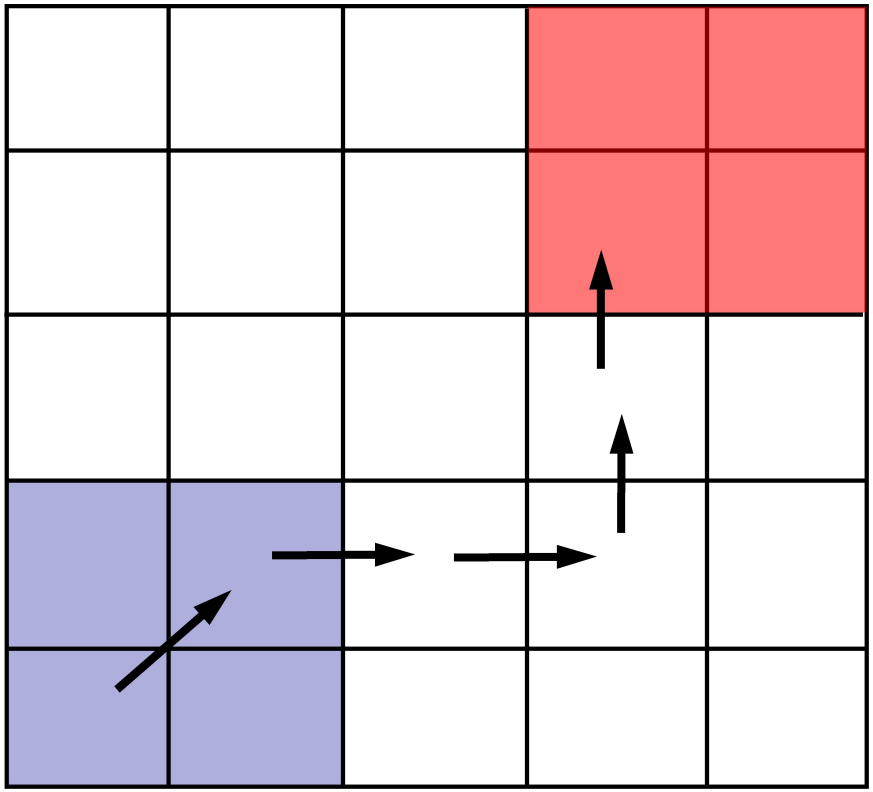
Safe ⇒ System Safe

No **false negatives**

Problem

Spurious **counterexamples**

Idea 2: Counterex.-Guided Refinement

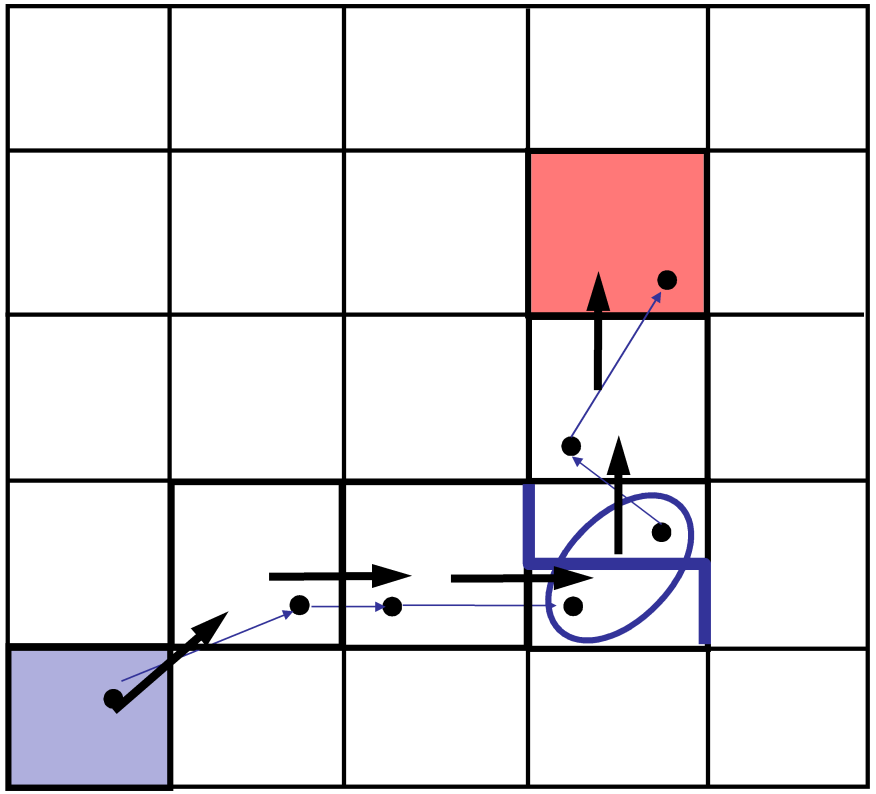


Solution

Use spurious **counterexamples**

to **refine** abstraction!

Idea 2: Counterex.-Guided Refinement



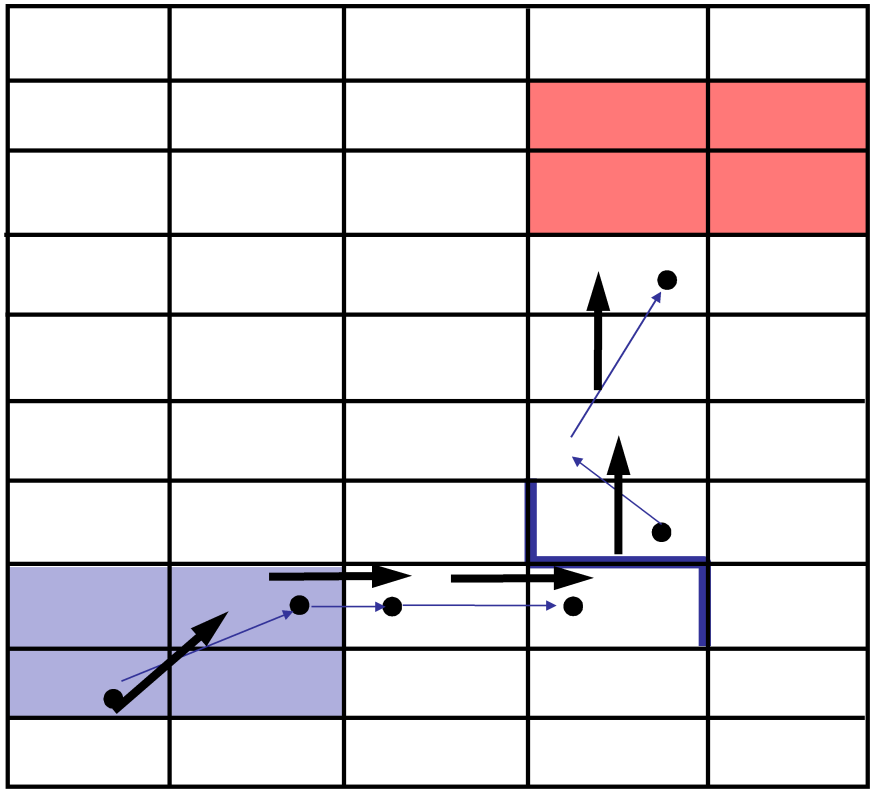
Solution

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



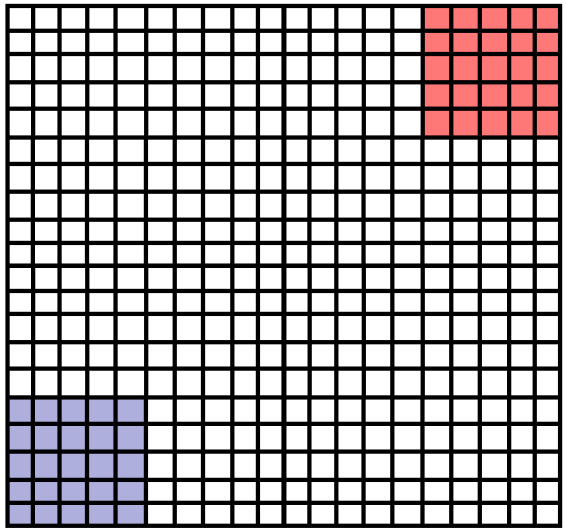
[Kurshan et al 93] [Clarke et al 00]
[Ball-Rajamani 01]

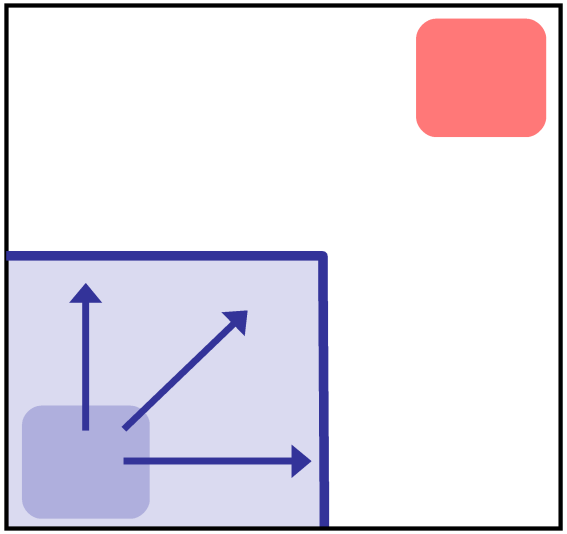
Solution

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

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

Problem: Abstraction is Expensive





Reachable

Problem

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

Exponential Thm. Prover queries

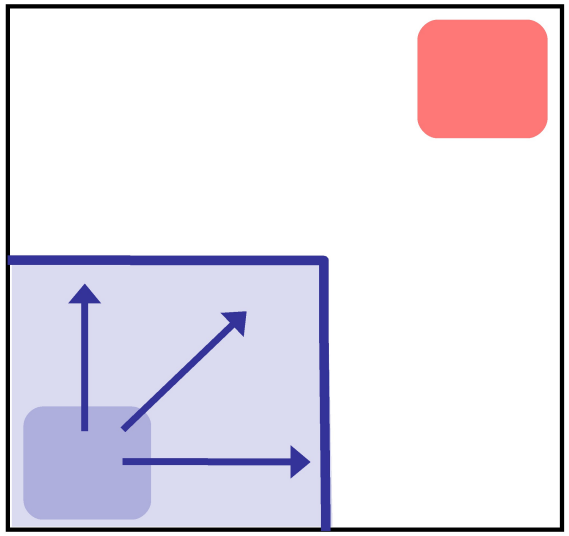
Observe

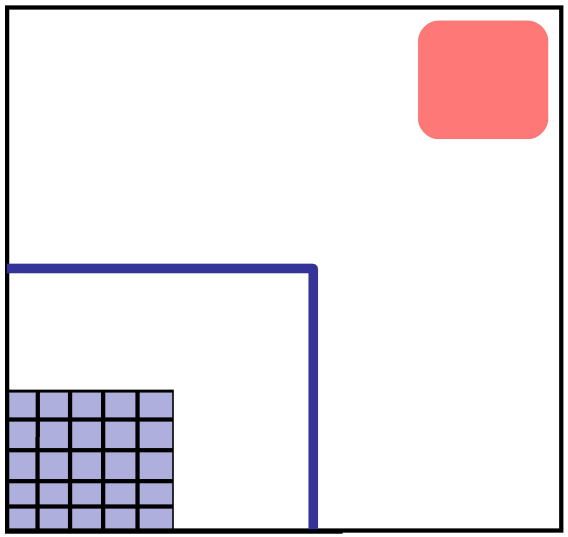
Fraction of state space reachable

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

#Reach ~ 1000's

Solution1: Only Abstract Reachable States





Safe

Problem

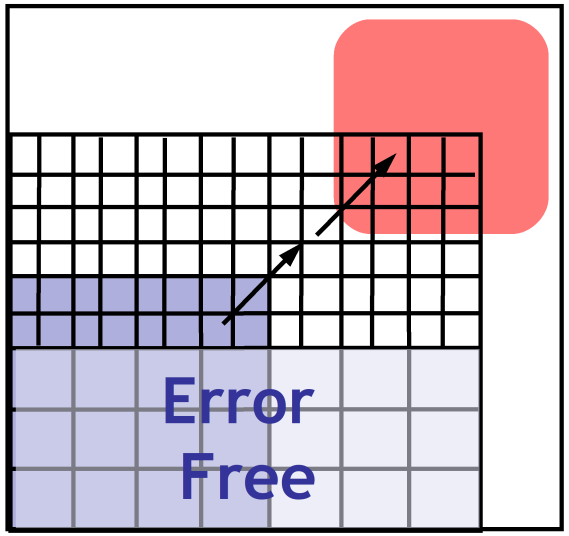
#abstract states = $2^{\#predicates}$

Exponential Thm. Prover queries

Solution

Build abstraction **during** search

Solution2: Don't Refine Error-Free Regions



Problem

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

Exponential Thm. Prover queries

Solution

Don't refine error-free regions

Sanskrit Epics

- The Ramayana (रामायणम्) consists of over 20,000 Sanskrit verses speaking of virtue, relationships, life and culture. It is a significant text in the Hindu tradition with a large influence on classical poets. *This character* is associated with sacrifice, love and purity. She chooses her husband in a heroic contest from among many others and follows him into exile in the forest.

Musical Theater

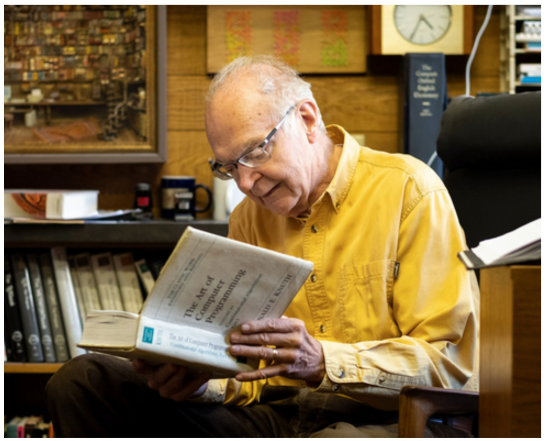
- This 2003 solo is sung by the main character, Elphaba, with two small duets at the beginning and the middle of the song (with Glinda) and a chorus at the end. It capstones the play's first act, in which Elphaba realizes the truth about the Wizard of Oz and vows to fight him, beginning her evolution into the "Wicked Witch of the West".

Textiles

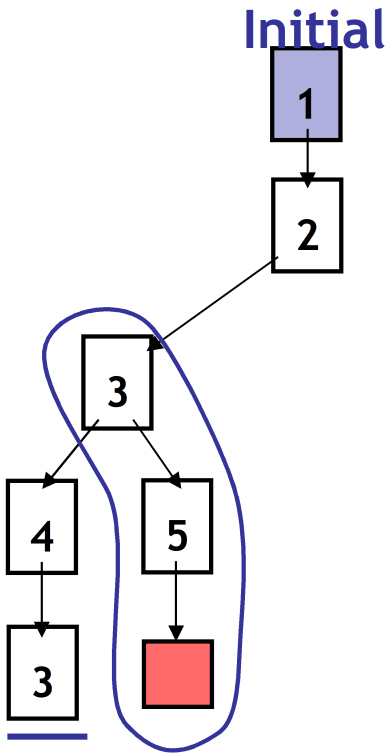
- This twill fabric was originally made from pure cotton. It is commonly used in trousers. Pants of this fabric gained popularity in the U.S. when Spanish–American War veterans returned from the Philippines with their twill military trousers. The American Heritage Dictionary says that the word is from American Spanish for "toasted", in reference to its usual color, but this is not a usual meaning of that Spanish word.

Q: Computer Science

- This American Turing award winner is sometimes called the “father” of analysis of algorithms, and is known for popularizing asymptotic notation, creating TeX, and co-developing a popular a string search algorithm. His most famous work is *The Art of Computer Programming*.



Key Idea: Reachability Tree



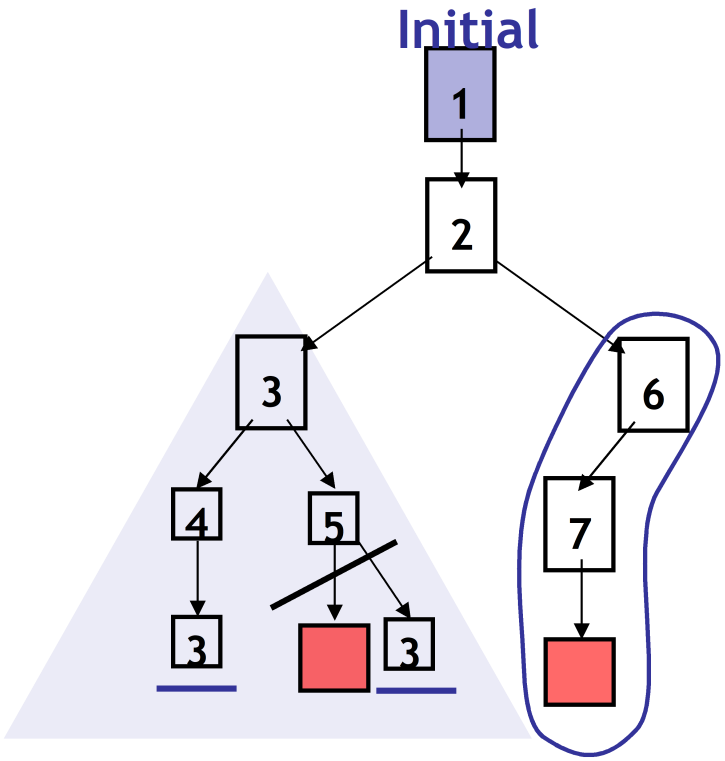
Unroll Abstraction

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



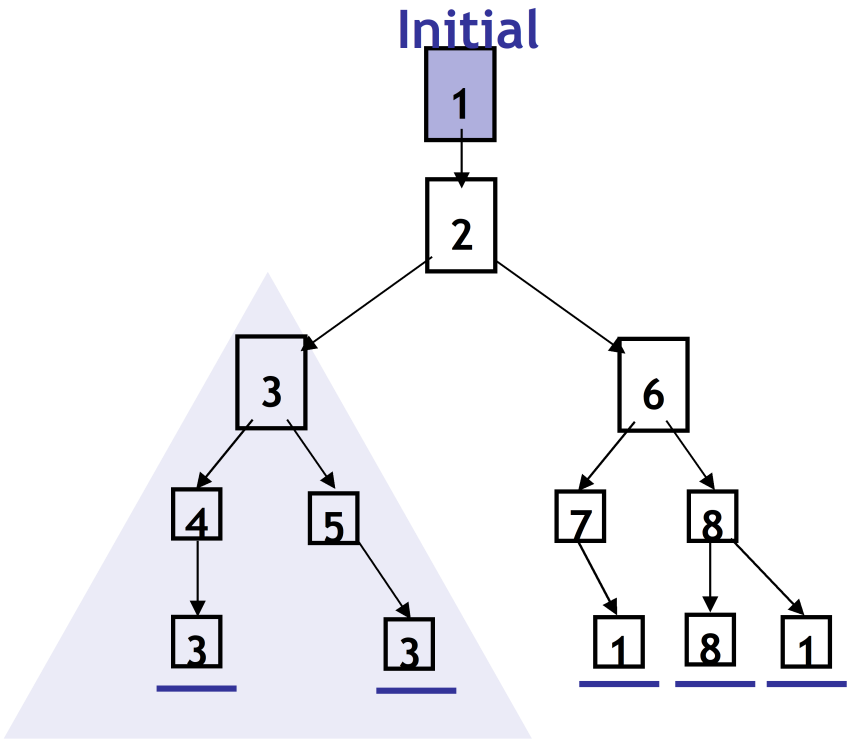
Unroll Abstraction

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

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

Find min spurious suffix

- Learn new predicates
- Rebuild subtree with new preds.

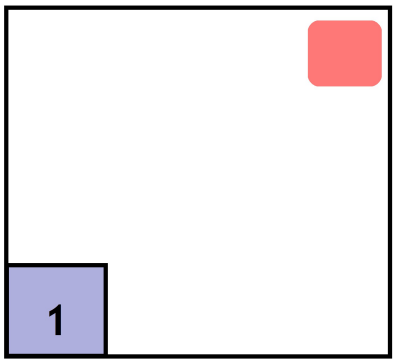
SAFE

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 ();
}
```

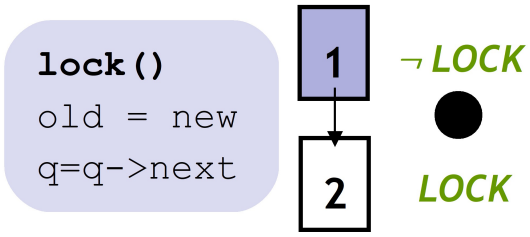


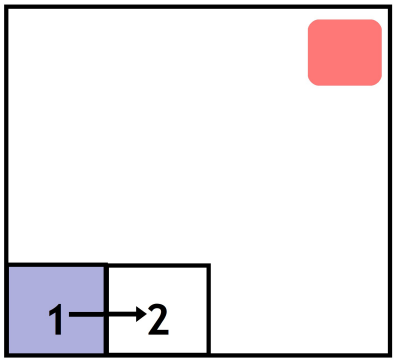
Predicates: *LOCK*

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 ();
}
```



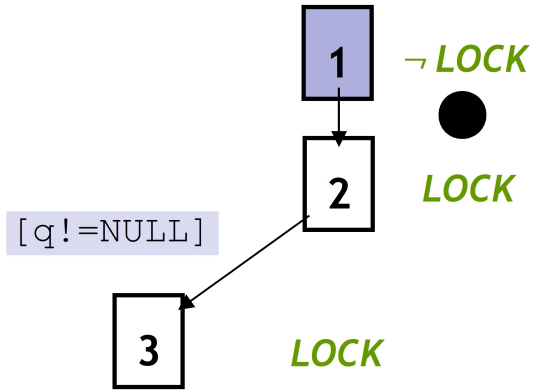


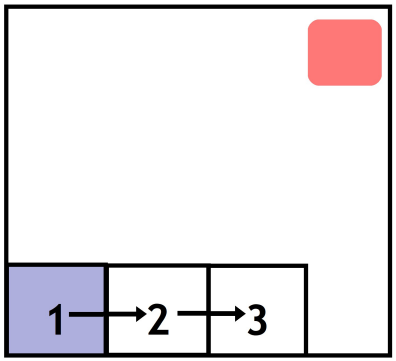
Predicates: *LOCK*

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 ();
}
```



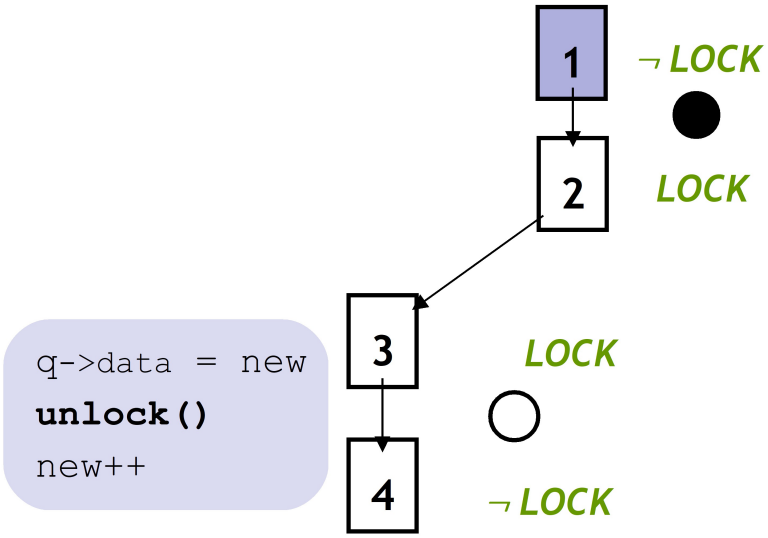


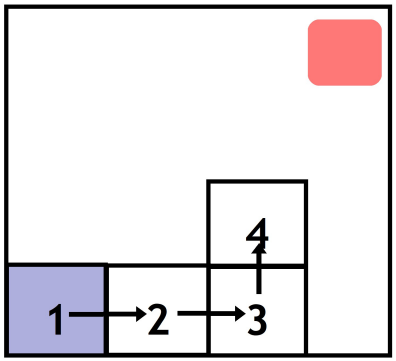
Predicates: *LOCK*

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 ();
}
```



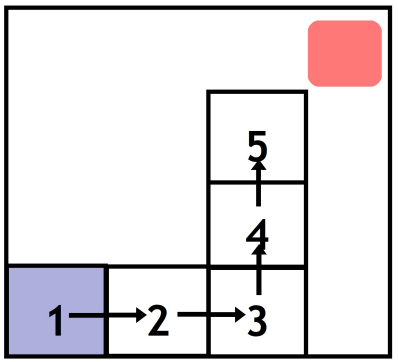


Predicates: *LOCK*

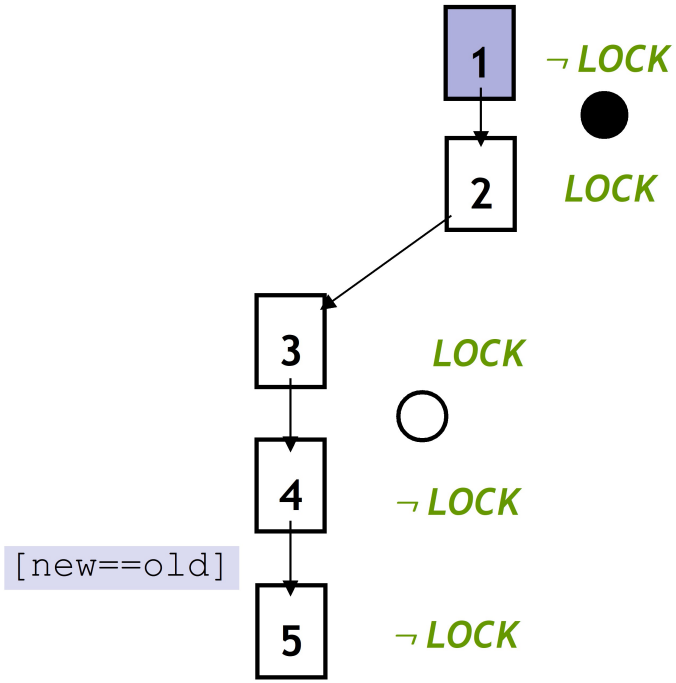
Reachability Tree

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 ();
}
```



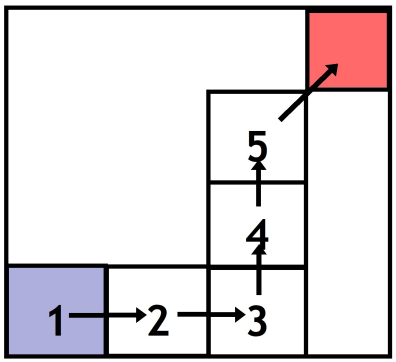
Predicates: *LOCK*



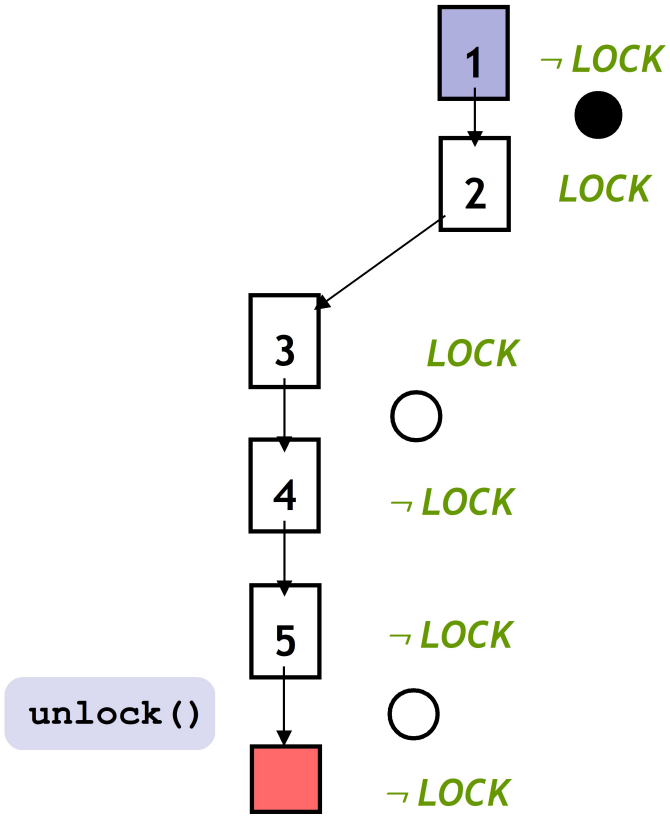
Reachability Tree

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 ();
}
```



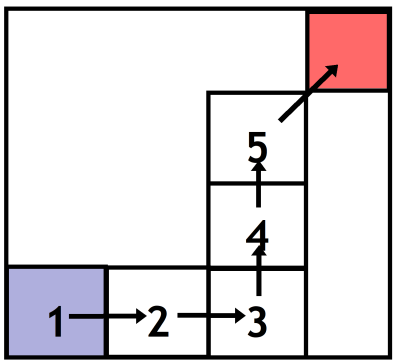
Predicates: *LOCK*



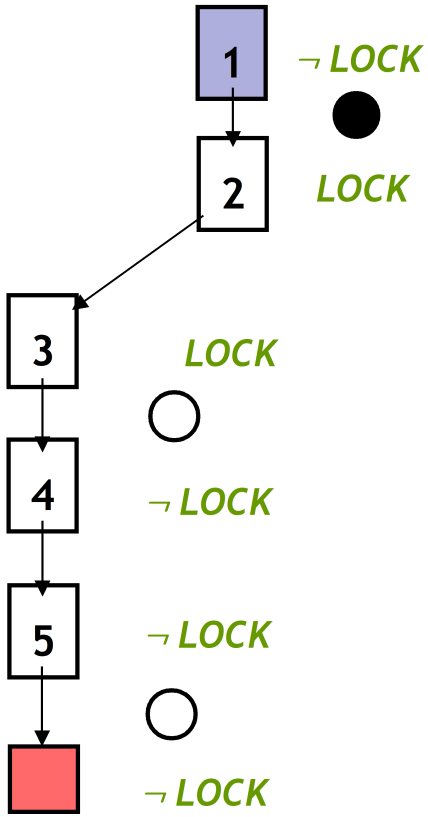
Reachability Tree

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 ();
}
```



Predicates: *LOCK*



lock()
old = new
q=q->next

[q!=NULL]

q->data = new
unlock()
new++

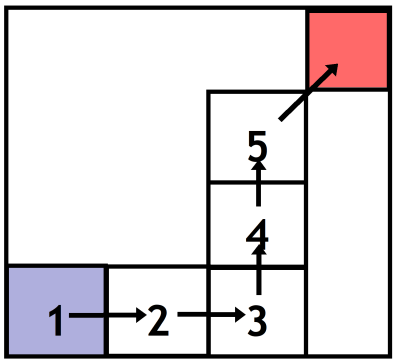
[new==old]

unlock()

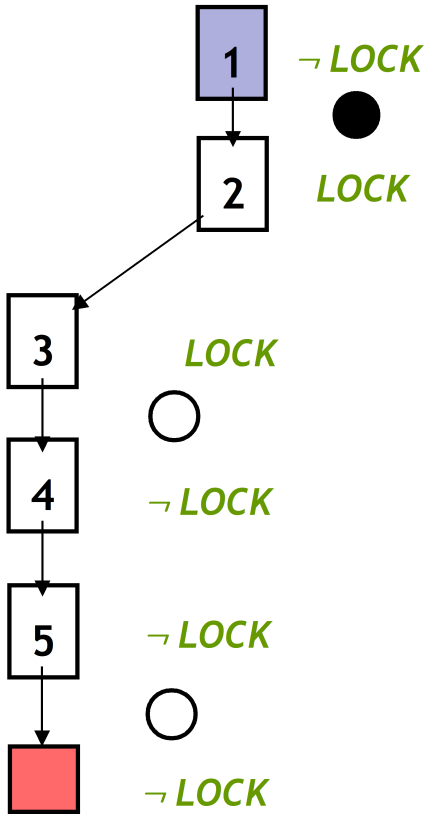
Reachability Tree

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]

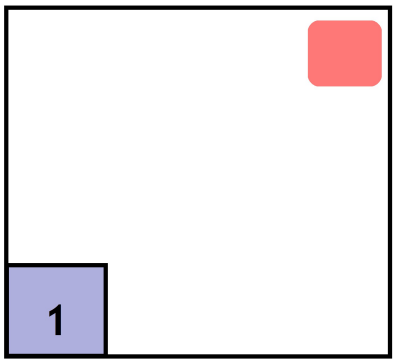
Inconsistent

new == old

Reachability Tree

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 ();
}
```

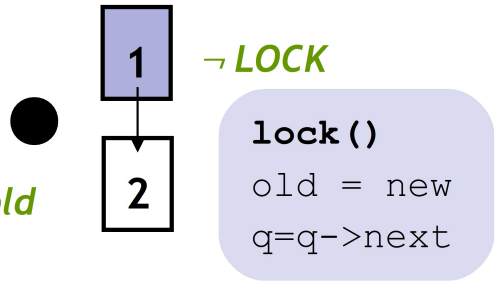


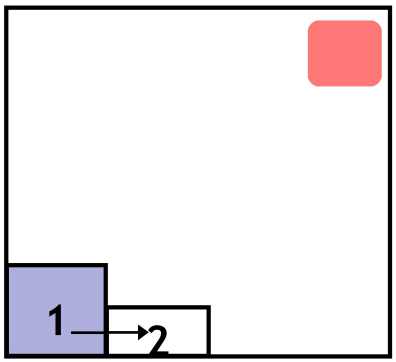
Predicates: *LOCK, new==old*

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 ();
}
```



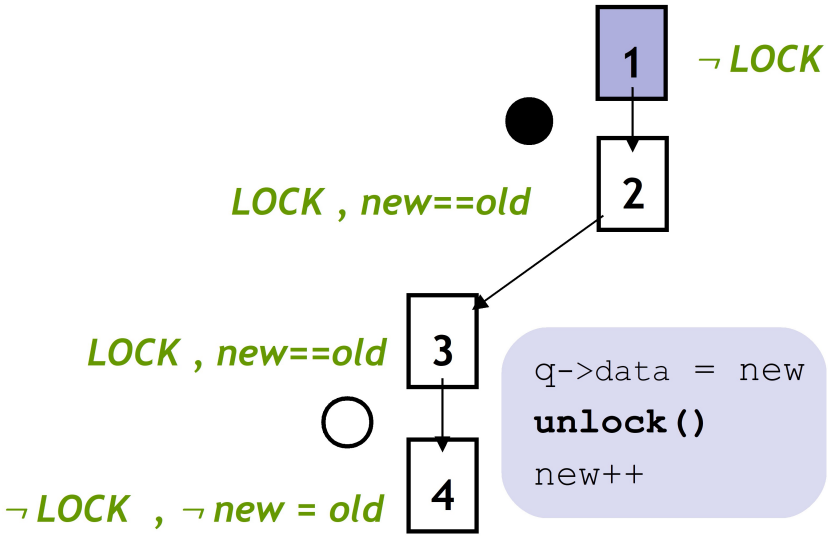


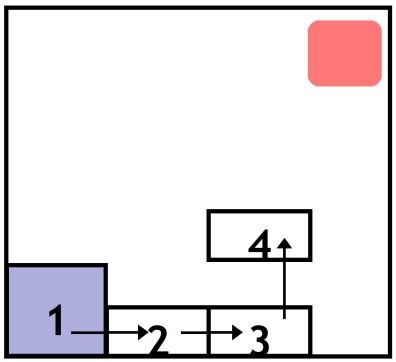
Predicates: *LOCK, new==old*

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 ();
}
```



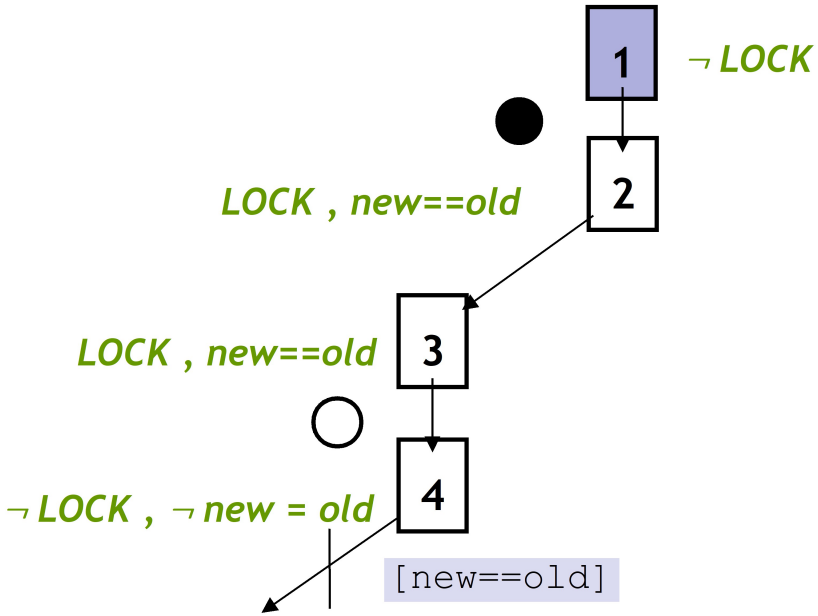


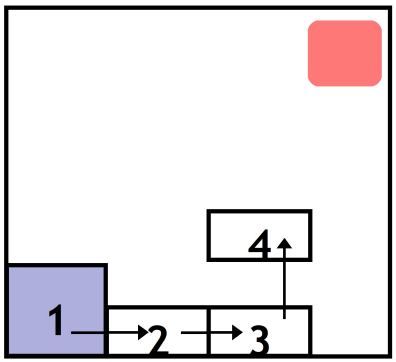
Reachability Tree

Predicates: *LOCK, new==old*

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 ();
}
```



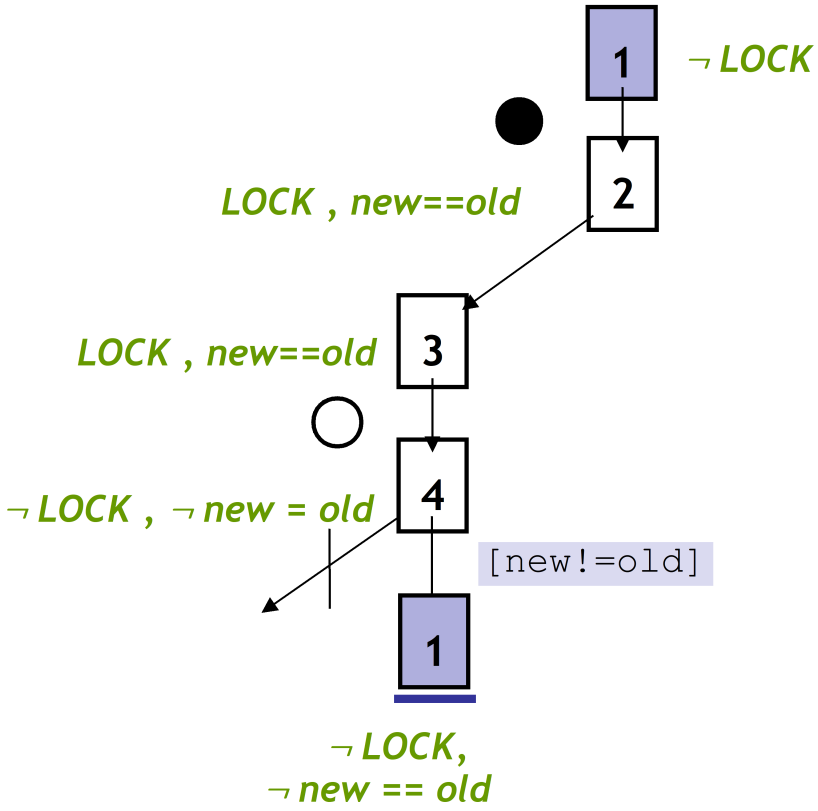


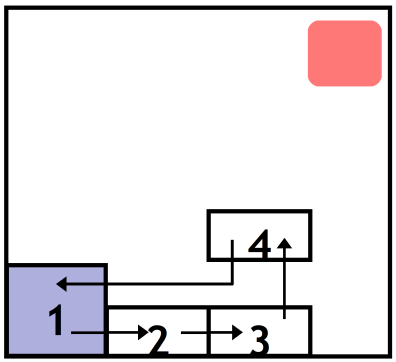
Reachability Tree

Predicates: *LOCK, new==old*

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 ();
}
```



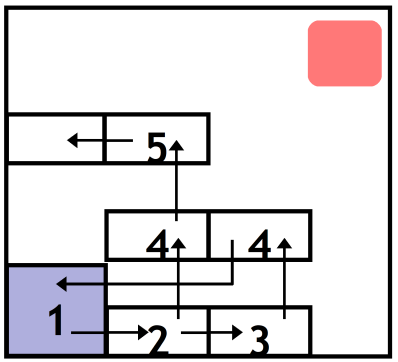


Reachability Tree

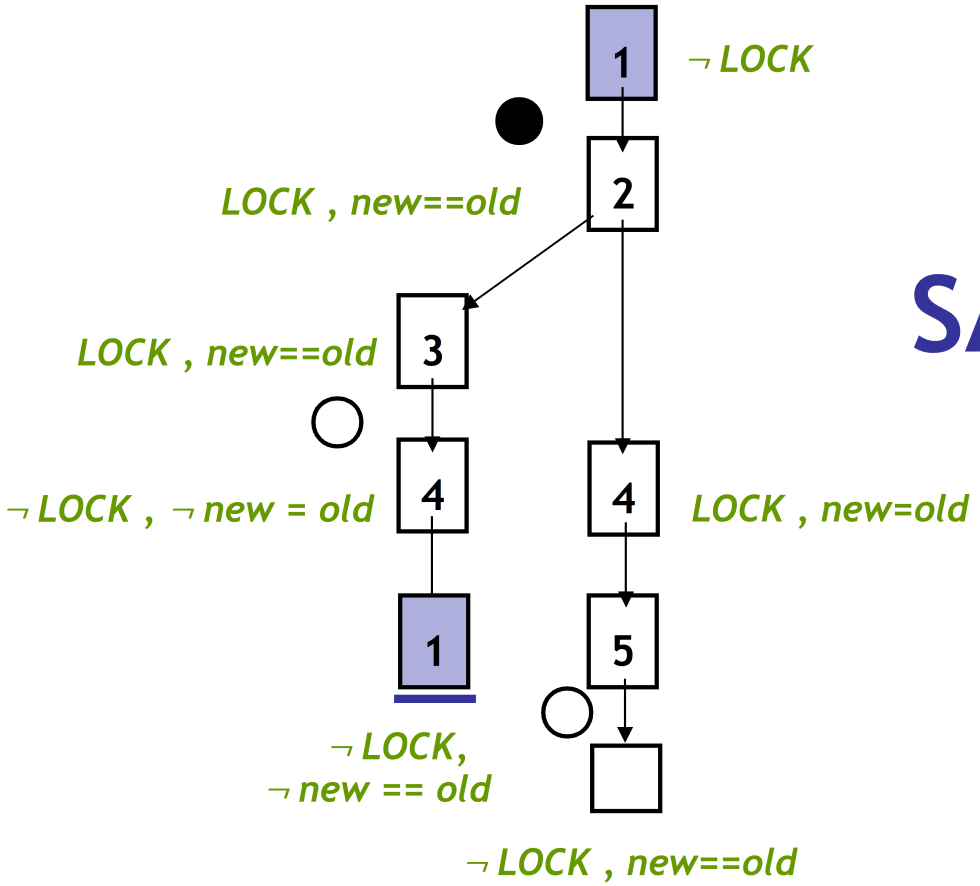
Predicates: *LOCK, new==old*

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 ();
}
```



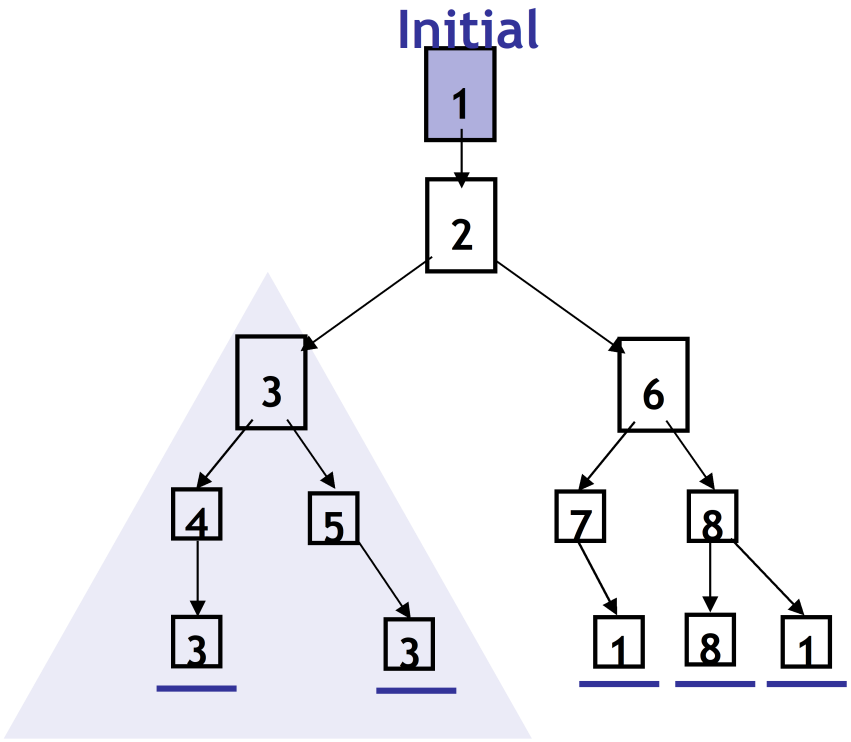
Predicates: *LOCK, new==old*



SAFE

Reachability Tree

Key Idea: Reachability Tree



Unroll

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

Find min spurious suffix

- Learn new predicates
- Rebuild subtree with new preds.

SAFE

S1: Only Abstract Reachable States

S2: Don't refine error-free regions

Two handwaves

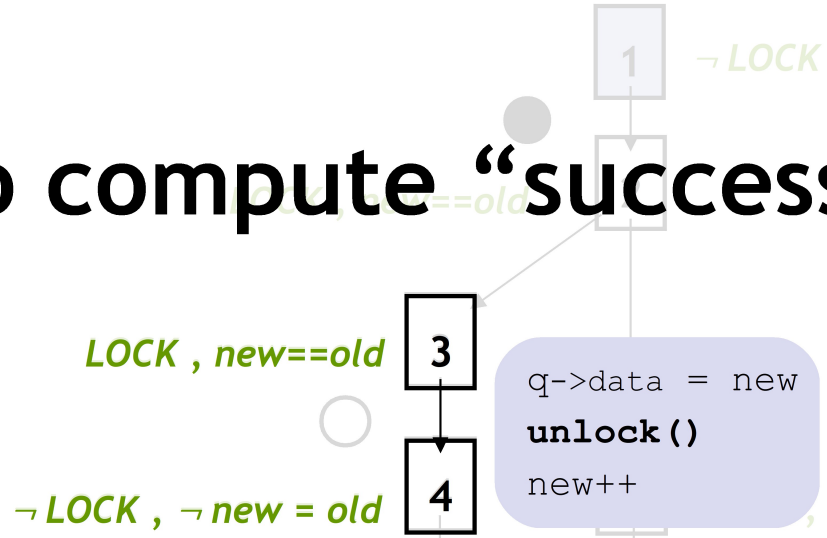
```
Example ( ) {
1: do{
      lock();
      old = new;
      q = q->next;
2:    if (q != NULL){
3:      q->data = new;
        unlock();
        new ++;
      }
4:}while(new != old);
5: unlock ();
}
```


Predicates: LOCK, new==old

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

Two handwaves

Q. How to compute “successors” ?

Q. How to find predicates ?

Refinement

Predicates: *LOCK, new==old*

Two handwaves

Q. How to compute “successors” ?

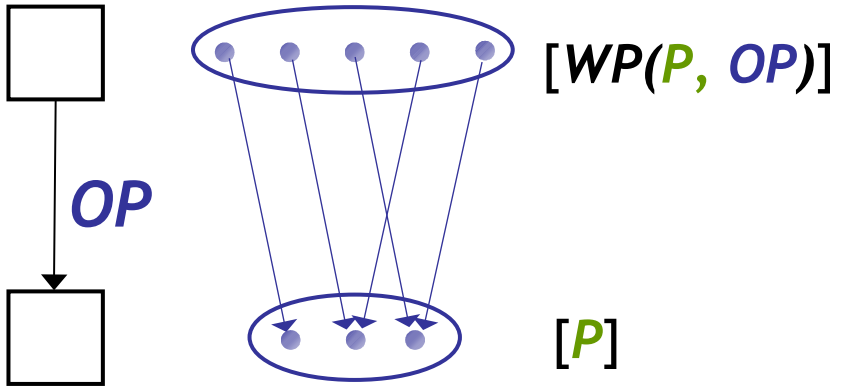
Weakest Preconditions

WP(P,OP)

Weakest formula *P'* s.t.

if *P'* is true before *OP*

then *P* is true after *OP*



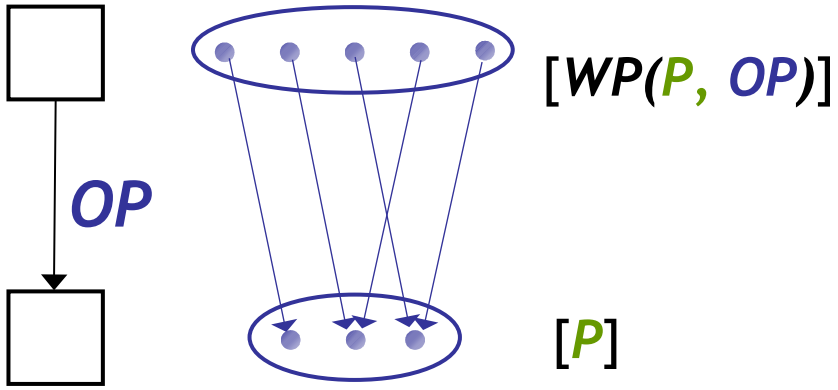
Weakest Preconditions

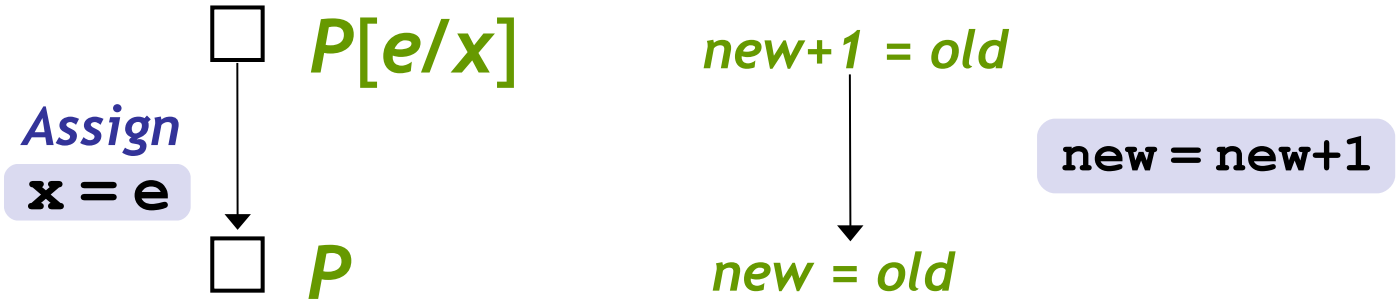
WP(P,OP)

Weakest formula *P'* s.t.

if *P'* is true before *OP*

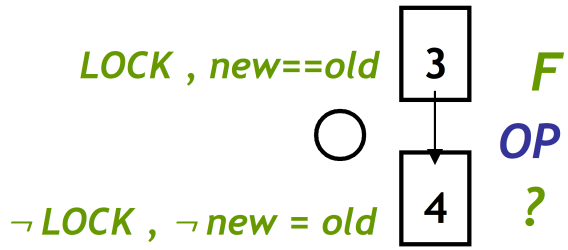
then *P* is true after *OP*





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***

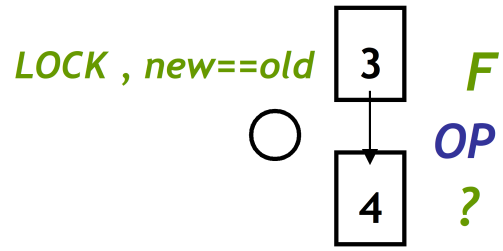
Q: When is ***p*** true after ***OP*** ?

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

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 ();
}7
```



For each *p*

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

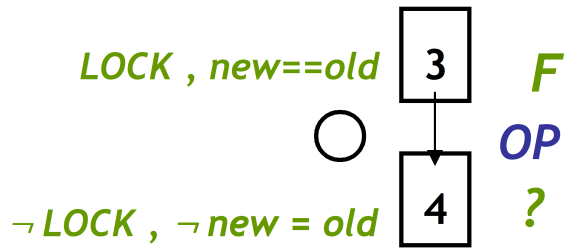
Q: When is ***p*** false <u>after</u> ***OP*** ?

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

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 after *OP* ?

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

Predicate: *new==old*

True ? $(LOCK , new==old) \Rightarrow (new + 1 = old)$ ***NO***

False ? $(LOCK , new==old) \Rightarrow (new + 1 \neq old)$ ***YES***

Advanced SLAM/BLAST

Too Many Predicates

- Use Predicates Locally

Counter-Examples

- Craig Interpolants

Procedures

- Summaries

Concurrency

- Thread-Context Reasoning

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

Optional: SLAM Weakness

```
1: F() {
2:   int x=0;
3:   lock();
4:   do 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 = 2345**45**67
- [x=0, ¬x+2≠88, x+2<77]
- Preds = {x=0,x+1=88,x+2=88}
- Path = 2345**454**567
- …
- Result: the predicates "count" the loop iterations

Homework

- Read Henzinger's Lazy Abstraction
- Optional Reading
- HW0 Due Friday