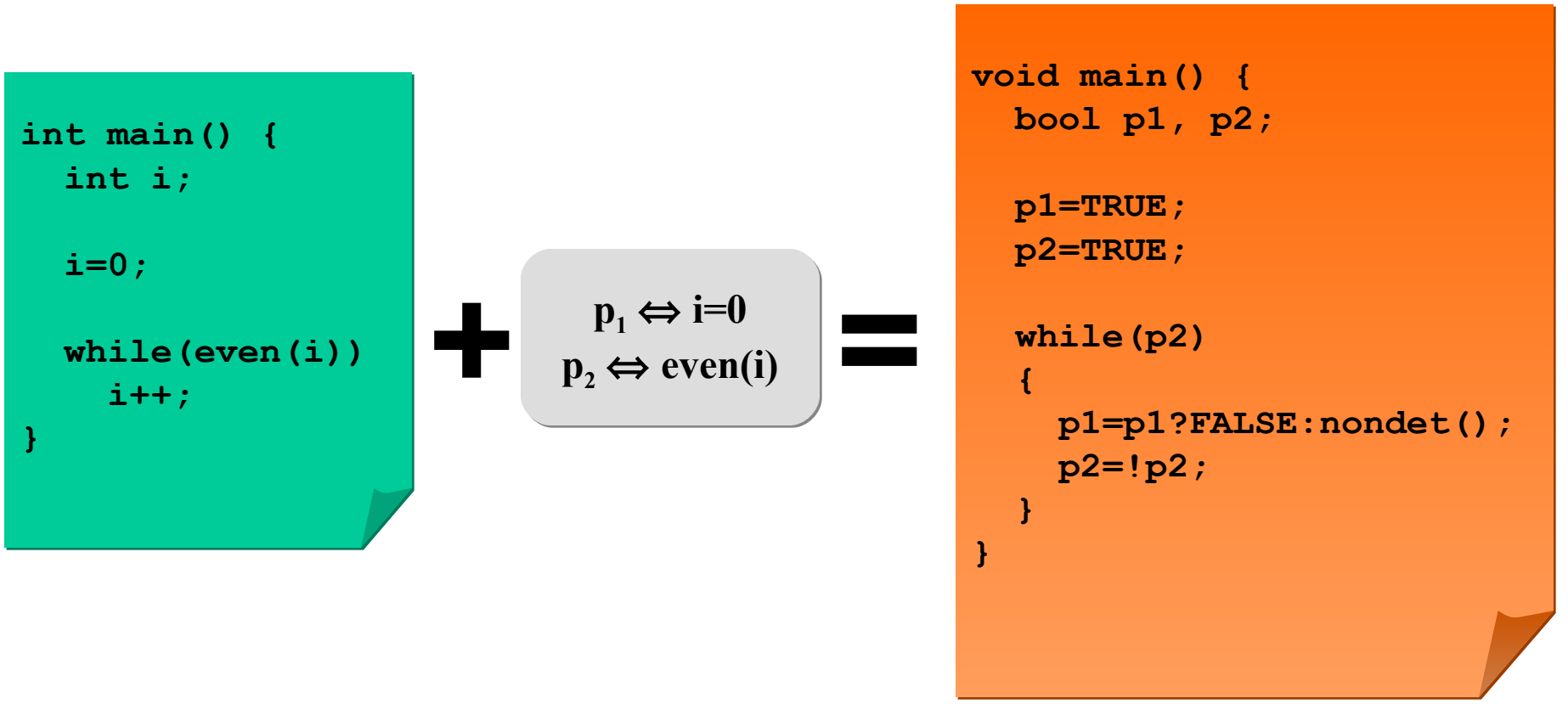
JPF2: Predicate Abstraction

CS 510/10

Predicate Abstraction

- Extract a finite state model from an infinite state system
- Used to prove assertions or safety properties
- Successfully applied for verification of C programs
  - SLAM (used in windows device driver verification)
  - MAGIC, BLAST, F-Soft

Example for Predicate Abstraction



C program | Predicates | Boolean program

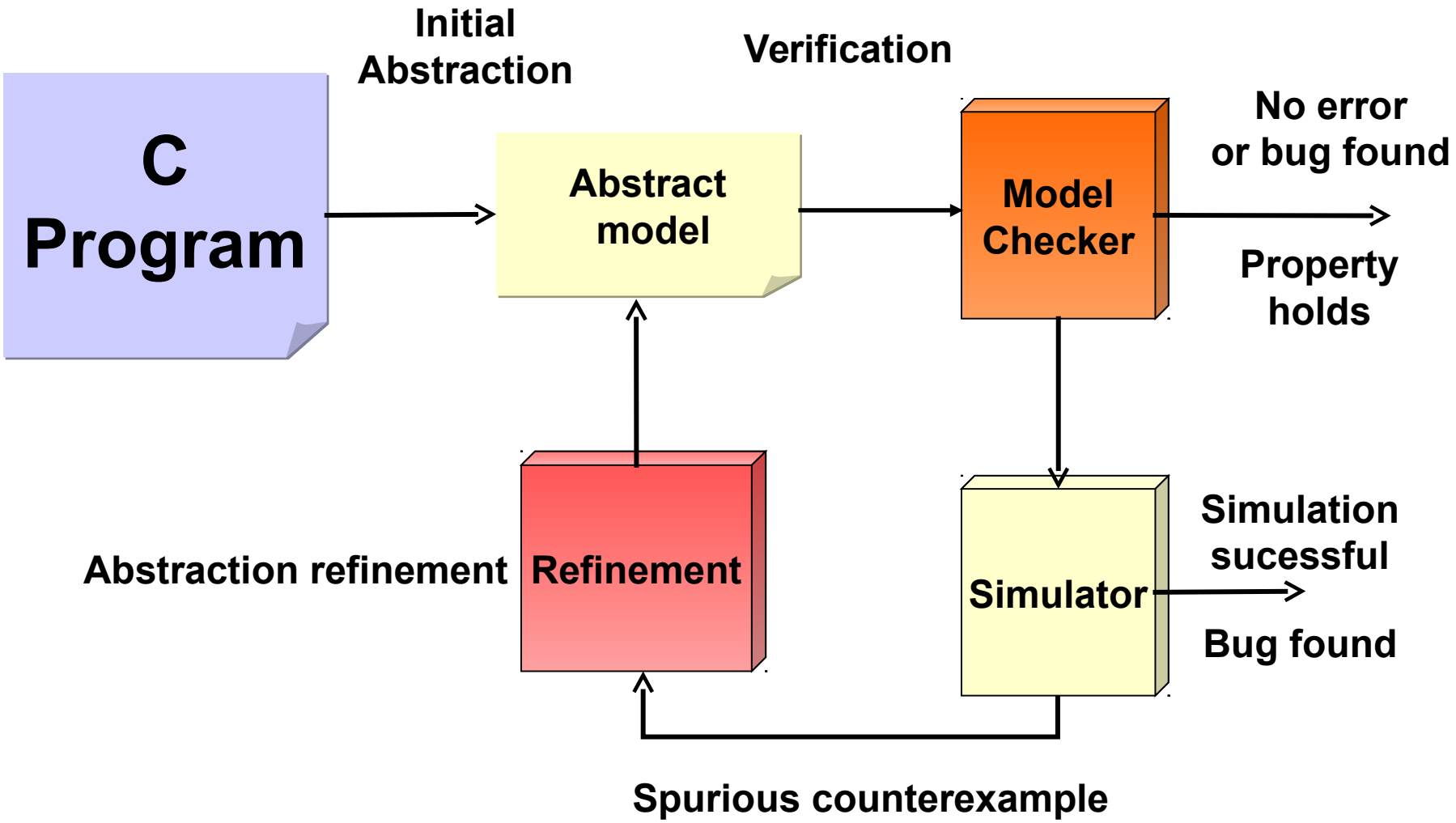
[Graf, Saidi '97]

[Ball, Rajamani '01]

Computing Predicate Abstraction

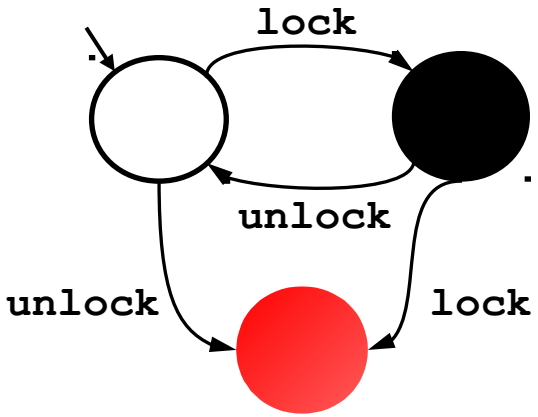
- How to get predicates for checking a given property?
- How do we compute the abstraction?
- Predicate Abstraction is an over-approximation
  - How to refine coarse abstractions?

Counterexample Guided Abstraction Refinement loop

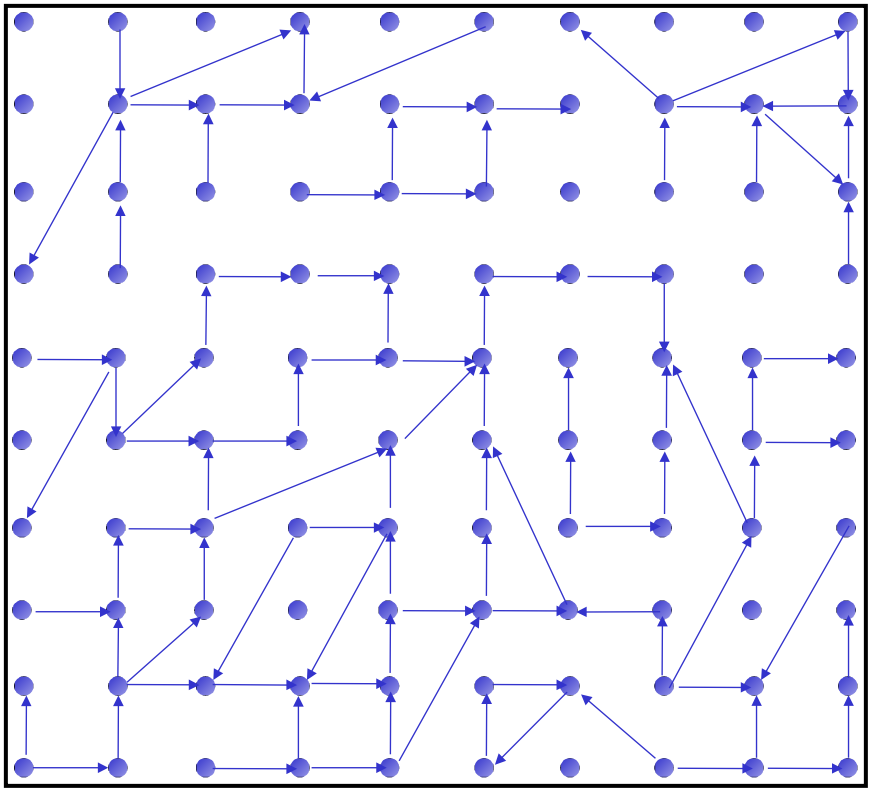


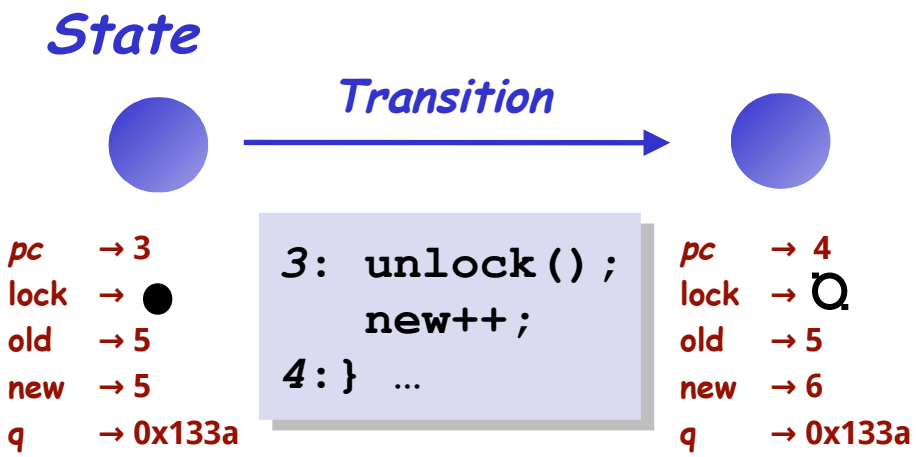
Example

```
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;
}
```



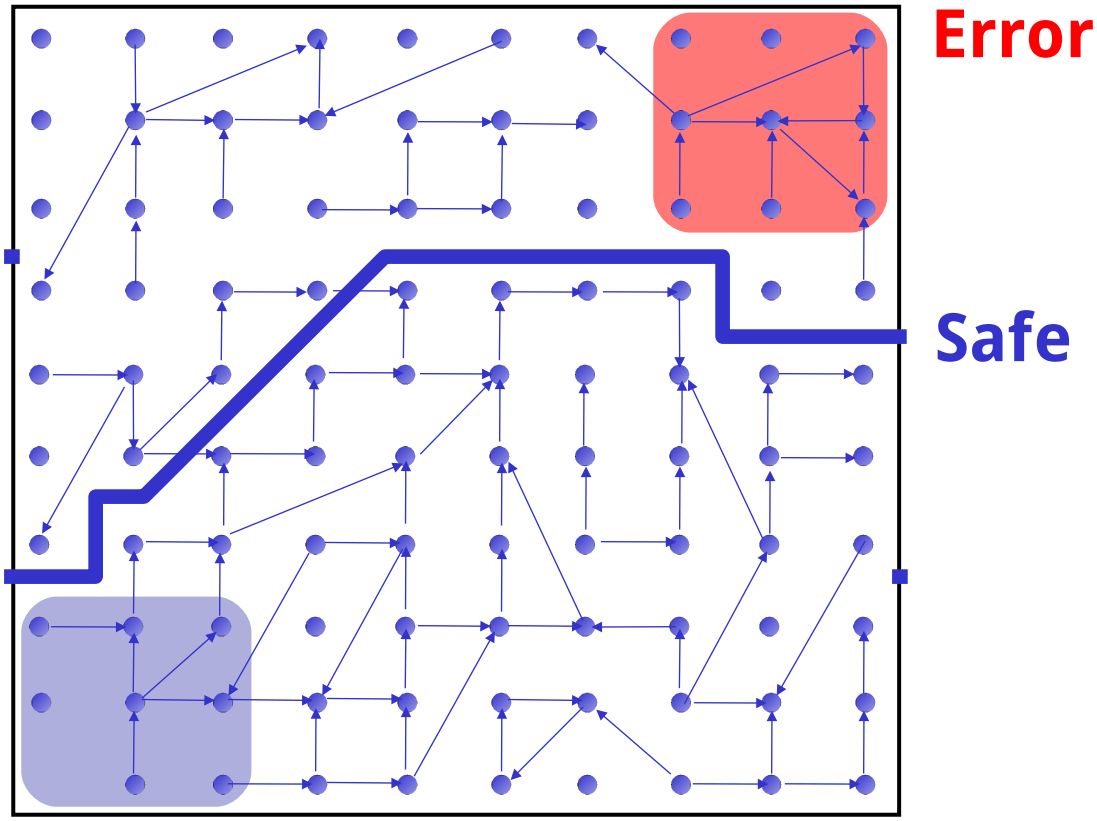
What a program *really* is…






```
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

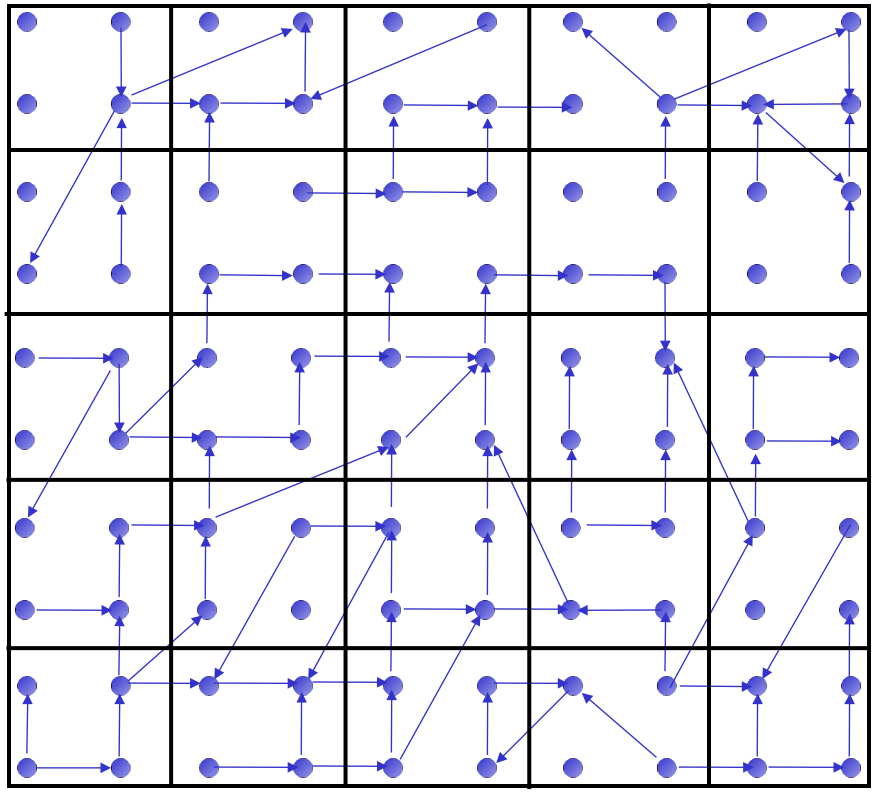


Is there a path from an initial to an error state ?

Problem: Infinite state graph

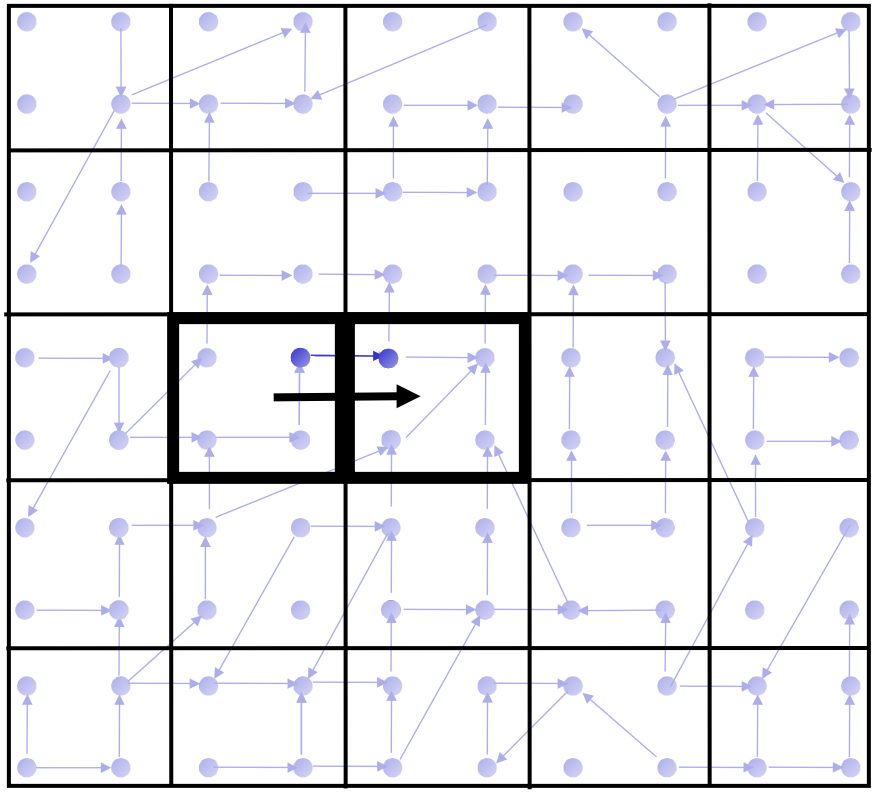
Solution : Set of states ' logical formula

Idea 1: Predicate Abstraction

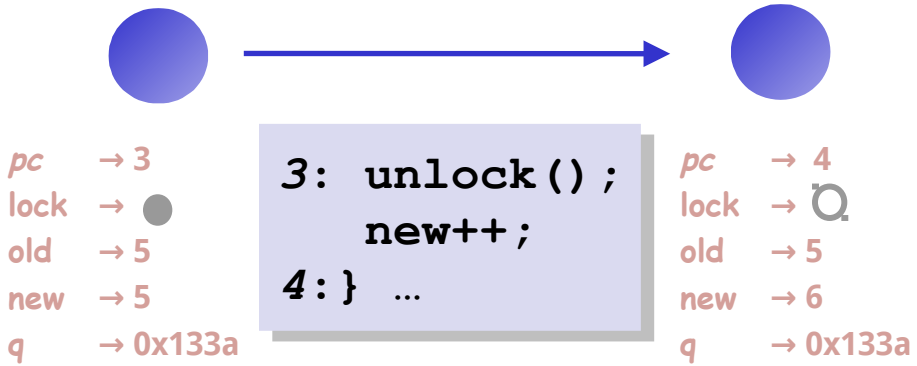


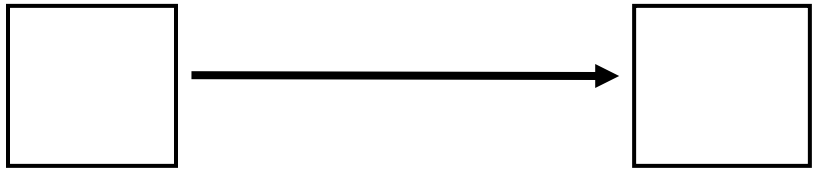
- **Predicates on program state:**
  lock
  old = new
- **States satisfying same predicates are equivalent**
  - **Merged into one abstract state**
- **#abstract states is finite**

Abstract States and Transitions



State

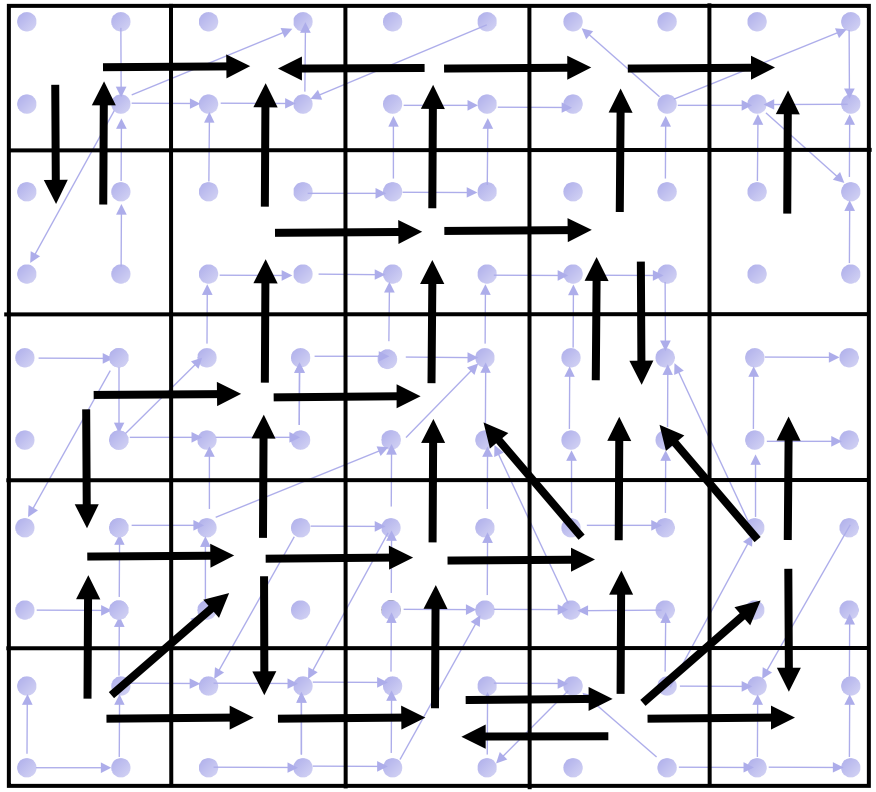




lock
old=new

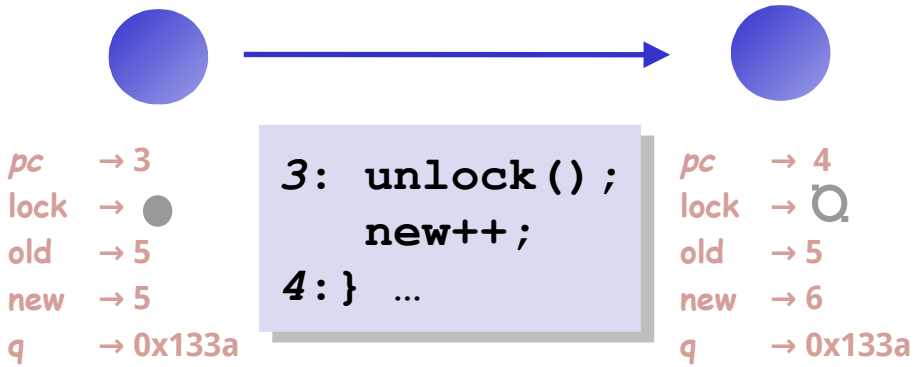
! lock
! old=new

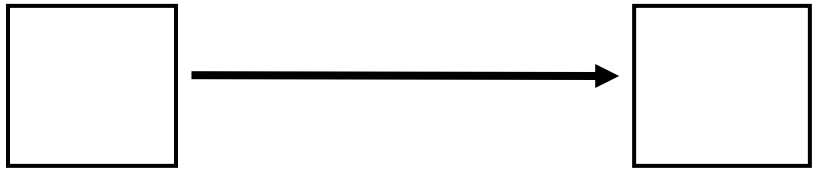
Abstraction



Existential Approximation

State

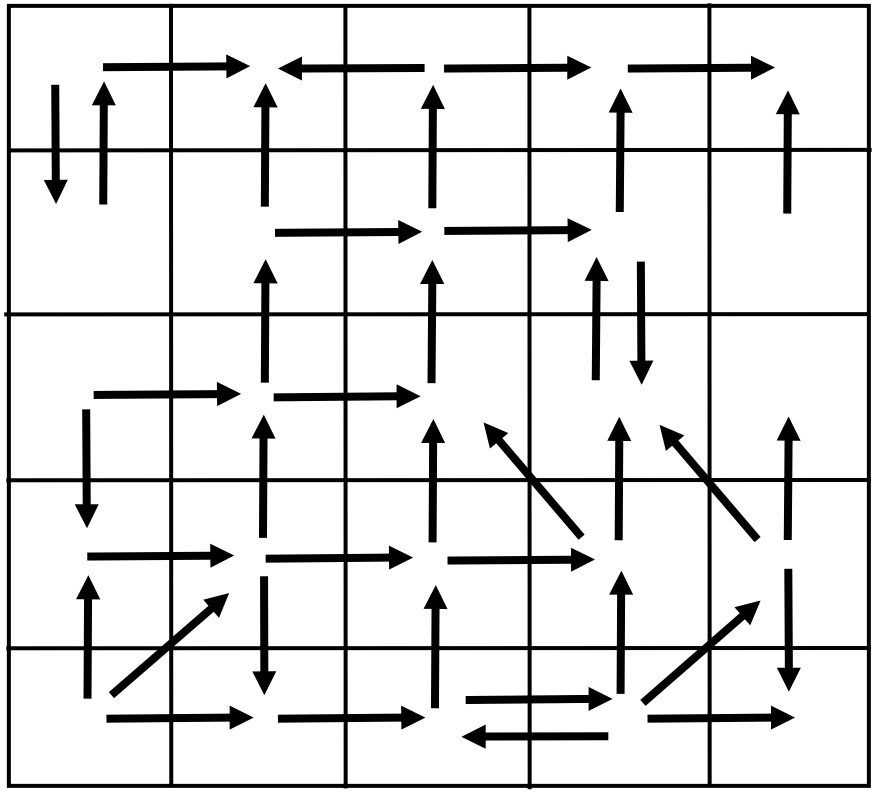


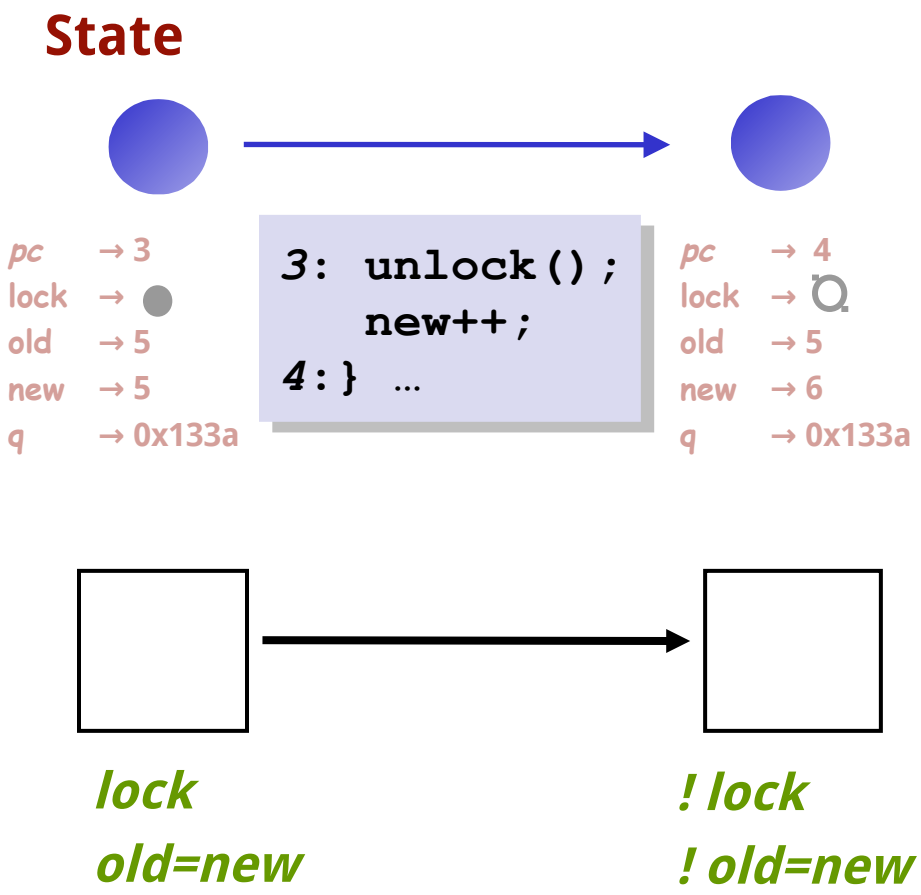


lock
old=new

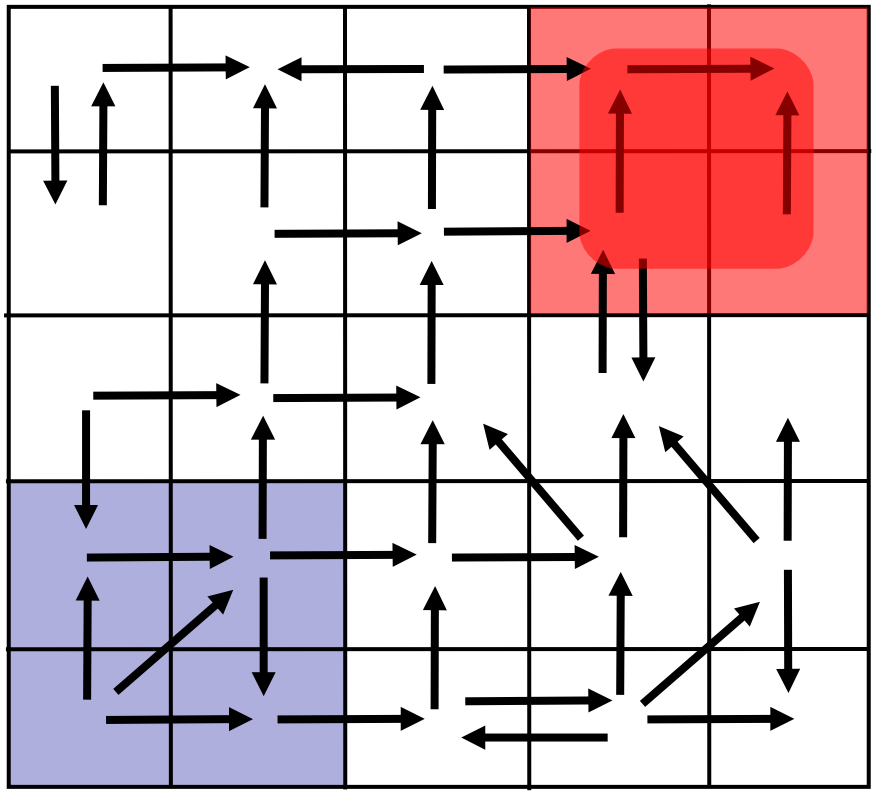
! lock
! old=new

Abstraction





Analyze Abstraction



Analyze finite graph

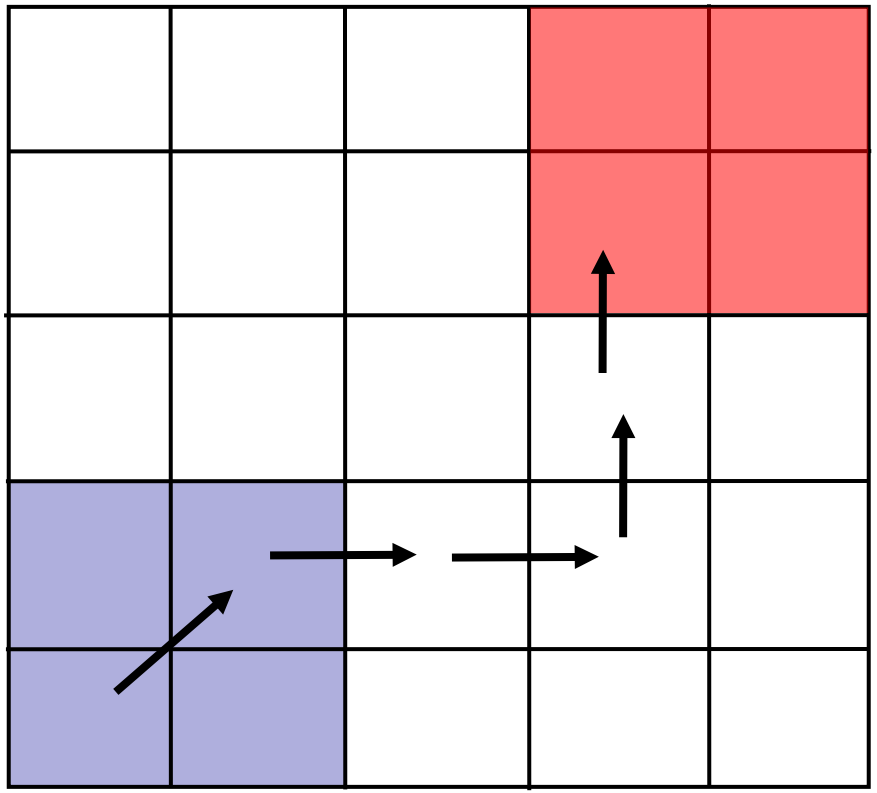
Over Approximate:

Safe => System Safe

Problem

Spurious counterexamples

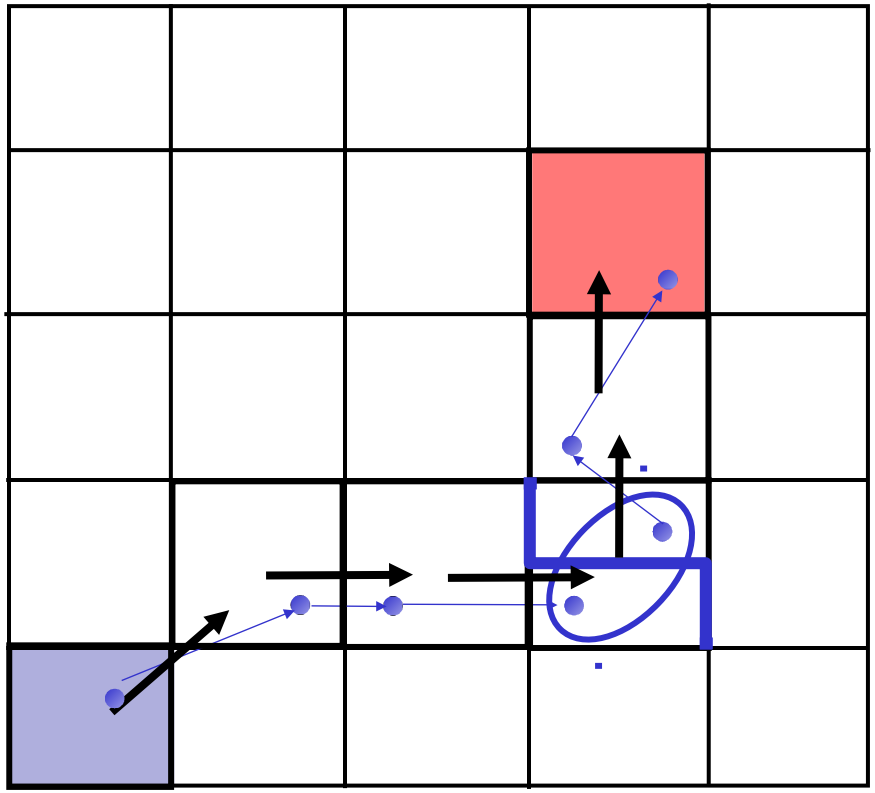
Idea 2: Counterex.-Guided Refinement



Solution

Use spurious counterexamples to refine abstraction !

Idea 2: Counterex.-Guided Refinement

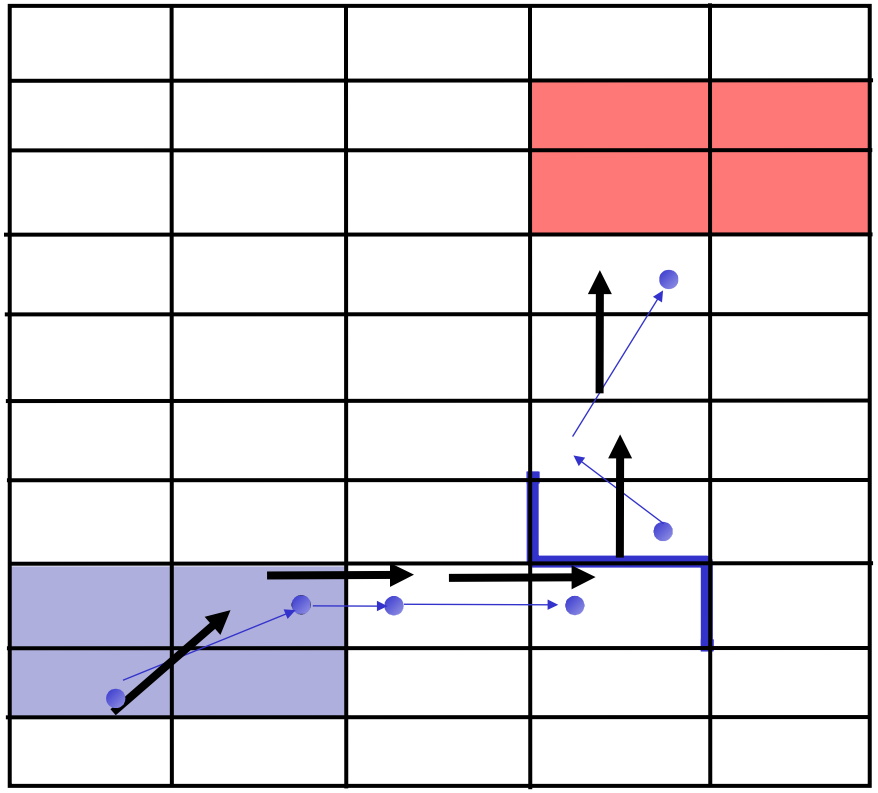


Solution

Use spurious counterexamples to refine abstraction

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

Iterative Abstraction-Refinement



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

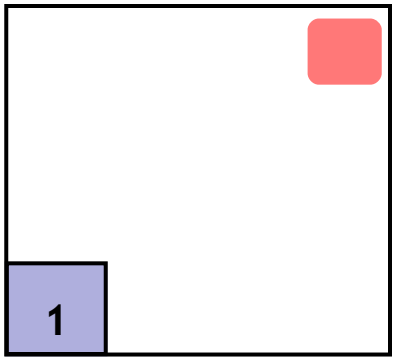
Solution

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

Build-and-Search

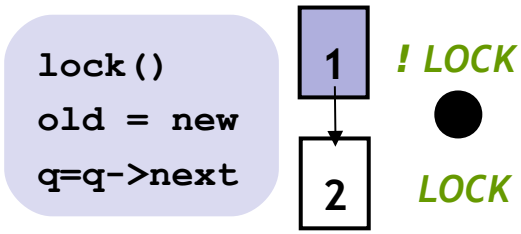
```
Example ( ) {
1: do{
     lock();
     old = new;
     q = q->next;
2:   if (q != NULL){
3:     q->data = new;
       unlock();
       new ++;
     }
4:}while(new != old);
5: unlock ();
}
```

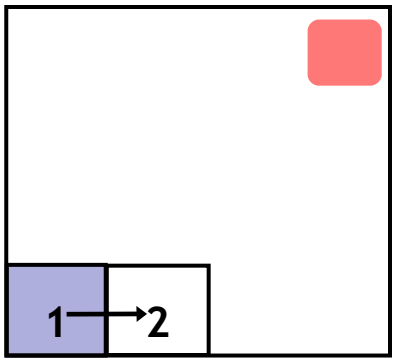


Predicates: *LOCK*

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 ();
}
```

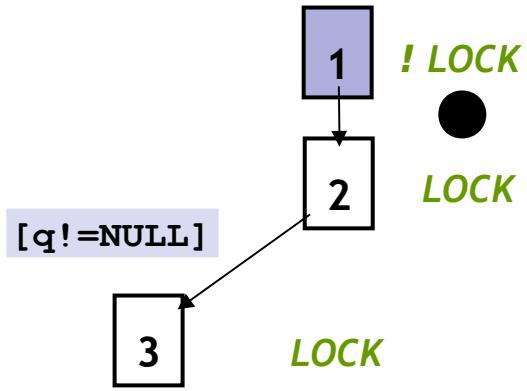


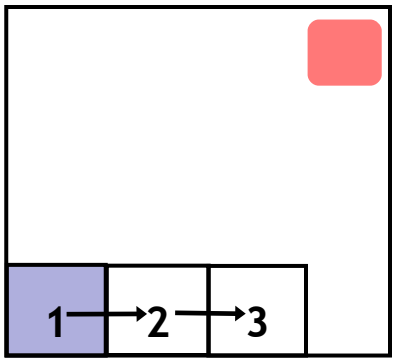


Predicates: *LOCK*

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 ();
}
```

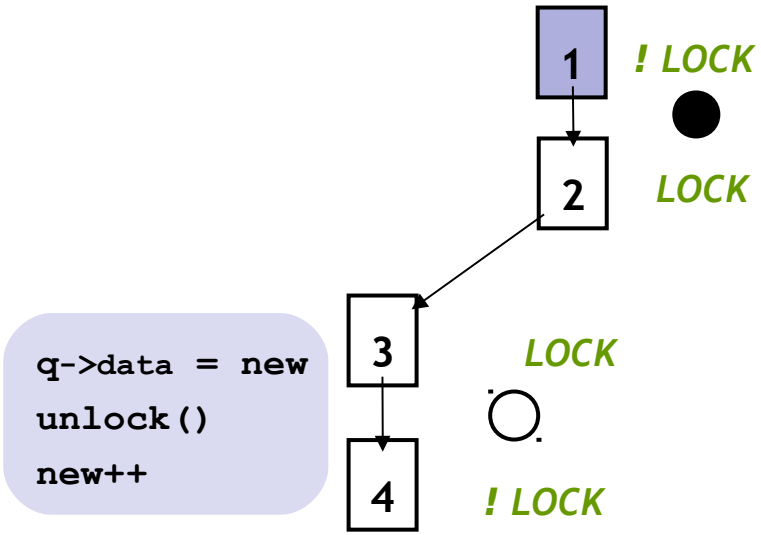


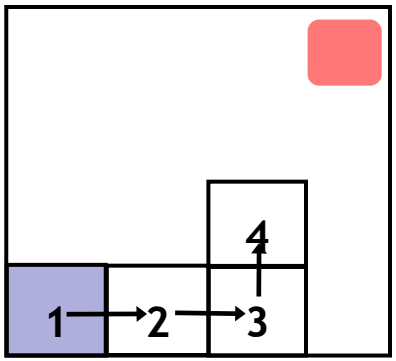


Predicates: *LOCK*

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 ();
}
```

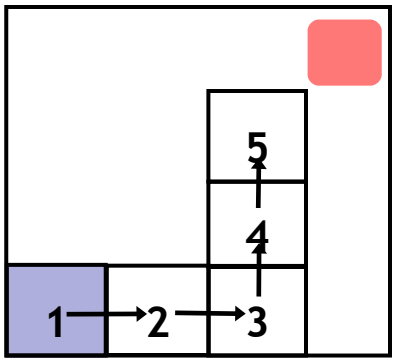


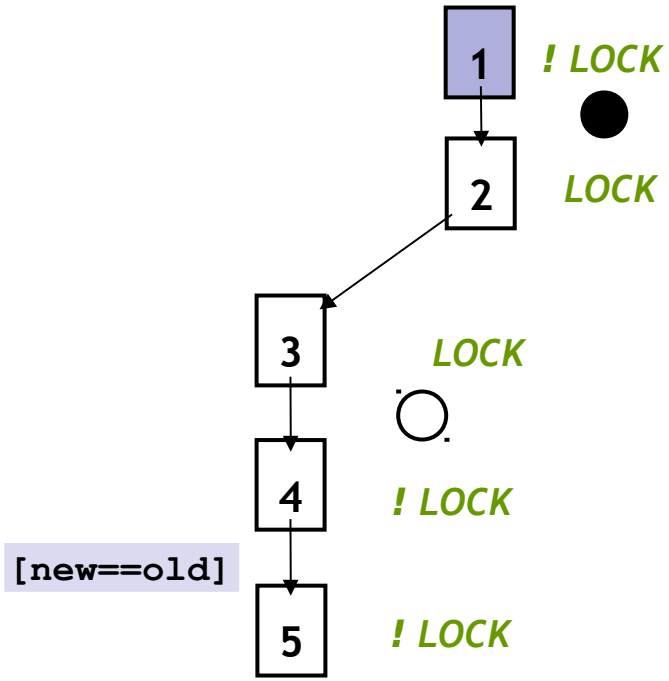


Predicates: *LOCK*

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 ();
}
```

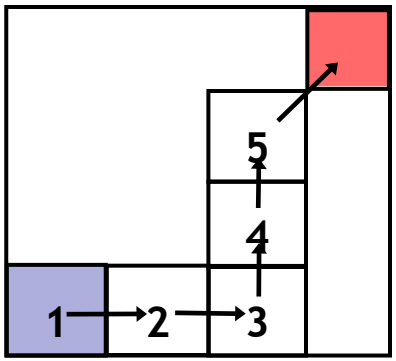


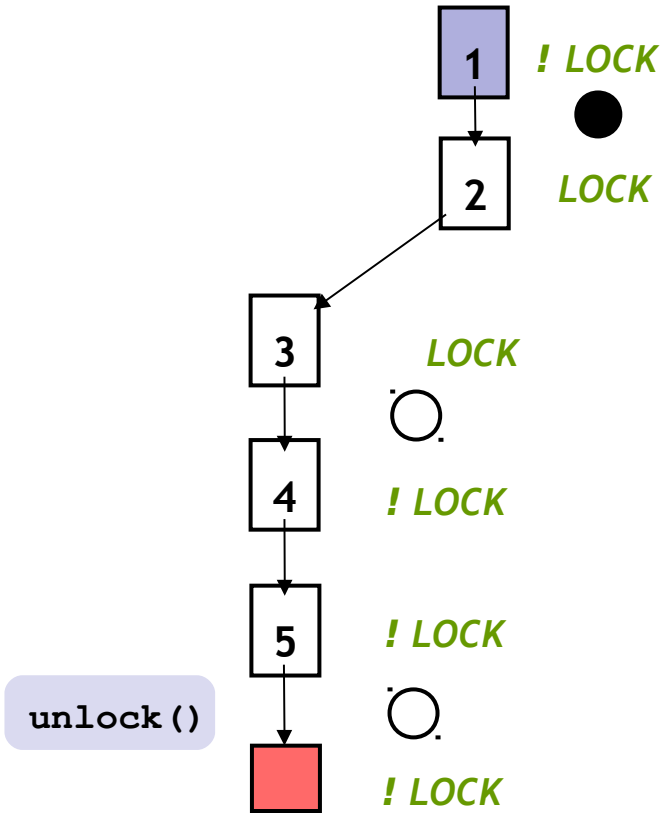


Predicates: *LOCK*

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 ();
}
```

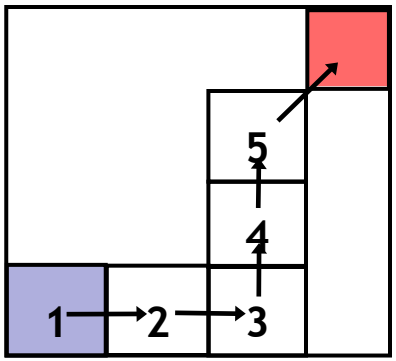


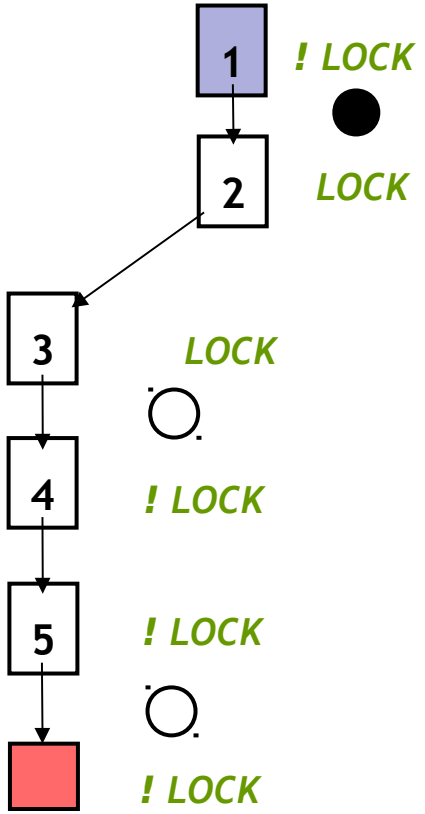


Predicates: *LOCK*

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++

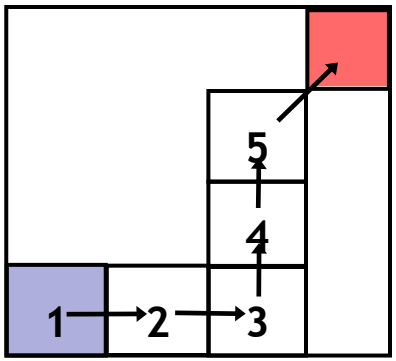
[new==old]

unlock()

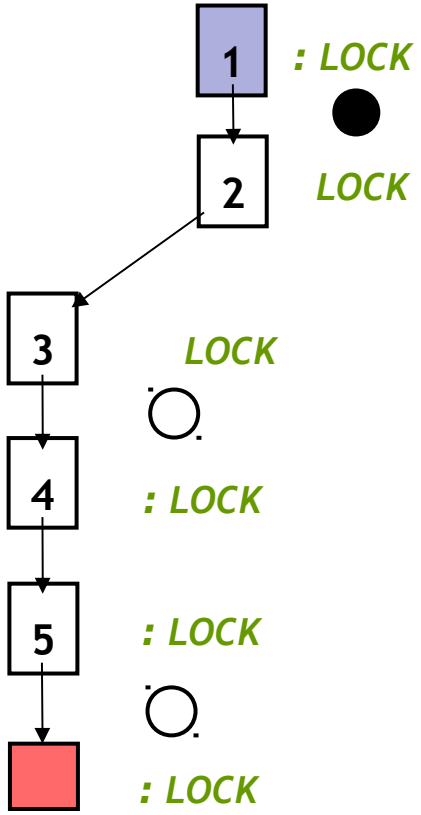
Predicates: *LOCK*

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++`

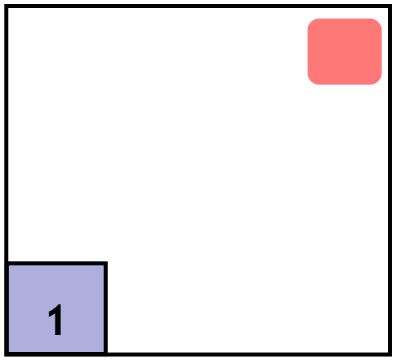
`[new==old]`

Inconsistent

new == old

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 ();
}
```

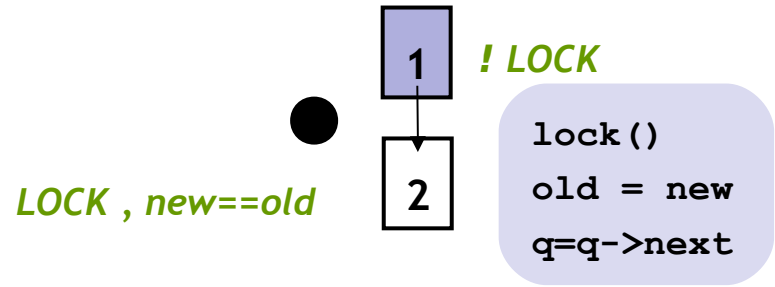


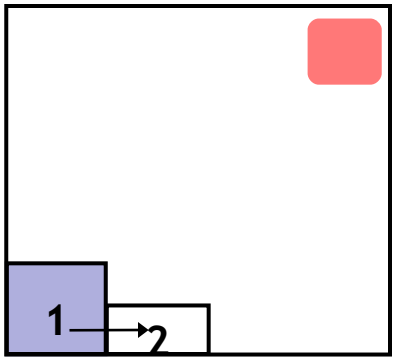
1 : LOCK

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 ();
}
```

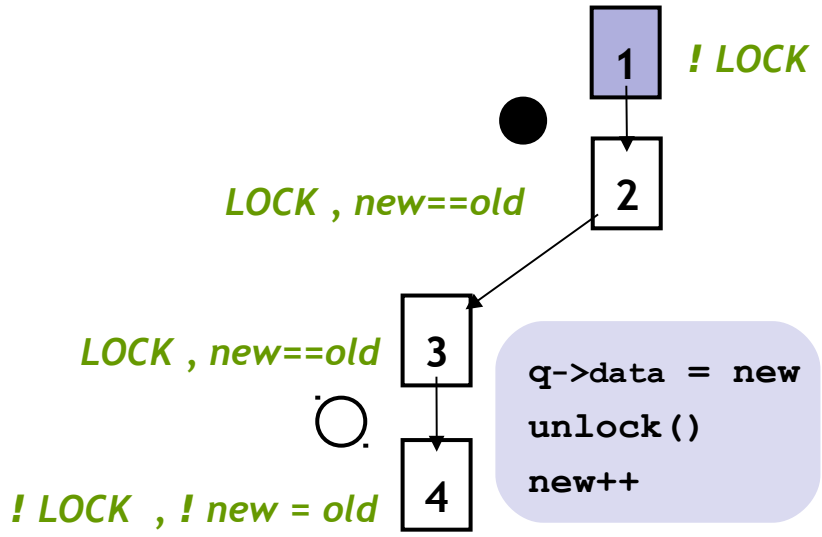


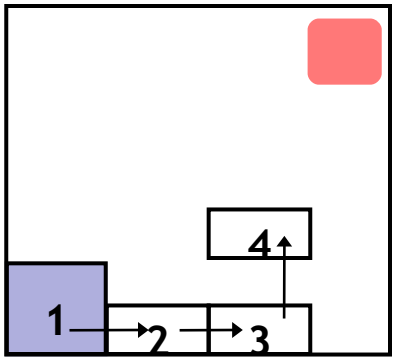


Predicates: *LOCK, new==old*

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 ();
}
```

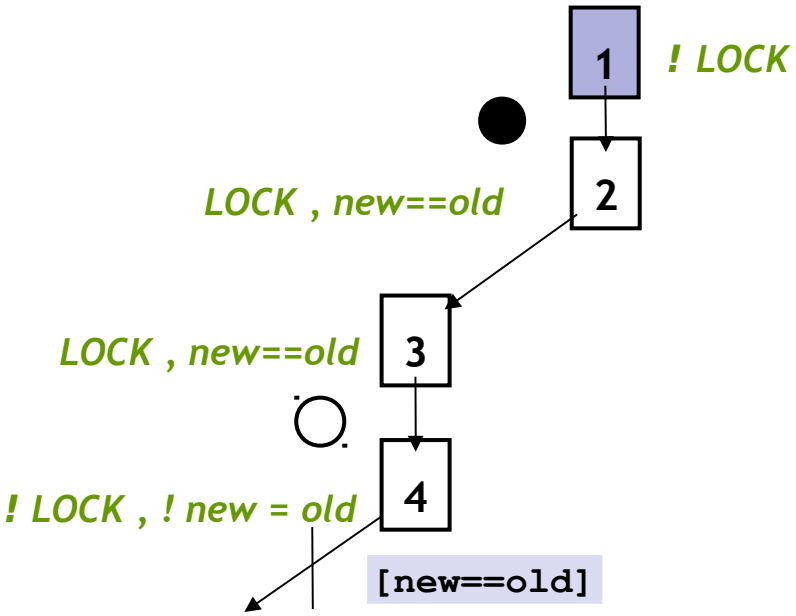


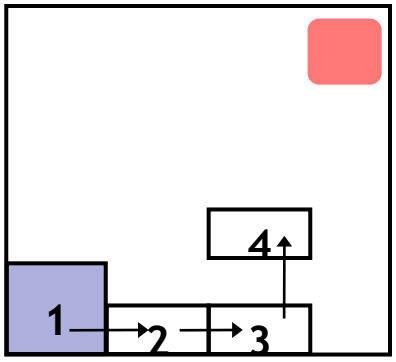


Predicates: *LOCK, new==old*

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 ();
}
```

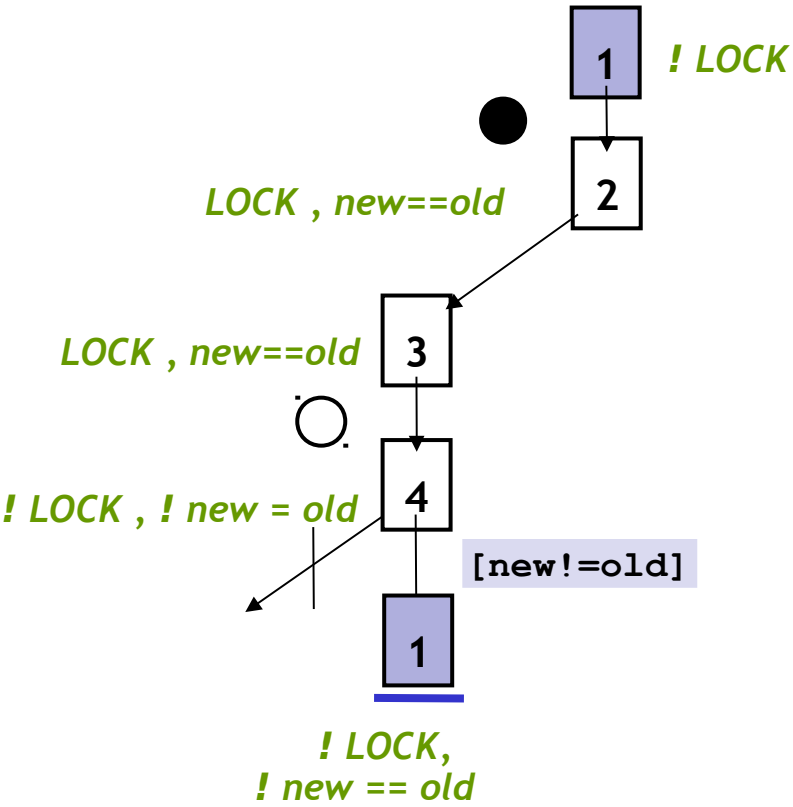


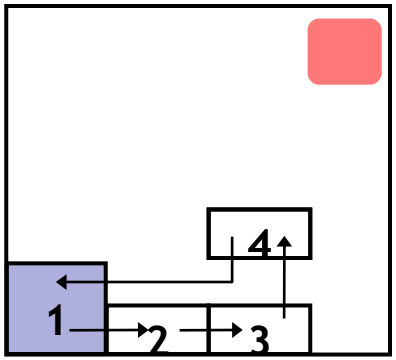


Predicates: *LOCK, new==old*

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 ();
}
```

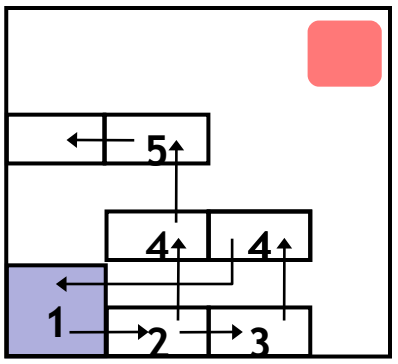


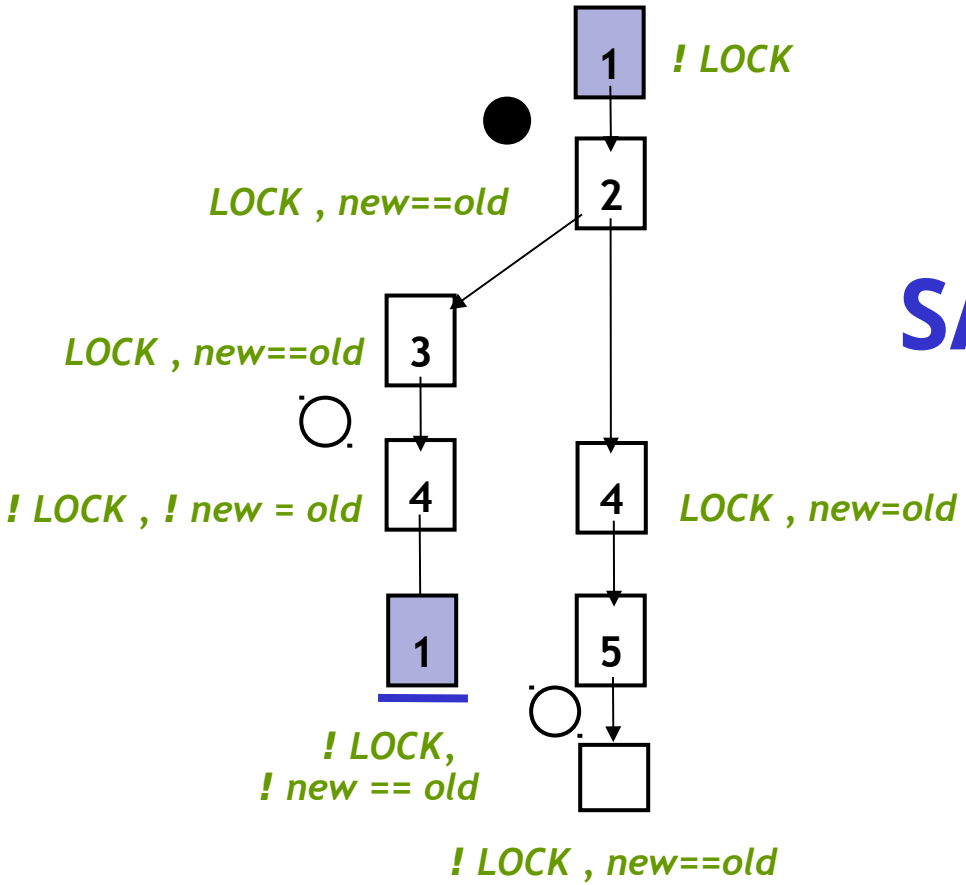


Predicates: *LOCK, new==old*

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

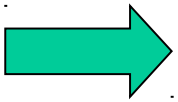
Predicates: *LOCK, new==old*

Another Example

```
1: x = ctr;
2: y = ctr + 1;
3: if (x == i-1){
4:    if (y != i){
          ERROR:
      }
   }
```

C program

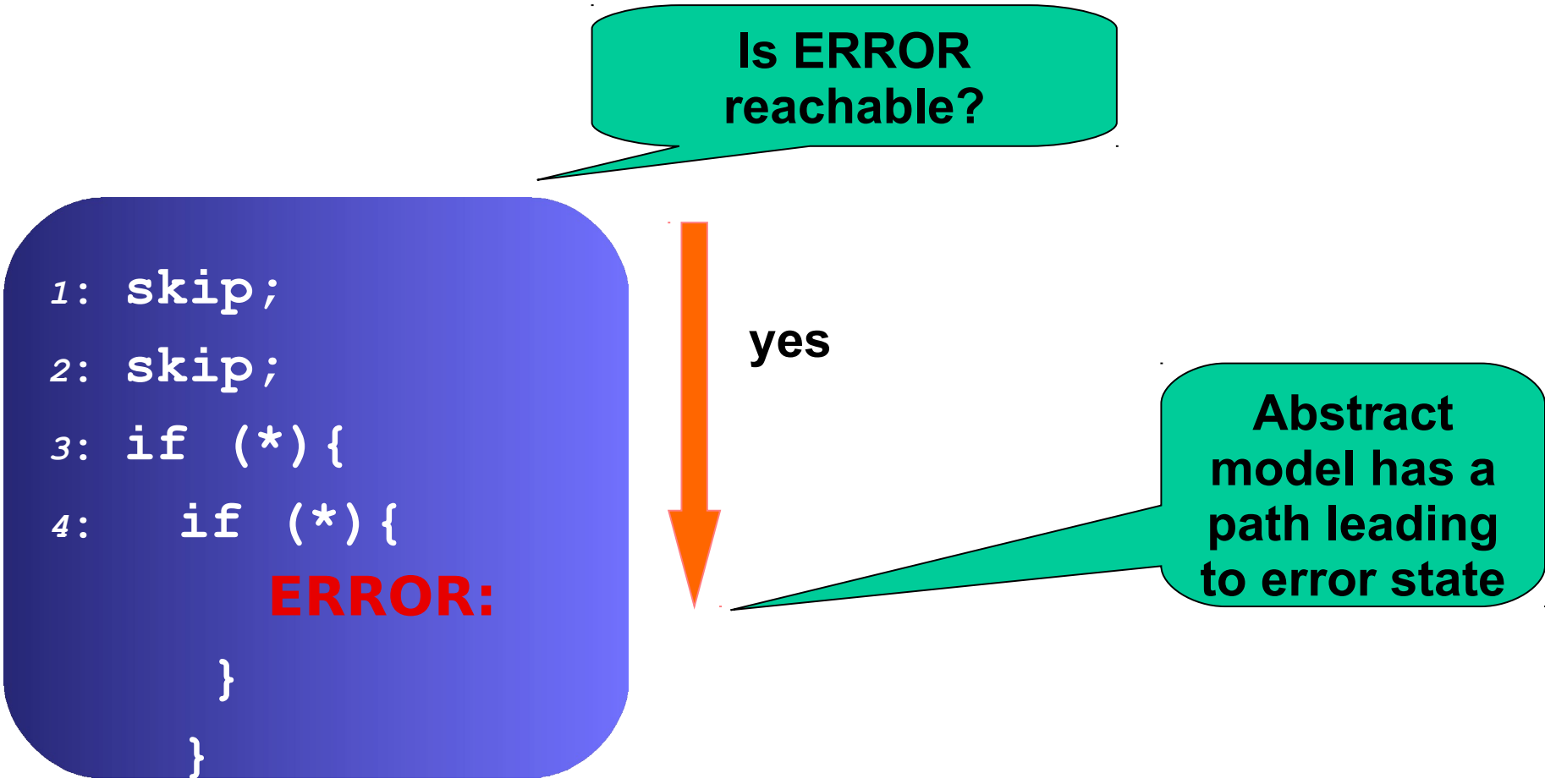
Abstract

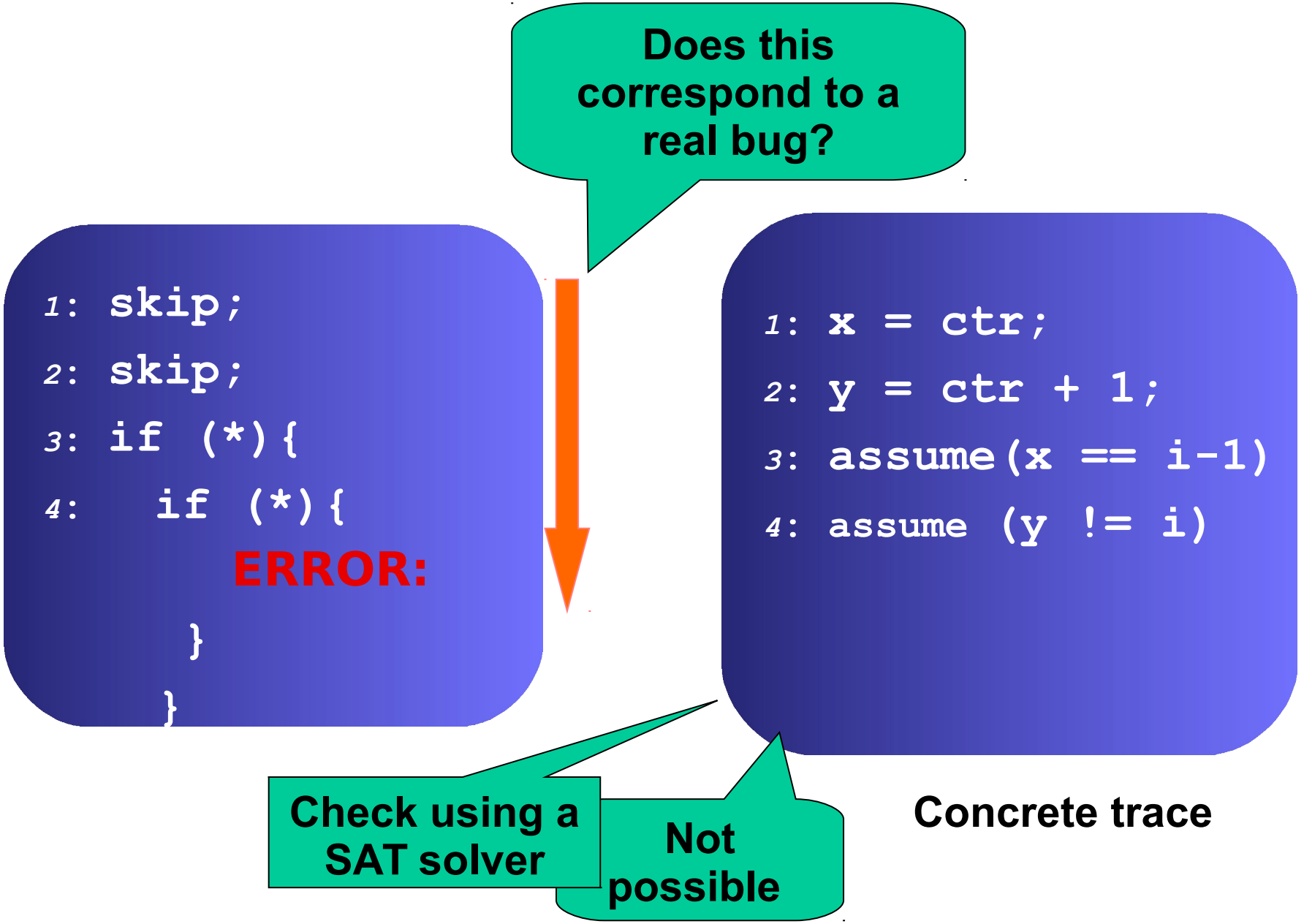


```
1: skip;
2: skip;
3: if (*){
4:    if (*){
          ERROR:
      }
   }
```

No predicates available currently

Checking the abstract model




Does this correspond to a real bug?
1: skip;
2: skip;
3: if (*){
4: if (*){
ERROR:
}
}
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
Check using a SAT solver
Not possible
Concrete trace

Refinement

```
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
```

Spurious Counterexample

```
1: skip;
2: skip;
3: if (*){
4:    if (*){
         ERROR:
      }
   }
```

Initial abstraction

Refinement

```
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
```

```
1: skip;
2: skip;
3: if (*){
4:   if (b0){
       ERROR:
     }
   }
```

boolean b0 : y != i

Refinement

```
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
```

```
1: skip;
2: skip;
3: if (b1){
4:    if (b0){
          ERROR:
      }
   }
```

boolean b0 : y != i

boolean b1 : x== i-1

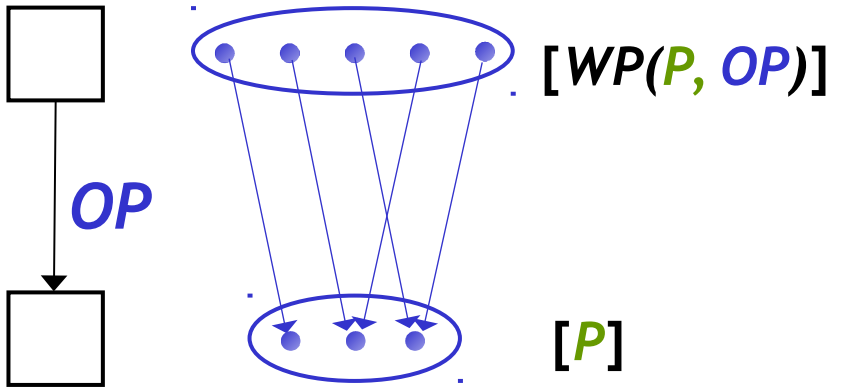
Weakest Preconditions

WP(P,OP)

Weakest formula *P'* s.t.

if *P'* is true before *OP*

then *P* is true after *OP*



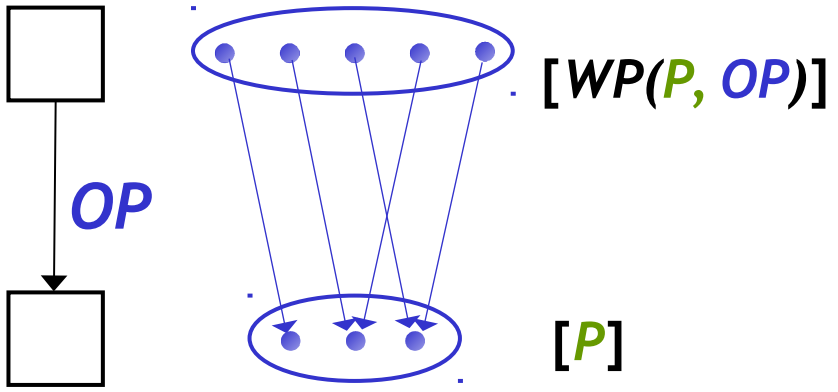
Weakest Preconditions

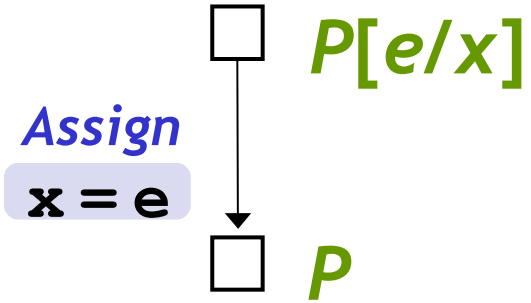
WP(P,OP)

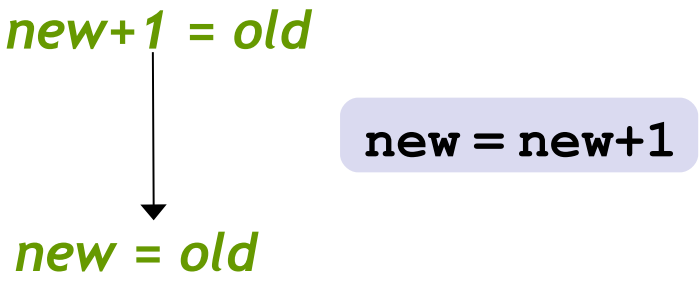
Weakest formula *P'* s.t.

if *P'* is true before *OP*

then *P* is true after *OP*







Refinement

Weakest precondition of y != i

ctr + 1 ! = i

```
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
```

```
1: skip;
2: b0 = b2;
3: if (b1){
4:   if (b0){
       ERROR:
     }
   }
```

boolean b0 : y != i

boolean b1 : x== i-1

Refinement

boolean b2 : ctr + 1 ! = i

boolean b3: ctr == i -1

```
1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)
```

```
1: b1 = b3;
2: b0 = b2;
3: if (b1){
4:    if (b0){
          ERROR:
      }
   }
```

boolean b0 : y != i

boolean b1 : x== i-1

Refinement

boolean b2 : ctr + 1 ! = i

boolean b3: ctr == i -1

What about initial values of b2 and b3?

are mutually exclusive.

b2 =1, b3 = 0

b2 =0 , b3 = 1

So system is safe!

```
1: b1 = b3;
2: b0 = b2;
3: if (b1){
4:    if (b0){
          ERROR:
      }
   }
```

boolean b0 : y != i

boolean b1 : x== i-1

Tools for Predicate Abstraction of C

- **SLAM at Microsoft**
  - Used for verifying correct sequencing of function calls in windows device drivers
- **MAGIC at CMU**
  - Allows verification of concurrent C programs
  - Found bugs in MicroC OS
- **BLAST at Berkeley**
  - Lazy abstraction, interpolation
- **SATABS at CMU**
  - Computes predicate abstraction using SAT
  - Can handle pointer arithmetic, bit-vectors
- **F-Soft at NEC Labs**
  - Localization, register sharing