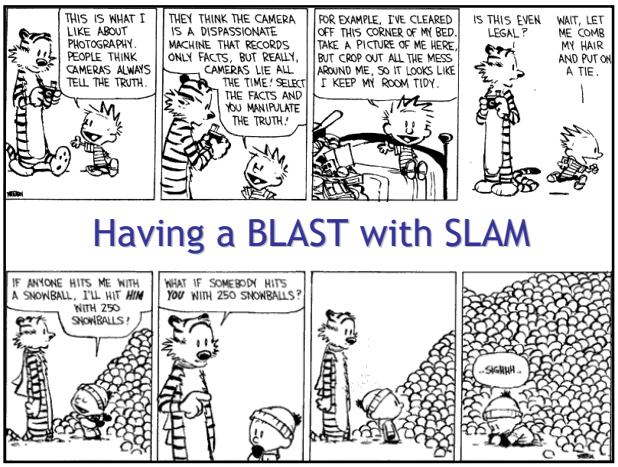


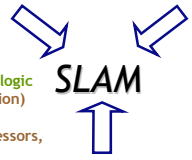
Having a BLAST with SLAM

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

- There are easily two dozen SLAM/BLAST/MAGIC papers; I will skim.

#2

Combining Strengths



Theorem Proving

- Need loop invariants
(will find automatically)
+ Behaviors encoded in logic
(used to refine abstraction)
+ Theorem provers
(used to compute successors, refine abstraction)

Program Analysis

- Imprecise
(will be precise)
+ Abstraction
(will shrink the state space we must explore)

Model Checking

- Finite-state model, state explosion
(will find small good model)
+ State Space Exploration
(used to get a path sensitive analysis)
+ Counterexamples
(used to find relevant facts, refine abstraction)

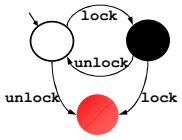
#3

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

#4

Property 1: Double Locking

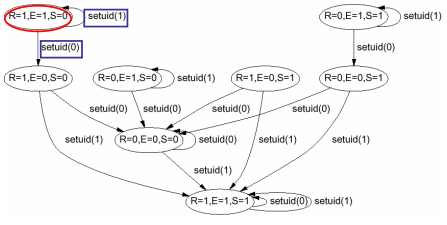


*"An attempt to re-acquire an acquired lock or release a released lock will cause a **deadlock**."*

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

#5

Property 2: Drop Root Privilege



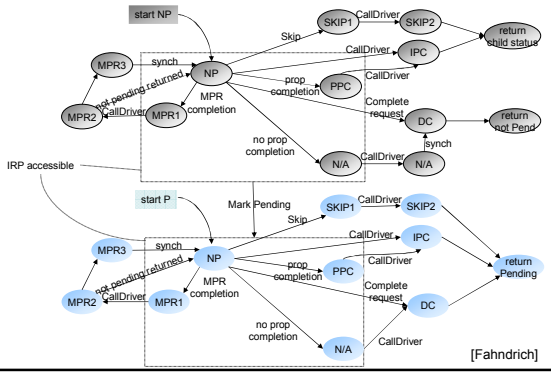
[Chen-Dean-Wagner '02]

"User applications must not run with root privilege"

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

#6

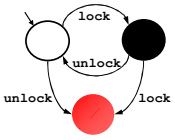
Property 3 : IRP Handler



[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;
}
```

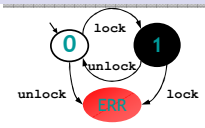


SLAM in a Nutshell

```
SLAM(Program p, Spec s) =                          // program name
 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 → 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

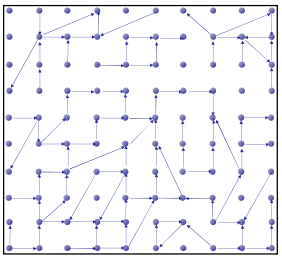
```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:       q->data = new;
         unlock();
         new ++;
       }
4: } while(new != old);
5:  unlock ();
    return;
}
```

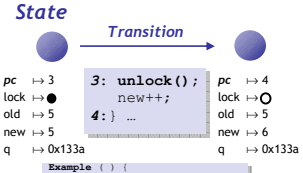



```
Example ( ) {
1: do{
       if L=1 goto ERR;
       else L=1;
       old = new;
       q = q->next;
2:     if (q != NULL){
3:       q->data = new;
         if L=0 goto ERR;
         else L=0;
         new ++;
       }
4: } while(new != old);
5:  if L=0 goto ERR;
    else L=0;
    return;
ERR: abort();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

Original program violates spec iff new program reaches ERR

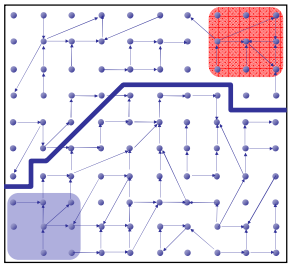
Program As Labeled Transition System






```
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

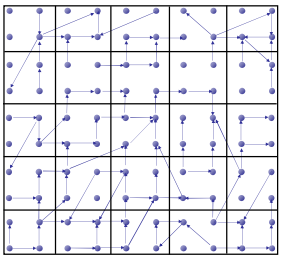
Representing [Sets of States] as *Formulas*

$[F]$ 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

#13

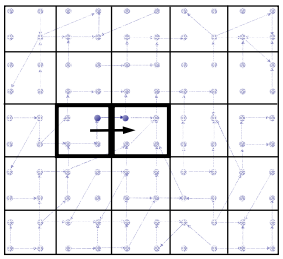
Idea 1: Predicate Abstraction

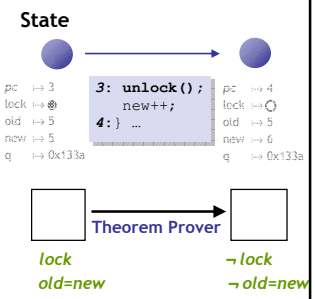


- Predicates on program state:
  - *lock* *(i.e., lock=true)*
  - *old = new*
- States satisfying same predicates are equivalent
  - Merged into one abstract state
- #abstract states is finite
  - Thus model-checking the abstraction will be feasible!

#14

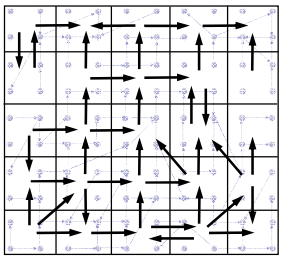
Abstract States and Transitions

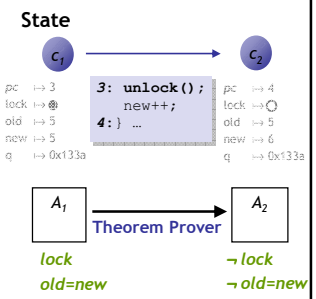




#15

Abstraction

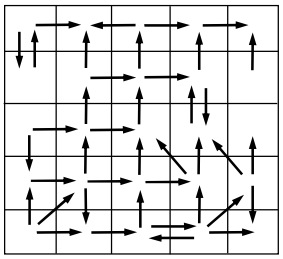


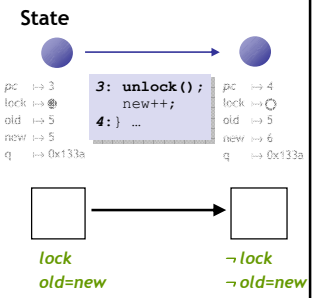


Existential Lifting
(i.e., $A_1 \rightarrow A_2$ iff $\exists c_1 \in A_1.\ \exists c_2 \in A_2.\ c_1 \rightarrow c_2$)

#16

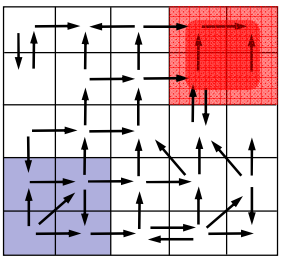
Abstraction





#17

Analyze Abstraction



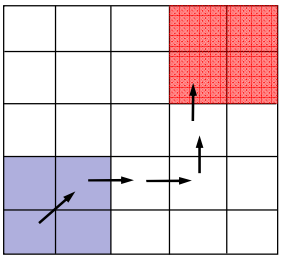
Analyze finite graph

Over Approximate:
Safe ⇒ System Safe
No **false negatives**

Problem
Spurious **counterexamples**

#18

Idea 2: Counterex.-Guided Refinement

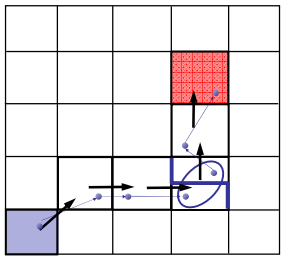


Solution

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

#19

Idea 2: Counterex.-Guided Refinement



Solution

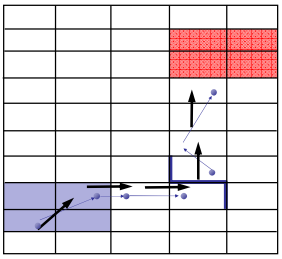
Use spurious **counterexamples** to **refine** abstraction

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

Imprecision due to **merge**

#20

Iterative Abstraction-Refinement



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

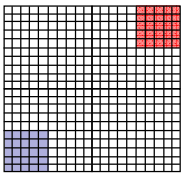
Solution

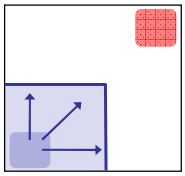
Use spurious **counterexamples** to **refine** abstraction

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

#21

Problem: Abstraction is Expensive





Reachable

Problem

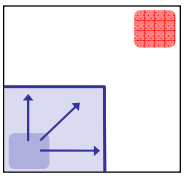
#abstract states = $2^{\#predicates}$
Exponential Thm. Prover queries

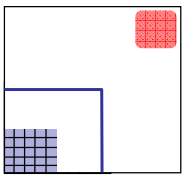
Observe

Fraction of state space reachable
#Preds ~ 100's, #States ~ 2^{100} ,
#Reach ~ 1000's

#22

Solution1: Only Abstract Reachable States





Safe

Problem

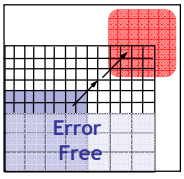
#abstract states = $2^{\#predicates}$
Exponential Thm. Prover queries

Solution

Build abstraction **during** search

#23

Solution2: Don't Refine Error-Free Regions



Problem

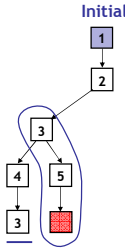
#abstract states = $2^{\#predicates}$
Exponential Thm. Prover queries

Solution

Don't refine error-free regions

#24

Key Idea: Reachability Tree



Unroll Abstraction

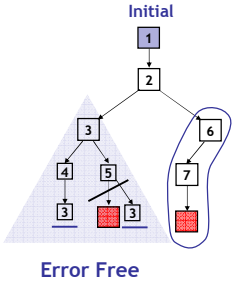
1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

- Learn new predicates
- Rebuild subtree with new preds.

#25

Key Idea: Reachability Tree



Error Free

Unroll Abstraction

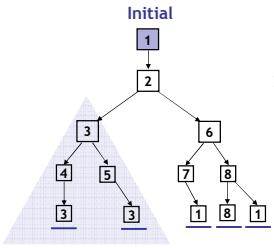
1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visiting abs. state, cut-off

Find min infeasible suffix

- Learn new predicates
- Rebuild subtree with new preds.

#26

Key Idea: Reachability Tree



Error Free

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

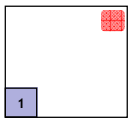
S1: Only Abstract Reachable States

S2: Don't refine error-free regions

#27

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*

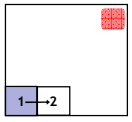
Reachability Tree

#28

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



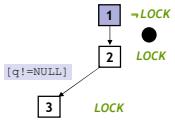
Predicates: *LOCK*

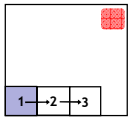
Reachability Tree

#29

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 ();
}
```





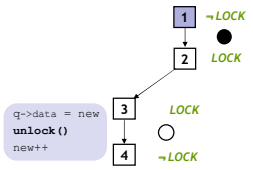
Predicates: *LOCK*

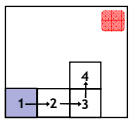
Reachability Tree

#30

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 ();
}
```





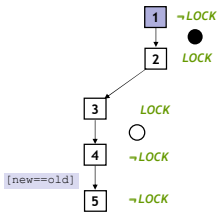
Predicates: *LOCK*

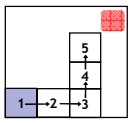
Reachability Tree

#31

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 ();
}
```





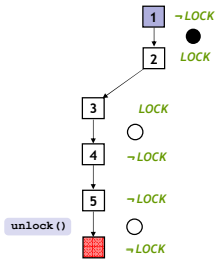
Predicates: *LOCK*

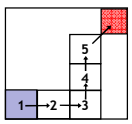
Reachability Tree

#32

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 ();
}
```





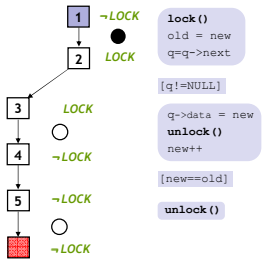
Predicates: *LOCK*

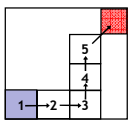
Reachability Tree

#33

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 ();
}
```





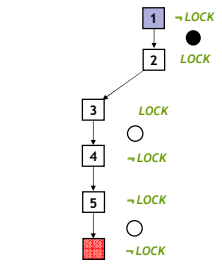
Predicates: *LOCK*

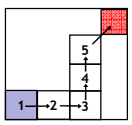
Reachability Tree

#34

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 ();
}
```





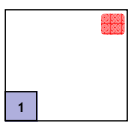
Predicates: *LOCK*

Reachability Tree

#35

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 ();
}
```



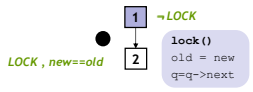
Predicates: *LOCK, new==old*

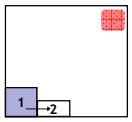
Reachability Tree

#36

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 ();
}
```





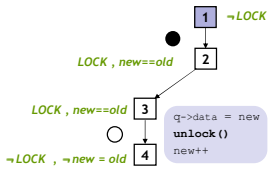
Reachability Tree

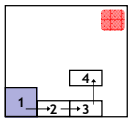
Predicates: *LOCK, new==old*

#37

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 ();
}
```





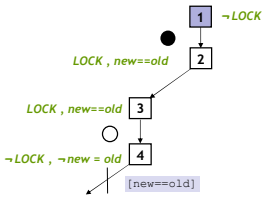
Reachability Tree

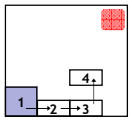
Predicates: *LOCK, new==old*

#38

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 ();
}
```





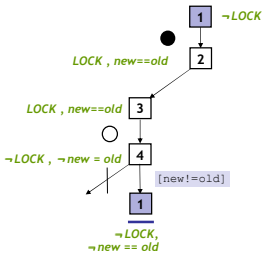
Reachability Tree

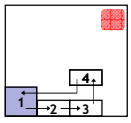
Predicates: *LOCK, new==old*

#39

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 ();
}
```





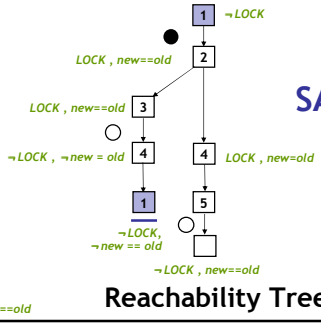
Reachability Tree

Predicates: *LOCK, new==old*

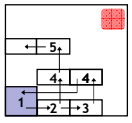
#40

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

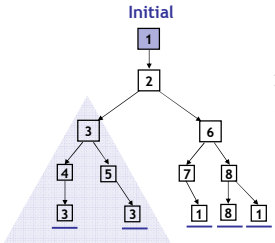


Reachability Tree

Predicates: *LOCK, new==old*

#41

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.

Error Free

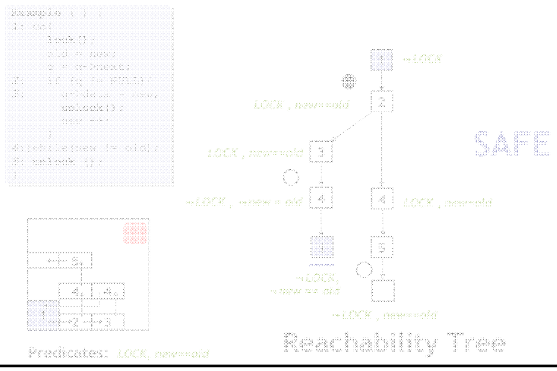
SAFE

S1: Only Abstract Reachable States

S2: Don't refine error-free regions

#42

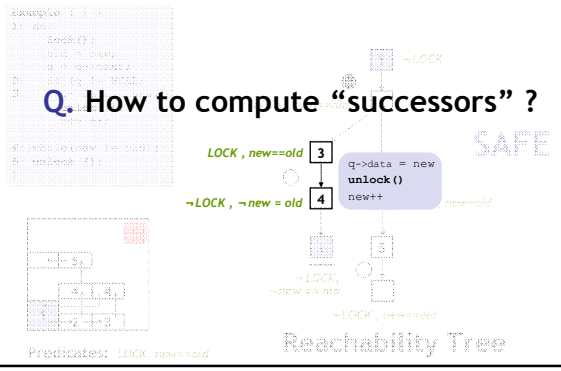
Two handwaves



#43

Two handwaves

Q. How to compute "successors" ?



#44

Two handwaves

Q. How to compute "successors" ?

Q. How to find predicates ?

Refinement

Predicates: *LOCK, new==old*

#45

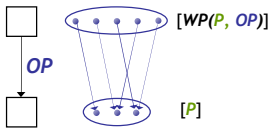
Two handwaves

Q. How to compute "successors" ?

#46

Weakest Preconditions

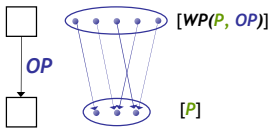
WP(P,OP)
Weakest formula *P'* s.t.
if *P'* is true before *OP*
then *P* is true after *OP*

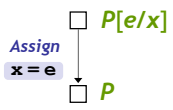


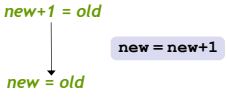
#47

Weakest Preconditions

WP(P,OP)
Weakest formula *P'* s.t.
if *P'* is true before *OP*
then *P* is true after *OP*



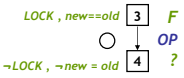




#48

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*

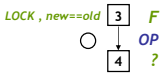
Q: When is *p* true after *OP* ?

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

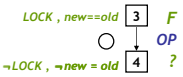
Q: When is *p* false after *OP* ?

- If *WP(¬p, OP)* is true before *OP* !
- We know *F* is true before *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 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*

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
   1) Use Weakest Preconditions and Theorem Prover Calls
4) Model-Check Resulting Boolean Program
   1) Use Symbolic Model Checking
5) Error State Not Reachable?
   1) Original Program Has No Errors: Done!
6) Check Counterexample Feasibility
   1) Use Symbolic Execution
7) Counterexample Is Feasible?
   1) 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**4**567
- [x=0, ¬x+2≠88, x+2<77]
- Preds = {x=0,x+1=88,x+2=88}
- Path = 2345**4**5**4**567
- …
- Result: the predicates "count" the loop iterations

Homework

- Project Status Update
- Project Due Tue Apr 25
  - You have ~14 days to complete it.
  - Need help? Stop by my office or send email.

#55