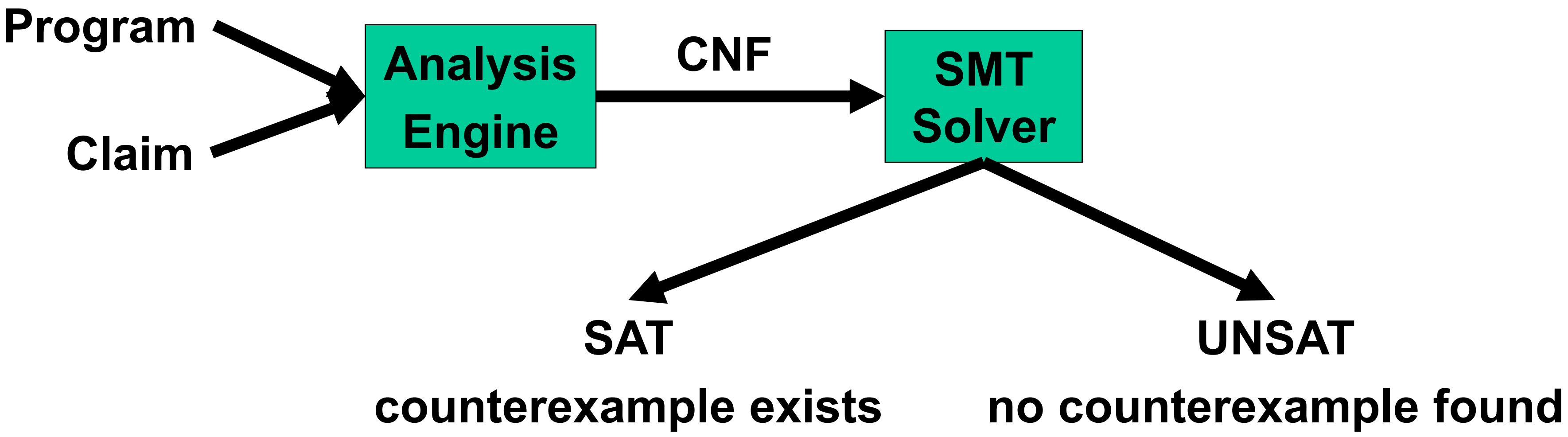
Software Model Checking

Xiangyu Zhang

Symbolic Software Model Checking

- Symbolic analysis explicitly explores individual paths, encodes and resolves path conditions
- Model checking directly encodes both the program and the property to check to constraints



Program

Claim

Analysis Engine

CNF

SMT Solver

SAT

counterexample exists

UNSAT

no counterexample found

A (very) simple example (1)

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 7 ||
        w == 9)
```

Constraints

y = 8,
z = x ? y – 1 : 0,
w = x ? 0 :y + 1,
z != 7,
w != 9

UNSAT

no counterexample

assertion always holds!

A (very) simple example (2)

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y – 1;
else
  w = y + 1;
assert (z == 5 ||
        w == 9)
```

Constraints

y = 8,
z = x ? y – 1 : 0,
w = x ? 0 :y + 1,
z != 5,
w != 9

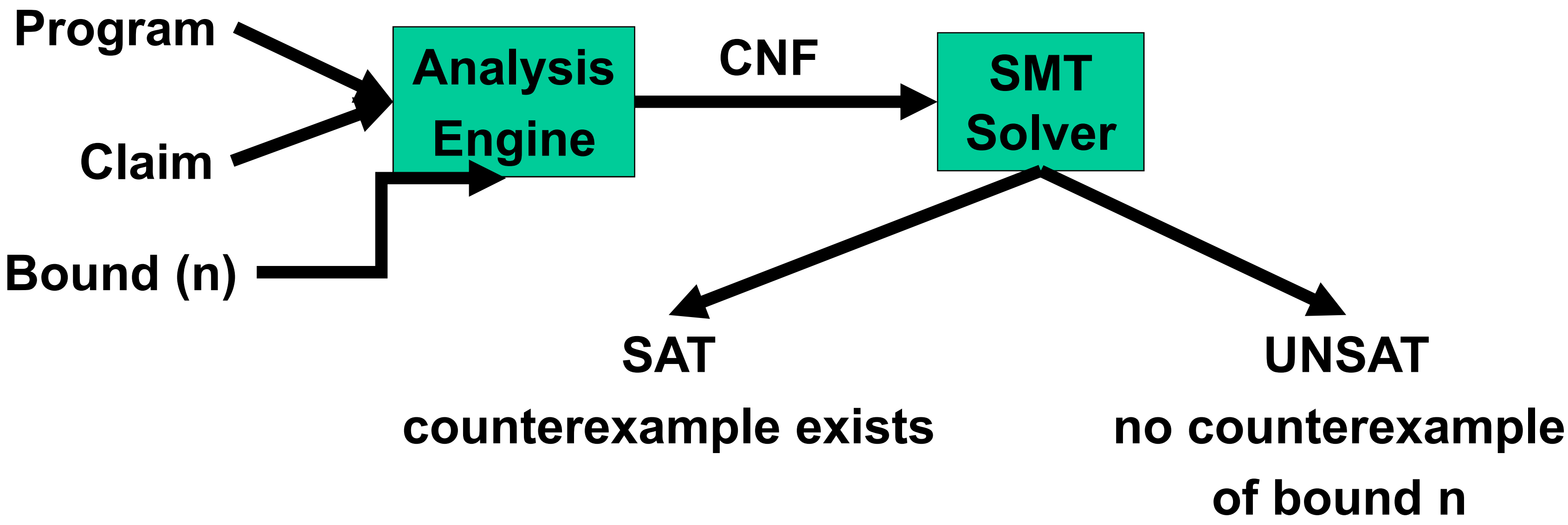
SAT

counterexample found!

y = 8, x = 1, w = 0, z = 7

Procedure

- Unroll loops



- Translate to SSA form
- SSA to SMT constraints

Loop Unwinding

- **All loops are unwound**
    - **can use different unwinding bounds for different loops**
    - **to check whether unwinding is sufficient special "unwinding assertion" claims are added**
- **If a program satisfies all of its claims and all unwinding assertions then it is correct!**
- **Same for backward `goto` jumps and recursive functions**

Loop Unwinding

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      while(cond) {
        Body;
      }
    }
  }
  Remainder;
}
```

- while() loops are unwound iteratively
- Break / continue replaced by goto

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        while(cond) {
          Body;
        }
      }
    }
  }
  Remainder;
}
```

- **while() loops are unwound iteratively**
- **Break / continue replaced by goto**
- **Assertion inserted after last iteration: violated if program runs longer than bound permits**

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assert(!cond);

      }
    }
  }
  Remainder;
}
```

Unwinding assertion

- while() loops are unwound iteratively
- Break / continue replaced by goto
- Assertion inserted after last iteration: violated if program runs longer than bound permits

Example: Sufficient Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 2)
    j = j + 1;
  Remainder;
}
```

–unwind = 3

```
void f(...) {
  j = 1
  if(j <= 2) {
    j = j + 1;
    if(j <= 2) {
      j = j + 1;
      if(j <= 2) {
        j = j + 1;
        assert(!(j <= 2));
        }
      }
    }
  }
  Remainder;
}
```

Example: Insufficient Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 10)
    j = j + 1;
  Remainder;
}
```

```
void f(...) {
  j = 1
  if(j <= 10) {
    j = j + 1;
    if(j <= 10) {
      j = j + 1;
      if(j <= 10) {
        j = j + 1;
        assert(!(j <= 10));
        }
      }
    }
  }
  Remainder;
}
```

–unwind = 3

Transforming Loop-Free Programs Into Equations (1)

Easy to transform when every variable is only assigned once!

Program

```
x = a;
y = x + 1;
z = y - 1;
```

Constraints

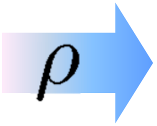
```
x = a &&
y = x + 1 &&
z = y – 1 &&
```

Transforming Loop-Free Programs Into Equations (2)

When a variable is assigned multiple times,
use a new variable for the RHS of each assignment

Program

```
x=x+y;
x=x*2;
a[i]=100;
```



SSA Program

$x_1=x_0+y_0;$

$x_2=x_1*2;$

$a_1[i_0]=100;$

What about conditionals?

Program

```
if (v)
  x = y;
else
  x = z;

w = x;
```

ρ

SSA Program

```
if (v_0)
  x_0 = y_0;
else
  x_1 = z_0;

w_1 = x??;
```

What should ‘x’ be?

What about conditionals?

Program

```
if (v)
   x = y;
else
   x = z;

w = x;
```

$$\xrightarrow{\rho}$$

SSA Program

```
if ($v_0$)
   $x_0$ = $y_0$;
else
   $x_1$ = $z_0$;
$x_2$ = $v_0$ ? $x_0$ : $x_1$;
$w_1$ = $x_2$
```

For each join point, add new variables with selectors

Encoding

- Declare symbolic variables for each (SSA) scalar variables
- Assignments to equivalence
- Phi functions to ITE expressions
- Array accesses to select/store operations
- Scalar pointer dereferences to identify operations
- Heap dereferences to select/store operations

```
if (v)
  p = &x;
else
  p = &y;

*p = 10;
q=p;
z=*q
```

```
p =(int*) malloc(100);
i = 10;
q = p+i
*q = 10
```

CBMC: C Bounded Model Checker

- Developed at CMU by Daniel Kroening et al.
- Available at: http://www.cs.cmu.edu/~modelcheck/cbmc/
- Supported platfoms: Windows (requires VisualStudio's CL), Linux
- Provides a command line and Eclipse-based interfaces

- Known to scale to programs with over 30K LOC
- Was used to find previously unknown bugs in MS Windows device drivers

Explicit State Model Checking

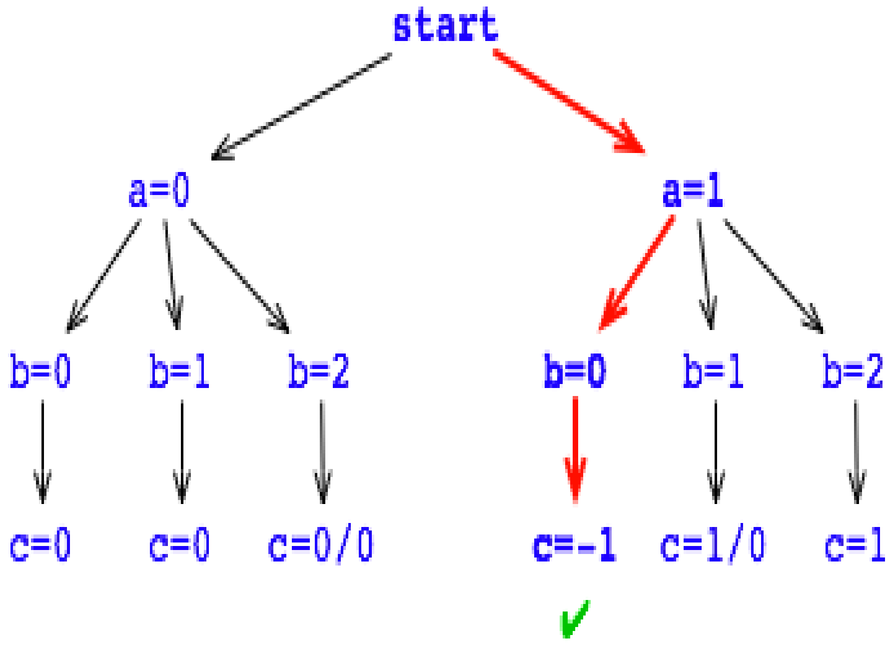
- The program is indeed executing
  - jpf <your class> <parameters>
    - Very similar to "java <your class> <parameters>
  - Execute in a way that all possible scenarios are explored
    - Thread interleaving
    - Undeterministic values (random values)
  - Concrete input is provided
  - A state is indeed a concrete state, consisting of
    - Concrete values in heap/stack memory

An Example

```
import java.util.Random;

public class Rand {
     public static void main (String[] args) {
          Random random = new Random(42);      // (1)

          int a = random.nextInt(2);           // (2)
          System.out.println("a=" + a);

          //... lots of code here

          int b = random.nextInt(3);           // (3)
          System.out.println("  b=" + b);

          int c = a/(b+a -2);                  // (4)
          System.out.println("    c=" + c);
     }
}
```

```
> java Rand
a=1
  b=0
    c=-1
>
```

An Example (cont.)

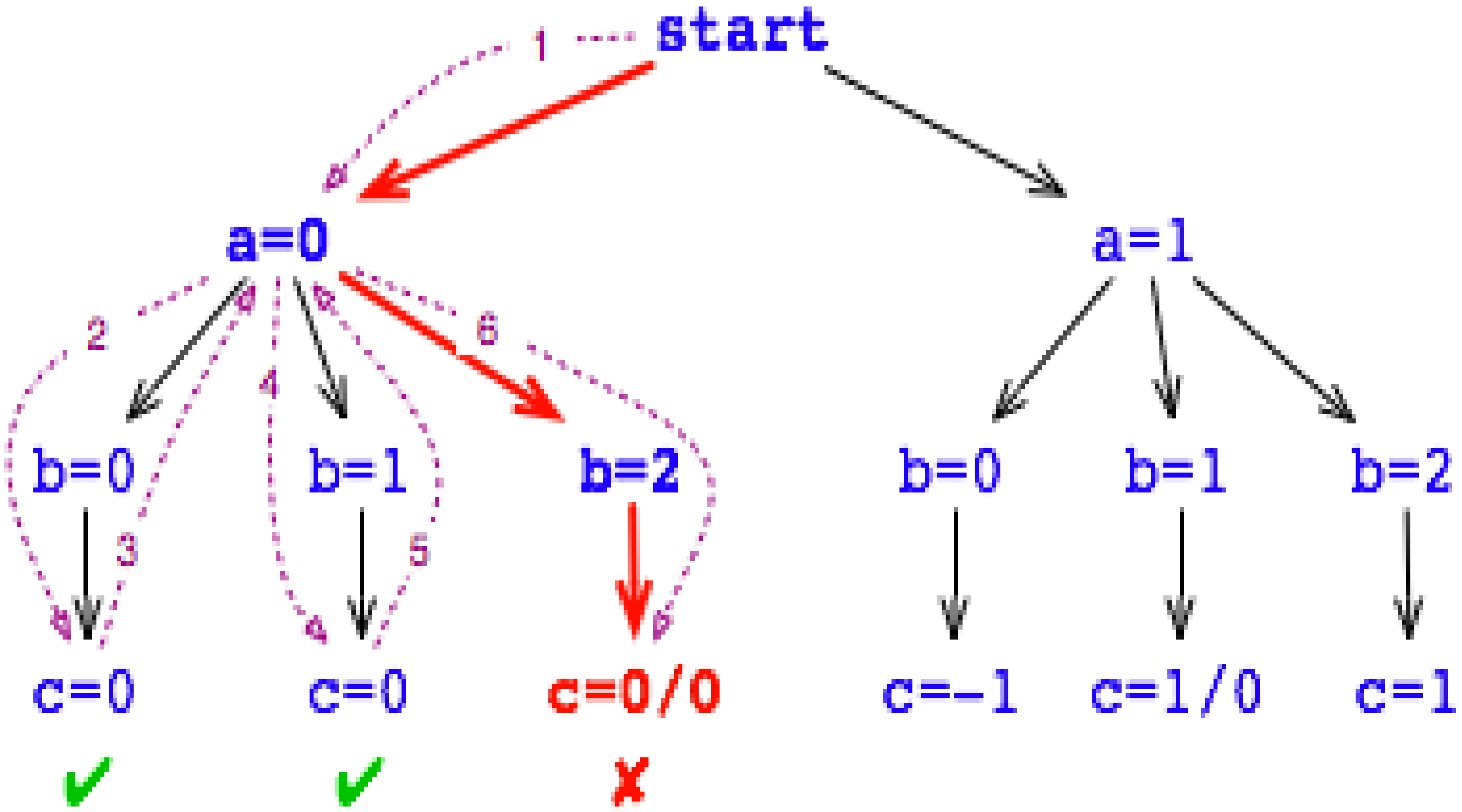


```
① Random random = new Random()

② int a = random.nextInt(2)

③ int b = random.nextInt(3)

④ int c = a/(b+a -2)
```

- **One execution corresponds to one path.**

```
> bin/jpf Rand

JavaPathfinder v4.1 - (C) 1999-2007 RIACS/NASA Ames Research Center
====================================================== system under test
application: /Users/pcmehlitz/tmp/Rand.java

====================================================== search started: 5/23/07 11:48 PM
a=1
  b=0
    c=-1

====================================================== results
no errors detected

====================================================== search finished: 5/23/07 11:48 PM
>
```

```
> bin/jpf +vm.enumerate_random=true Rand
JavaPathfinder v4.1 - (C) 1999-2007 RIACS/NASA Ames Research Center
======================================================= system under test
application: /Users/pcmehlitz/tmp/Rand.java

======================================================= search started: 5/23/07 11:49 PM
a=0
  b=0
    c=0
  b=1
    c=0
  b=2

======================================================= error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.ArithmeticException: division by zero
        at Rand.main(Rand.java:15)
....
>
```



- **JPF explores multiple possible executions GIVEN THE SAME CONCRETE INPUT**

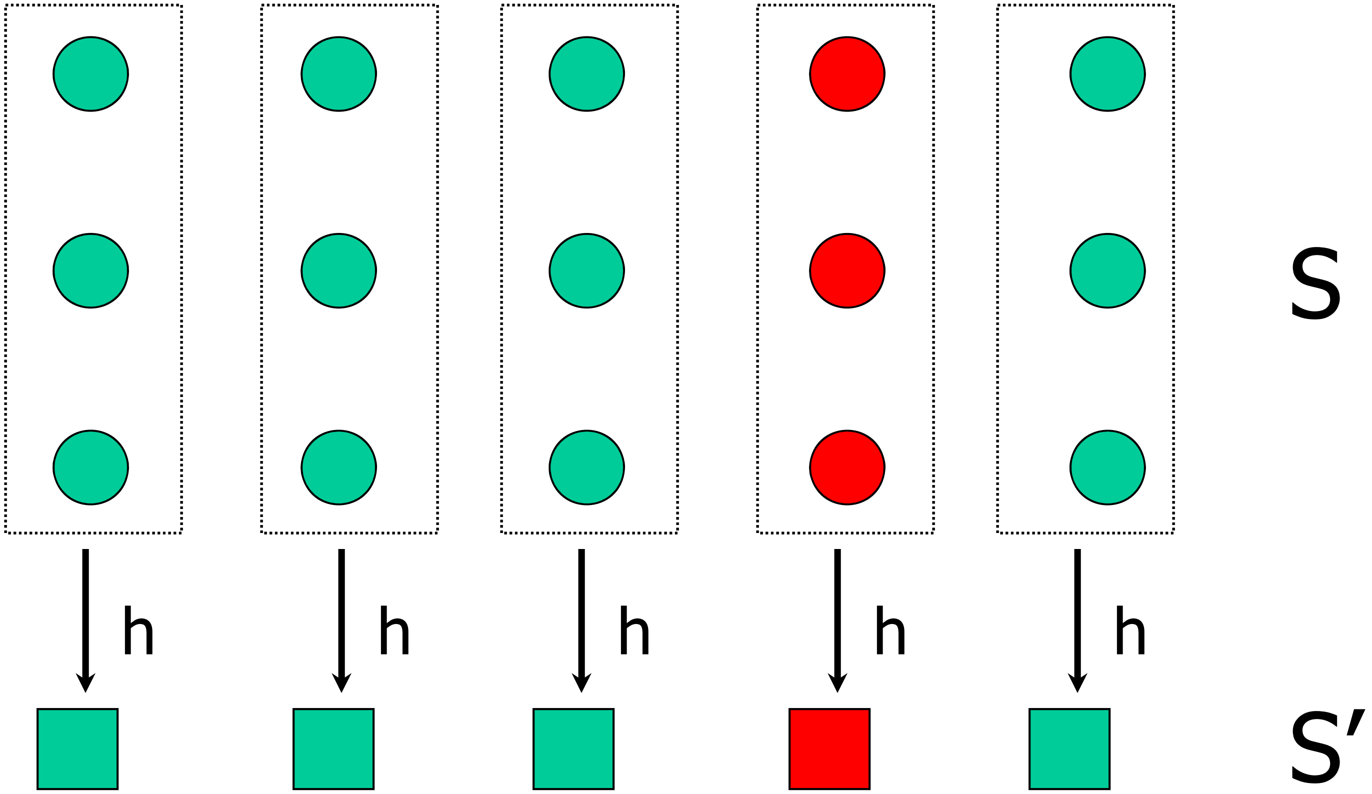
Two Essential Capabilities

- Backtracking
    - Means that JPF can restore previous execution states, to see if there are unexplored choices left.
        - While this is theoretically can be achieved by re-executing the program from the beginning, backtracking is a much more efficient mechanism if state storage is optimized.
- State matching
    - JPF checks every new state if it already has seen an equal one, in which case there is no use to continue along the current execution path, and JPF can backtrack to the nearest non-explored non-deterministic choice
        - Heap and thread-stack snapshots.

State Abstraction

- Eliminate details irrelevant to the property
- Obtain simple finite models sufficient to verify the property
- Disadvantage
  - Loss of Precision: False positives/negatives

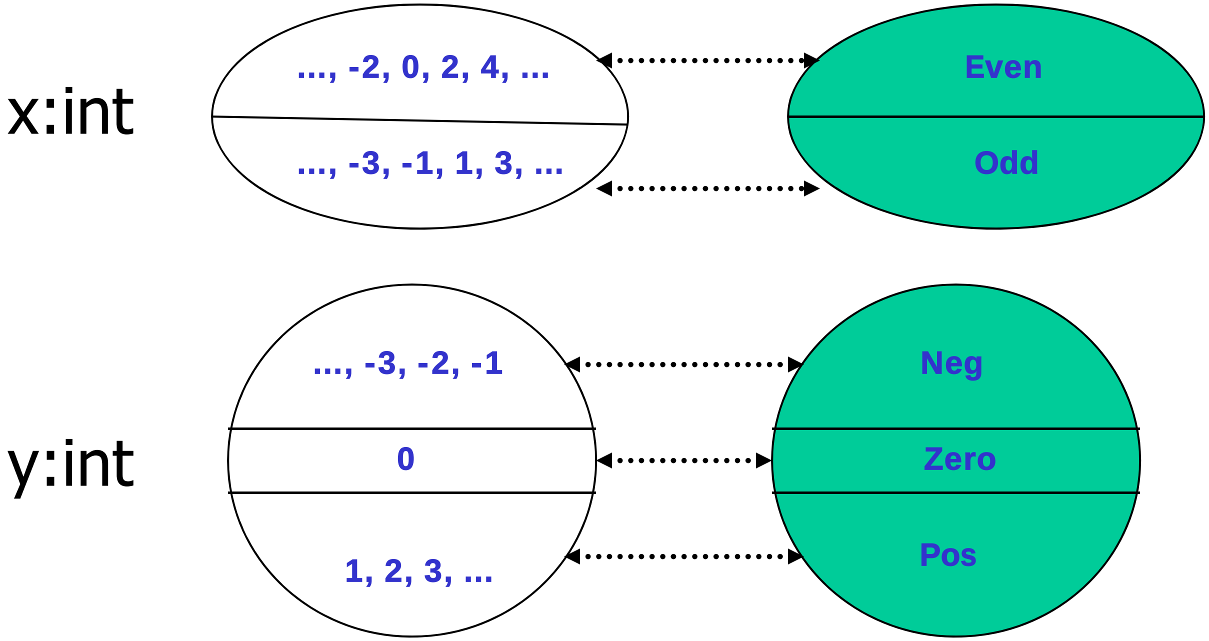
Data Abstraction



Abstraction Function h : from S to S′

Data Abstraction Example

- Abstraction proceeds component-wise, where variables are components


x:int

…, -2, 0, 2, 4, … ↔ Even

…, -3, -1, 1, 3, … ↔ Odd

y:int

…, -3, -2, -1 ↔ Neg

0 ↔ Zero

1, 2, 3, … ↔ Pos

How do we Abstract Behaviors?

- Abstract domain *A*
  - Abstract concrete values to those in *A*
- Then compute transitions in the abstract domain

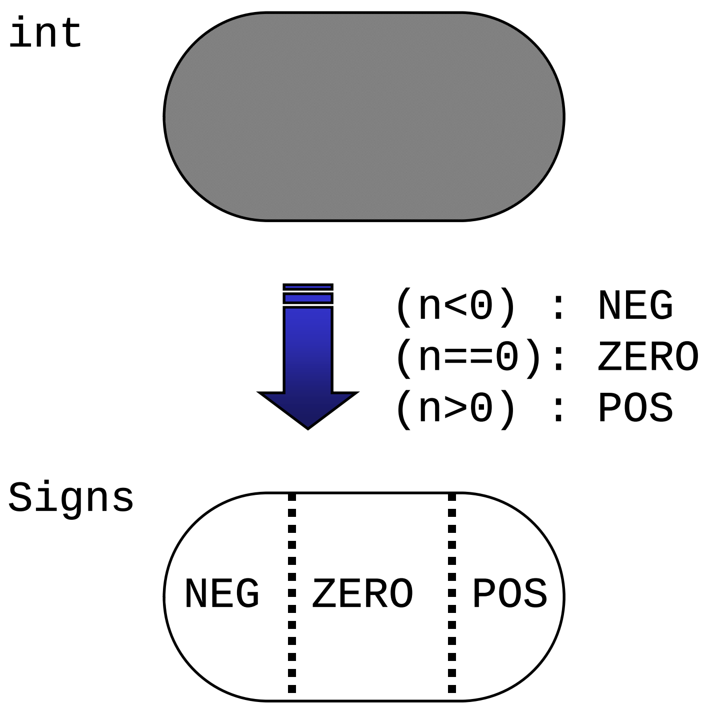
Data Type Abstraction

Code

```
int x = 0;
if (x == 0)
  x = x + 1;
```

```
Signs x = ZERO;
if (Signs.eq(x,ZERO))
  x = Signs.add(x,POS);
```

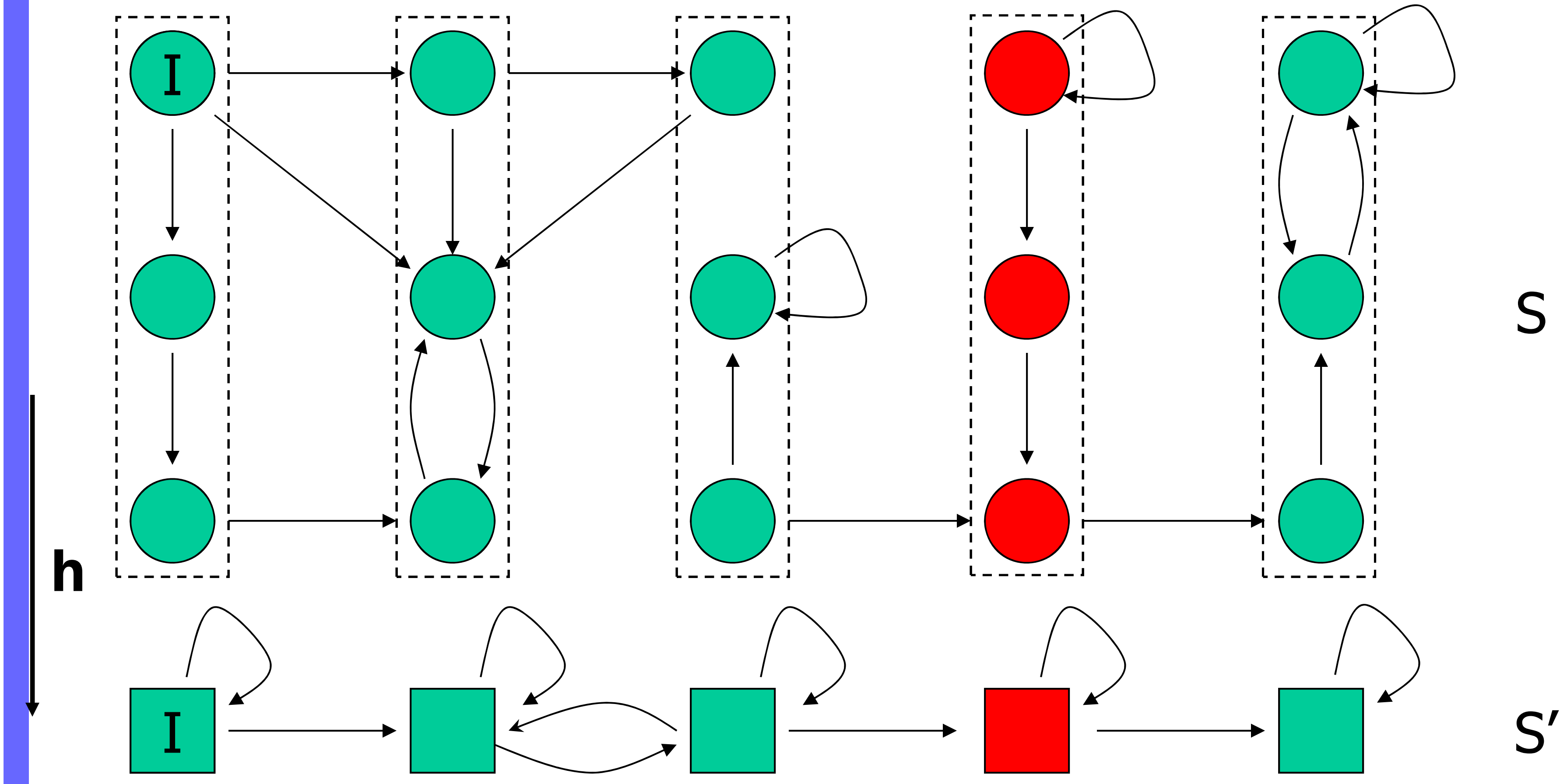
Abstract Data domain



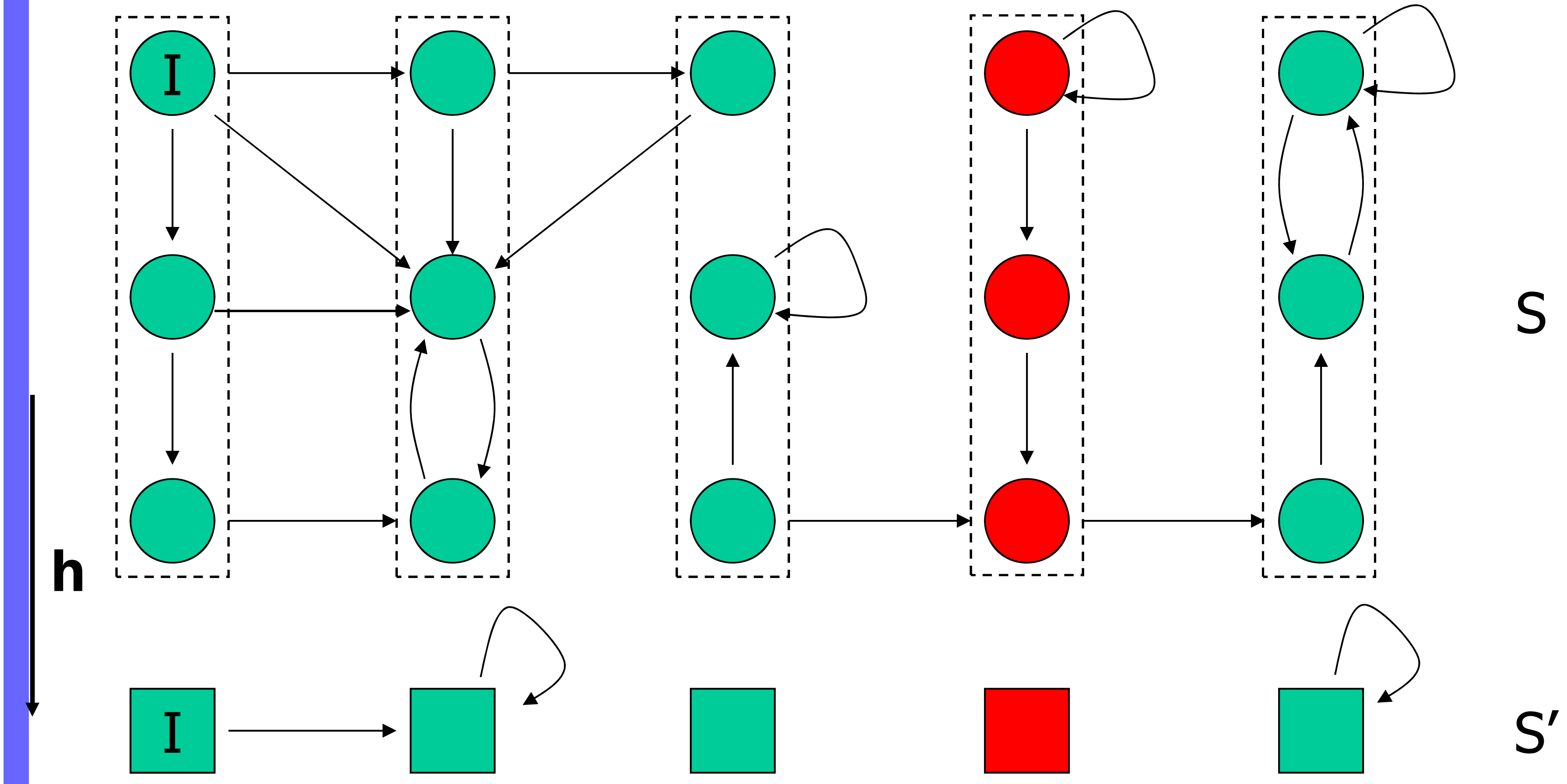
Existential/Universal Abstractions

- Existential
  - Make a transition from an abstract state if **at least one** corresponding concrete state has the transition.
  - Abstract model M' **simulates** concrete model M
- Universal
  - Make a transition from an abstract state if **all** the corresponding concrete states have the transition.

Existential Abstraction (Over-approximation)



Universal Abstraction (Under-Approximation)



Guarantees from Abstraction

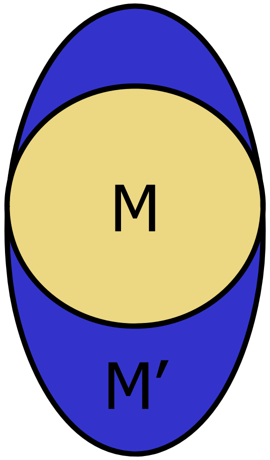
Assume M’ is an abstraction of M

- Strong Preservation:
  - P holds in M’ iff P holds in M
- Weak Preservation:
  - P holds in M’ implies P holds in M

Guarantees from Exist. Abstraction

- Let φ be a *hold-for-all-paths* property
- M′ existentially abstracts M
- **Preservation Theorem**

$$M' \models \varphi \rightarrow M \models \varphi$$



- Converse does not hold

$$M' \not\models \varphi \not\rightarrow M \not\models \varphi$$

- $M' \not\models \varphi$: counterexample may be spurious

Guarantees from Univ. Abstraction

- Let φ be a existential-quantified property and M simulates M'

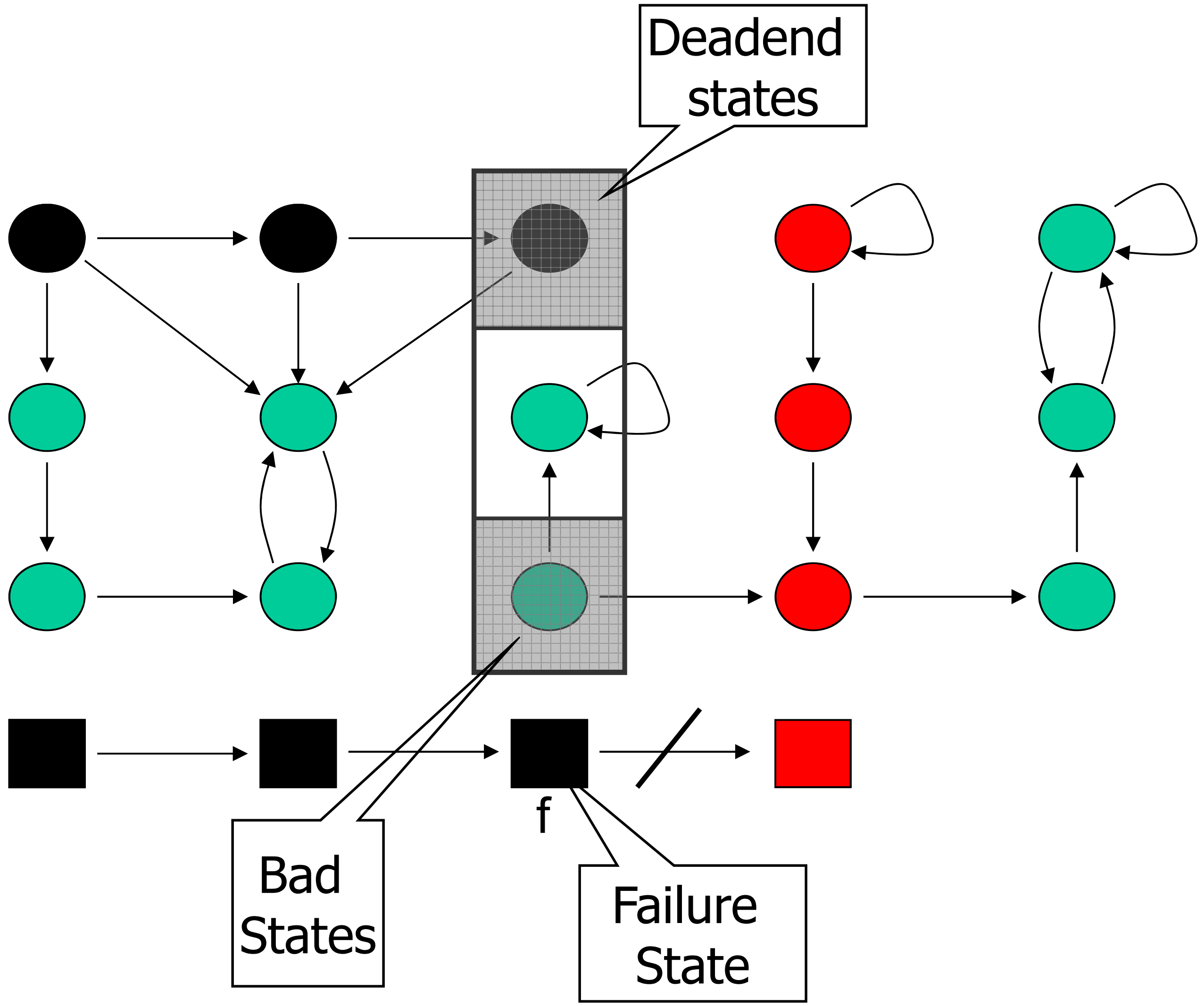
- Preservation Theorem

$$M' \models \varphi \;\rightarrow\; M \models \varphi$$

- Converse does not hold

$$M \not\models \varphi \;\not\rightarrow\; M' \not\models \varphi$$

Spurious counterexample in Over-approximation



Deadend states

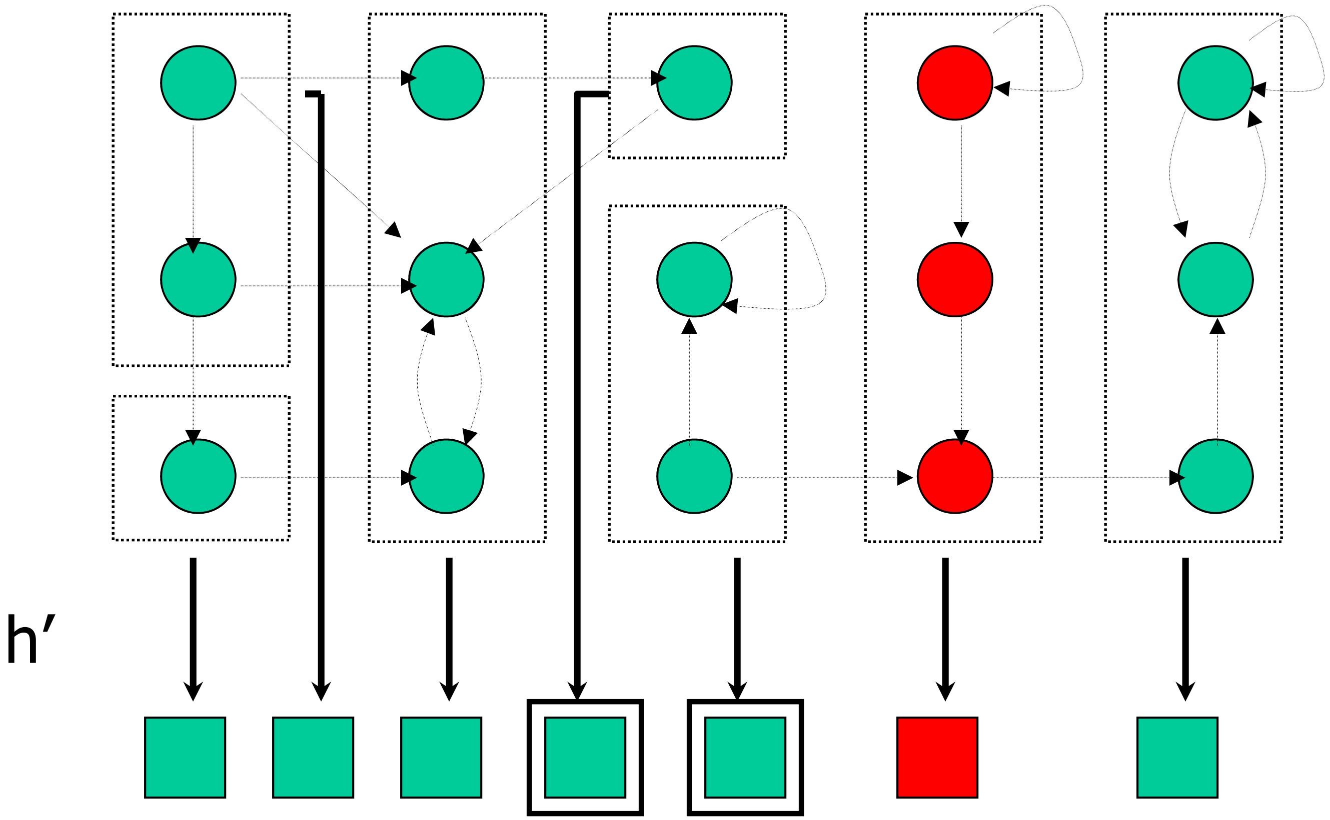
Bad States

Failure State

Refinement

- Problem: Deadend and Bad States are in the same abstract state.
- Solution: Refine abstraction function.
- The sets of Deadend and Bad states should be separated into different abstract states.

Refinement

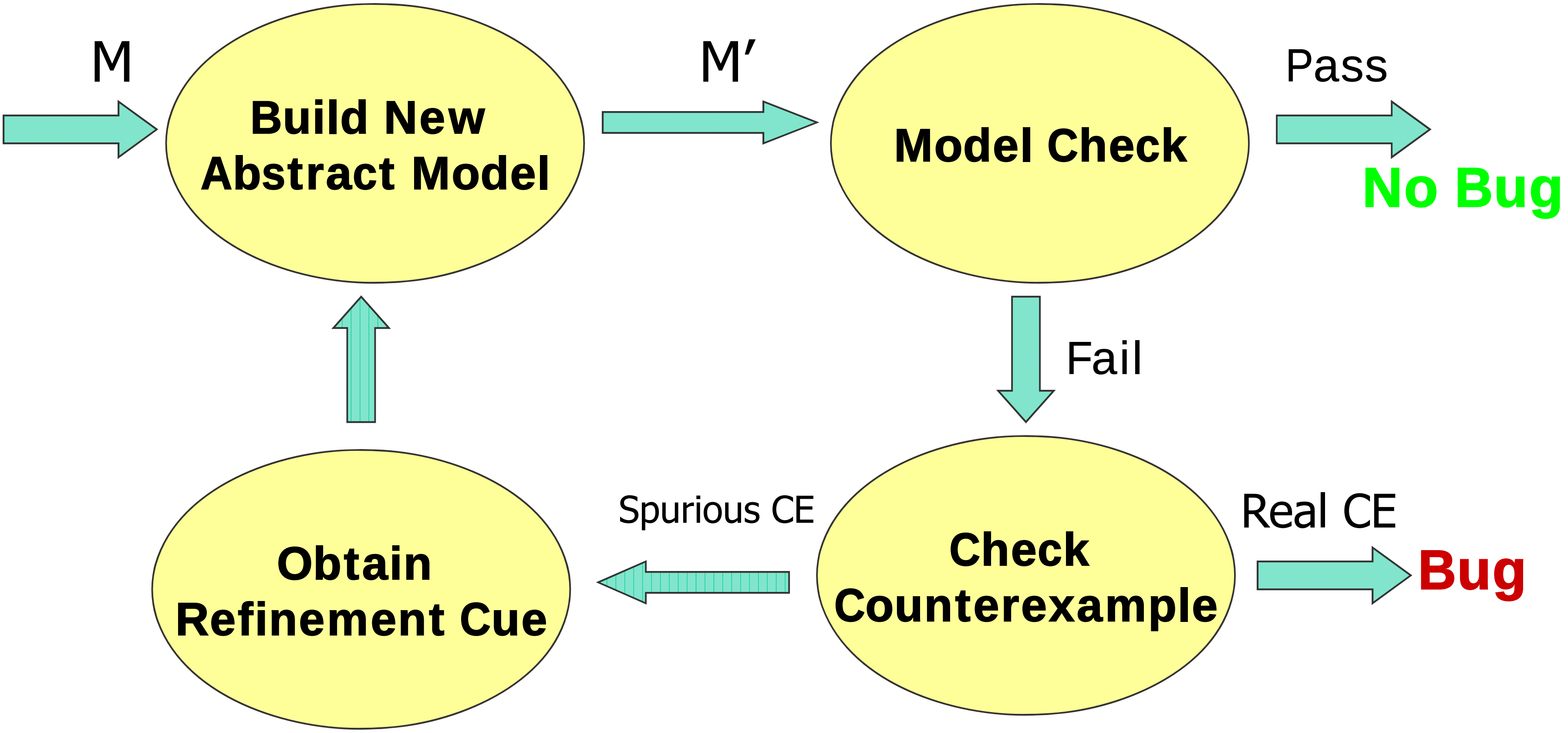


Refinement : h′

Automated Abstraction/Refinement

- Good abstractions are hard to obtain
  - Automate both Abstraction and Refinement processes
- Counterexample-Guided AR (CEGAR)
  - Build an abstract model M’
  - Model check property P, M’ ⊨ P?
  - If M’ ⊨ P, then M ⊨ P by Preservation Theorem
  - Otherwise, check if Counterexample (CE) is spurious
  - Refine abstract state space using CE analysis results
  - Repeat

Counterexample-Guided Abstraction-Refinement (CEGAR)



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

```
int main() {
  int i;

  i=0;

  while(even(i))
    i++;
}
```

+

$p_1 \Leftrightarrow i=0$

$p_2 \Leftrightarrow \text{even}(i)$

=

```
void main() {
  bool p1, p2;

  p1=TRUE;
  p2=TRUE;

  while(p2)
  {
    p1=p1?FALSE:nondet();
    p2=!p2;
  }
}
```

C program

Predicates

Boolean program

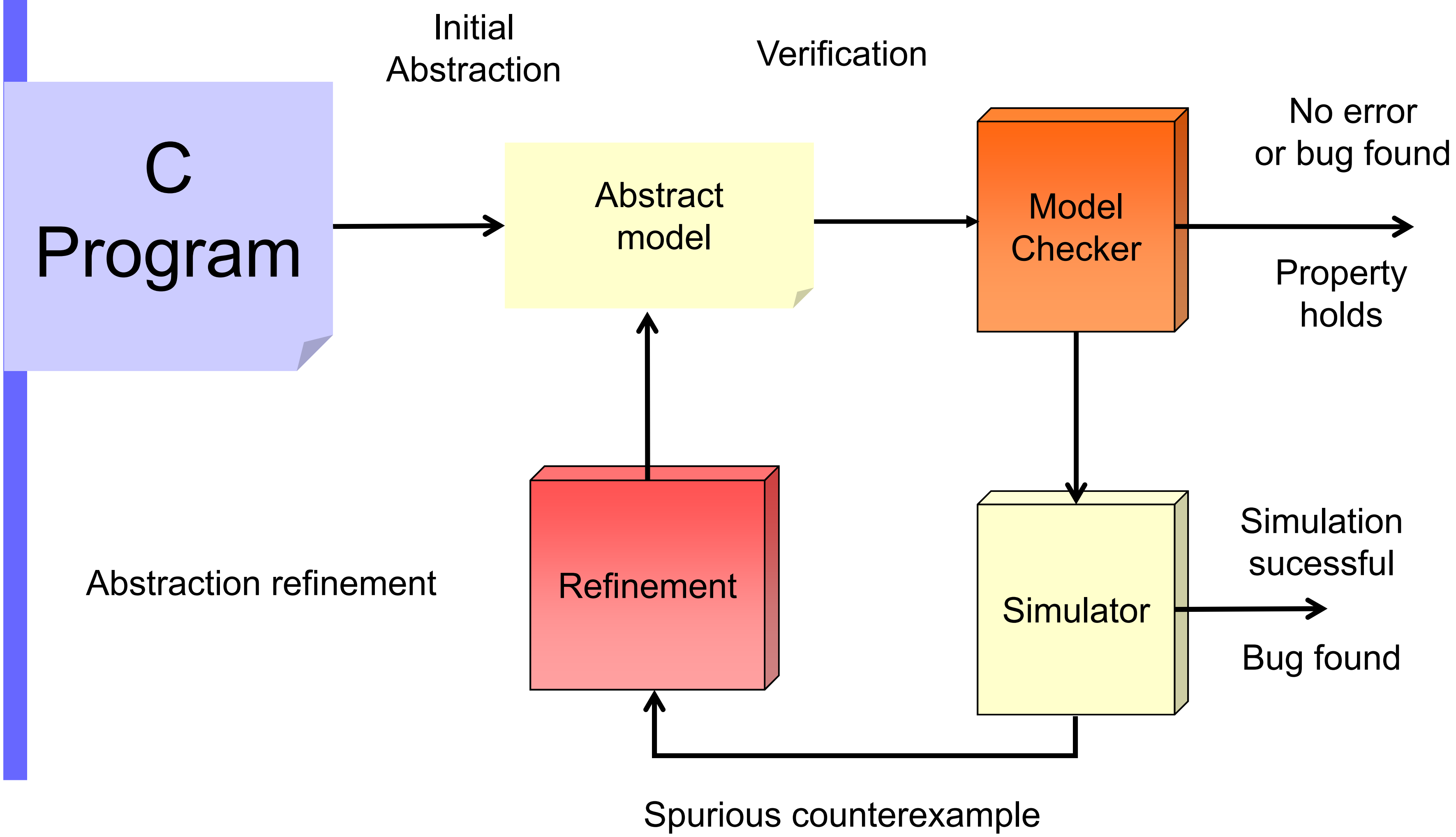
[Graf, Saidi ’97]

[Ball, Rajamani ’01]

Computing Predicate Abstraction

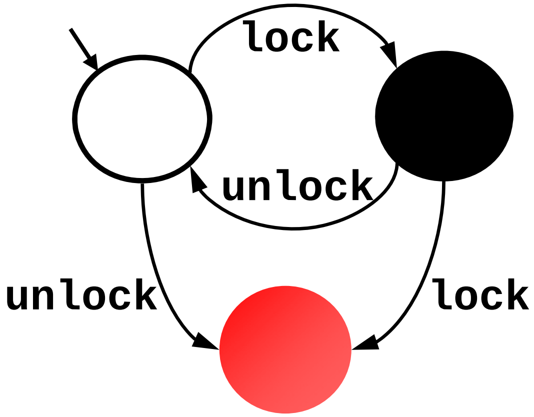
- How to get predicates for checking a given property?
- How do we compute the abstraction?
- Predicate Abstraction is an over-approximation
  - How to refine coarse abstractions

Counterexample Guided Abstraction Refinement loop

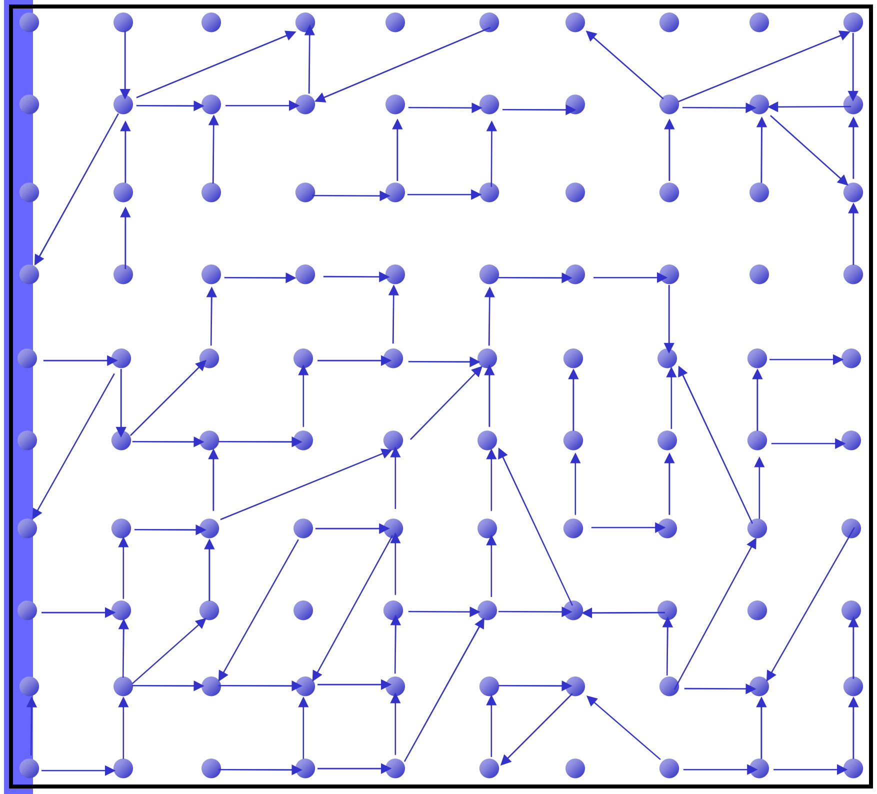


Example

```
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…



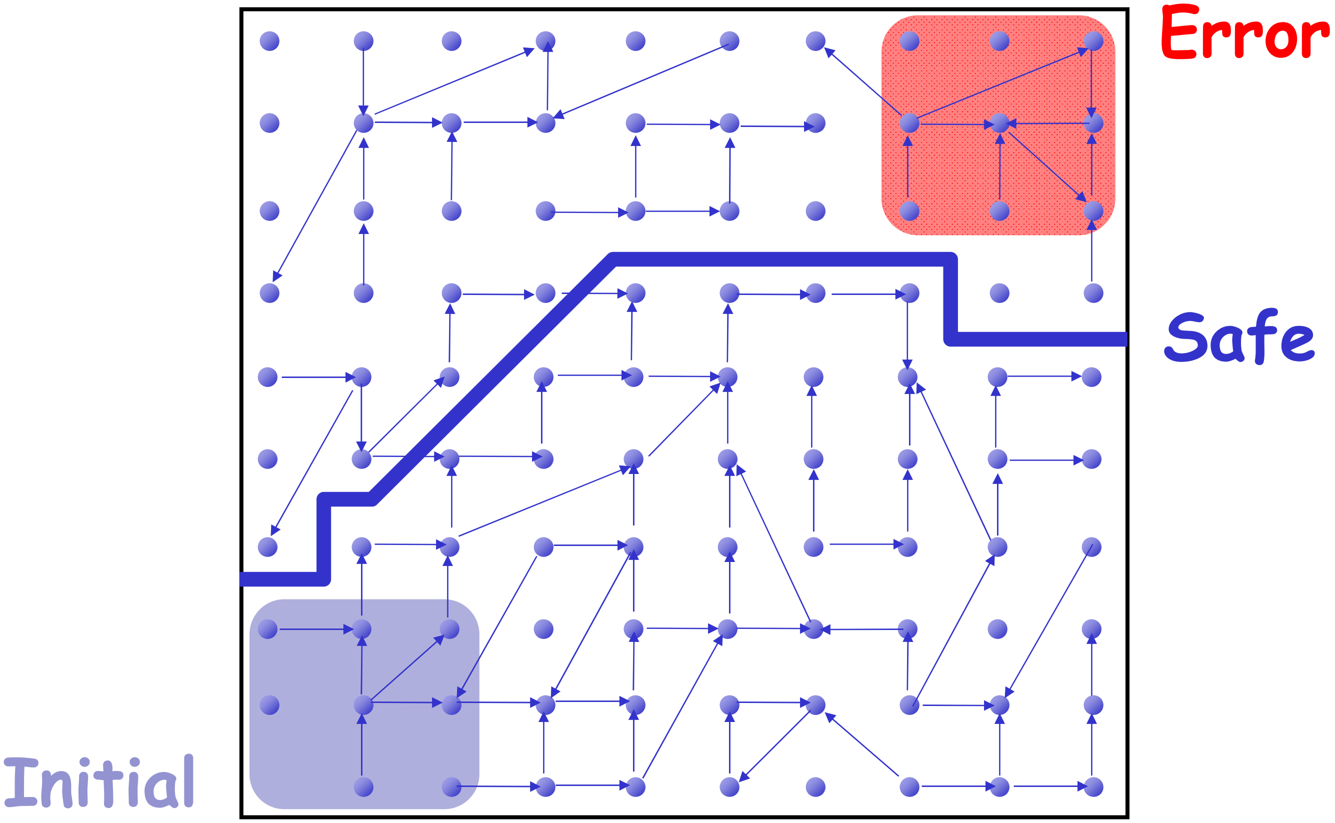
pc ↦ 3
lock ↦ ●
old ↦ 5
new ↦ 5
q ↦ 0x133a

```
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

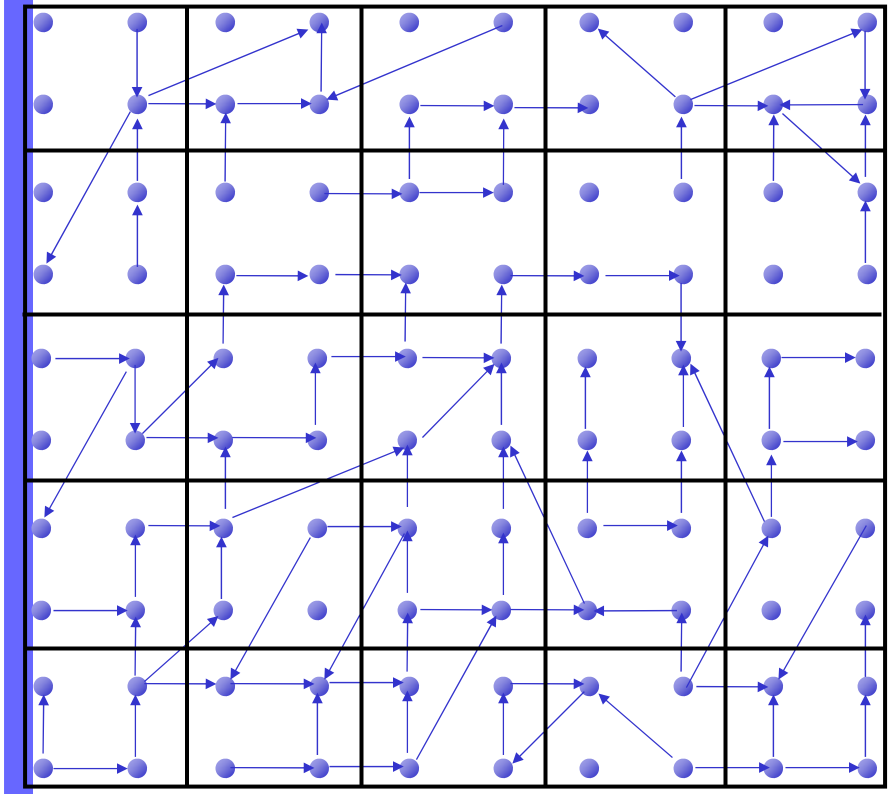


Is there a **path** from an initial to an **error** state ?

Problem: Infinite state graph

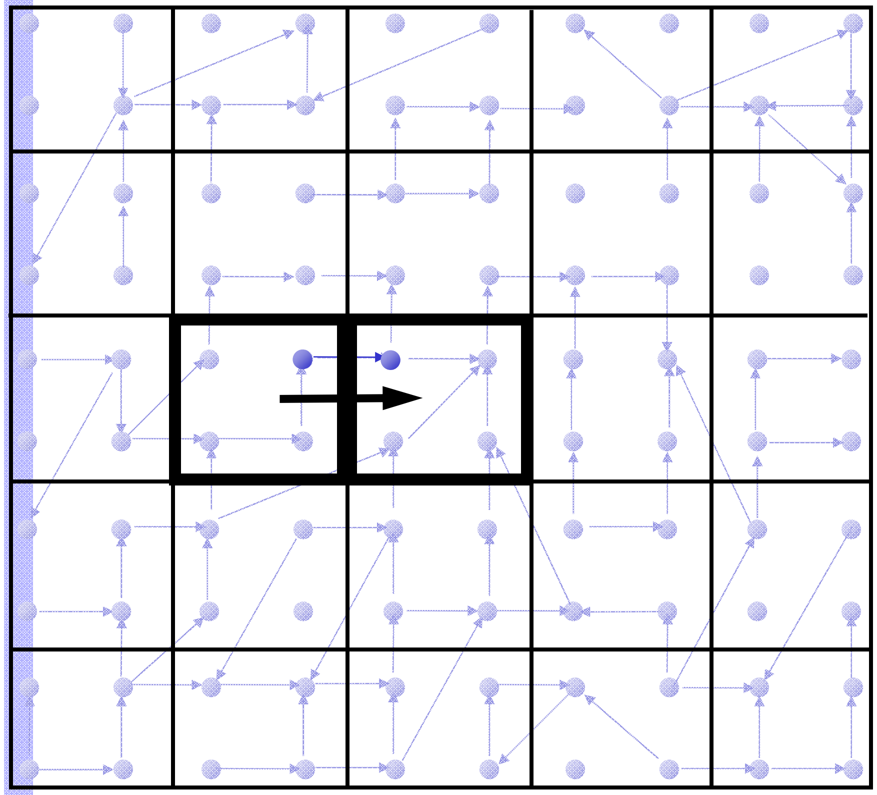
Solution : **Set** of states ' logical **formula**

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

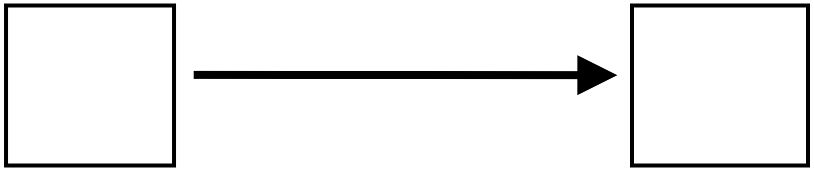


State

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

```
3: unlock();
   new++;
4:} …
```

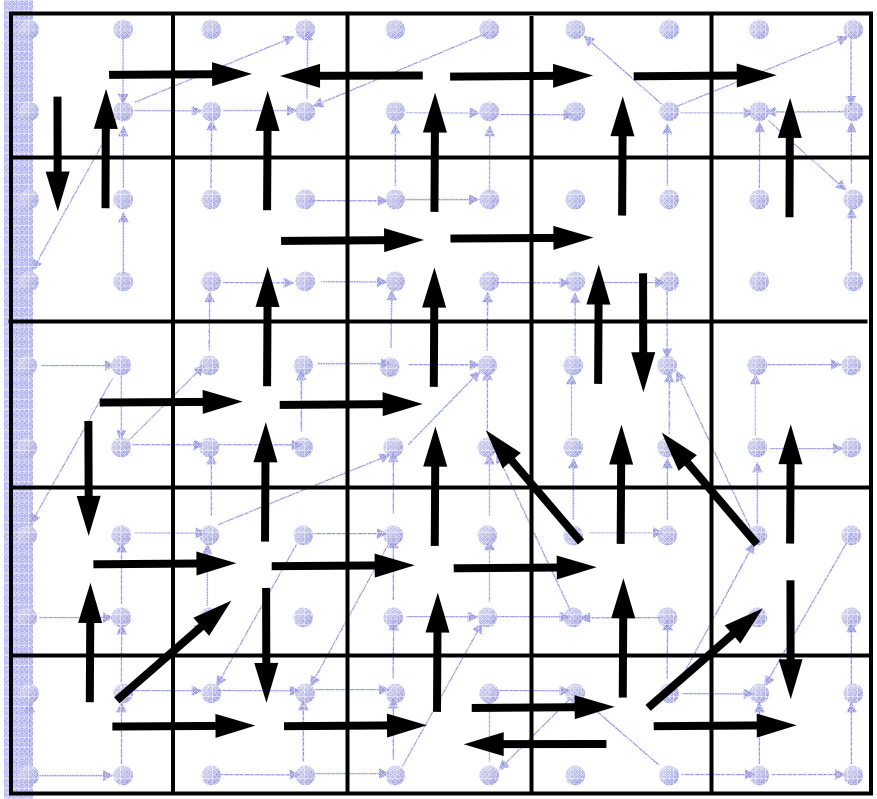
pc ↦ 4
lock ↦ ○
old ↦ 5
new ↦ 6
q ↦ 0x133a



lock
old=new

! lock
! old=new

Abstraction



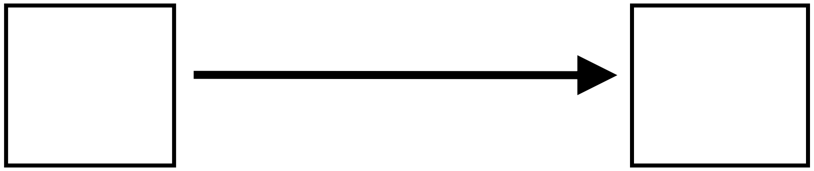
Existential Approximation

State

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

```
3: unlock();
   new++;
4:} …
```

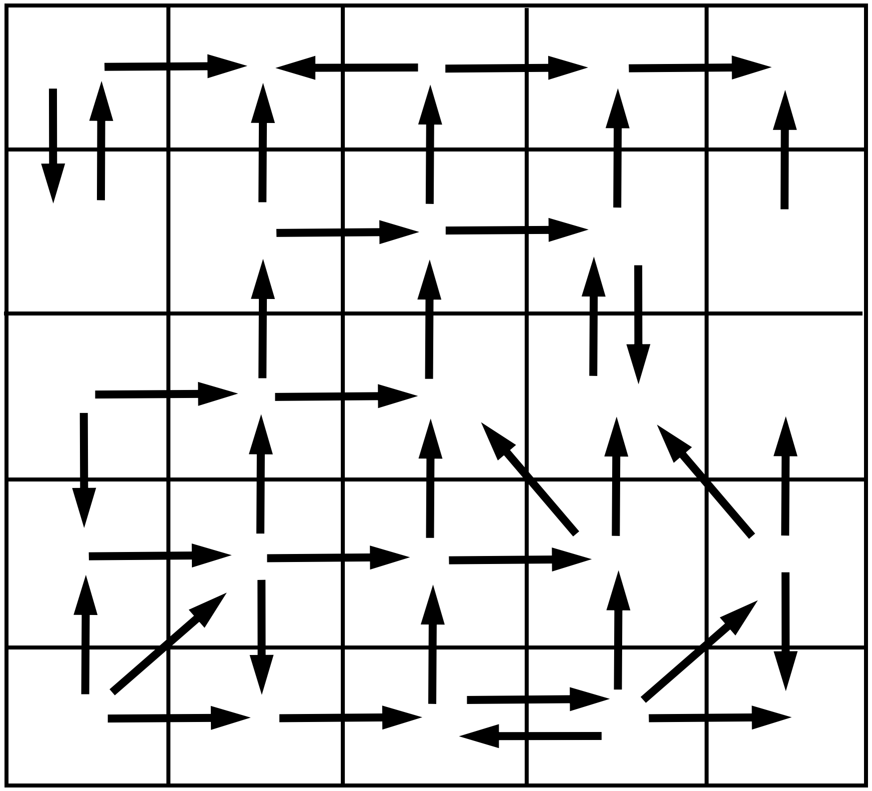
pc ↦ 4
lock ↦ ○
old ↦ 5
new ↦ 6
q ↦ 0x133a

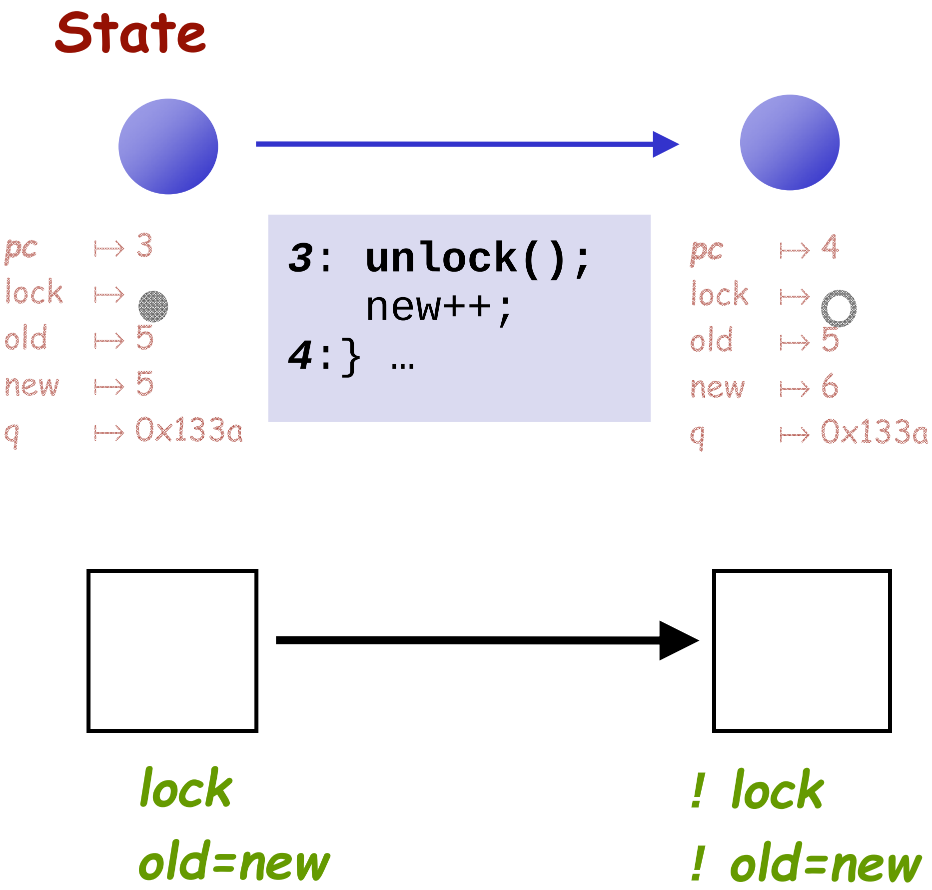


lock
old=new

! lock
! old=new

Abstraction





State

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

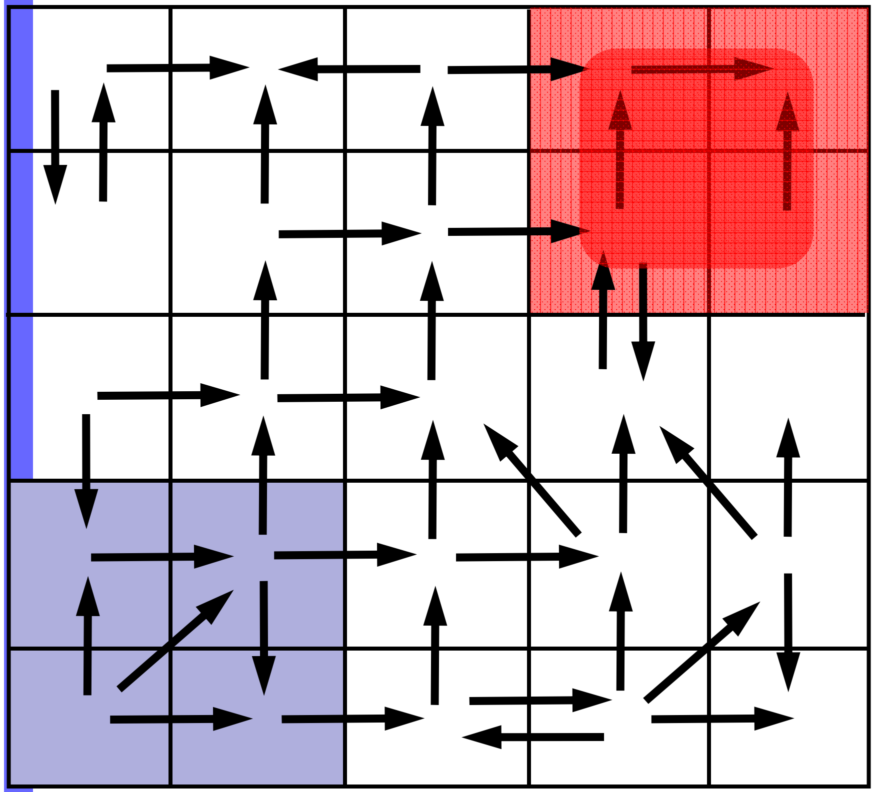
```
3: unlock();
   new++;
4:} …
```

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

lock
old=new

! lock
! old=new

Analyze Abstraction



Analyze finite graph

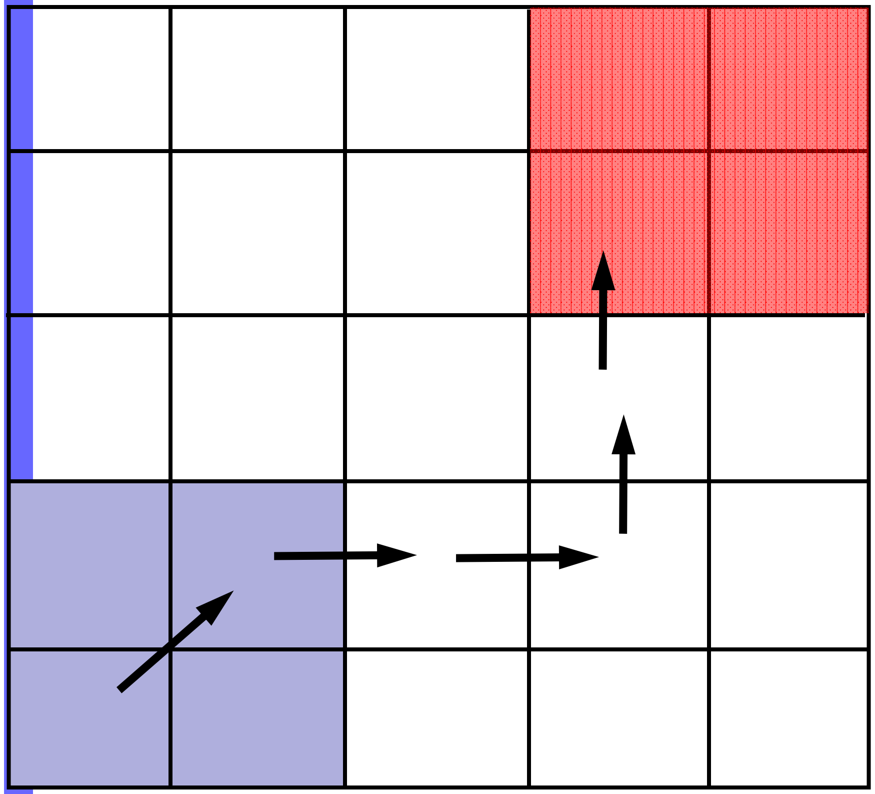
Over Approximate:

Safe => System Safe

Problem

Spurious **counterexamples**

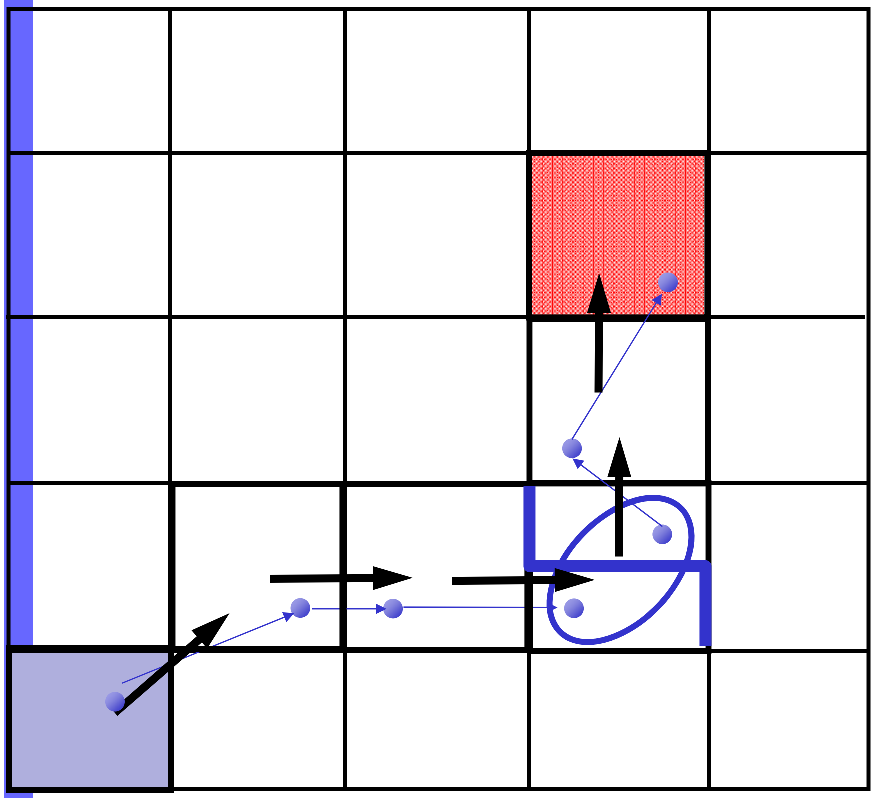
Idea 2: Counterex.-Guided Refinement



Solution

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

Idea 2: Counterex.-Guided Refinement

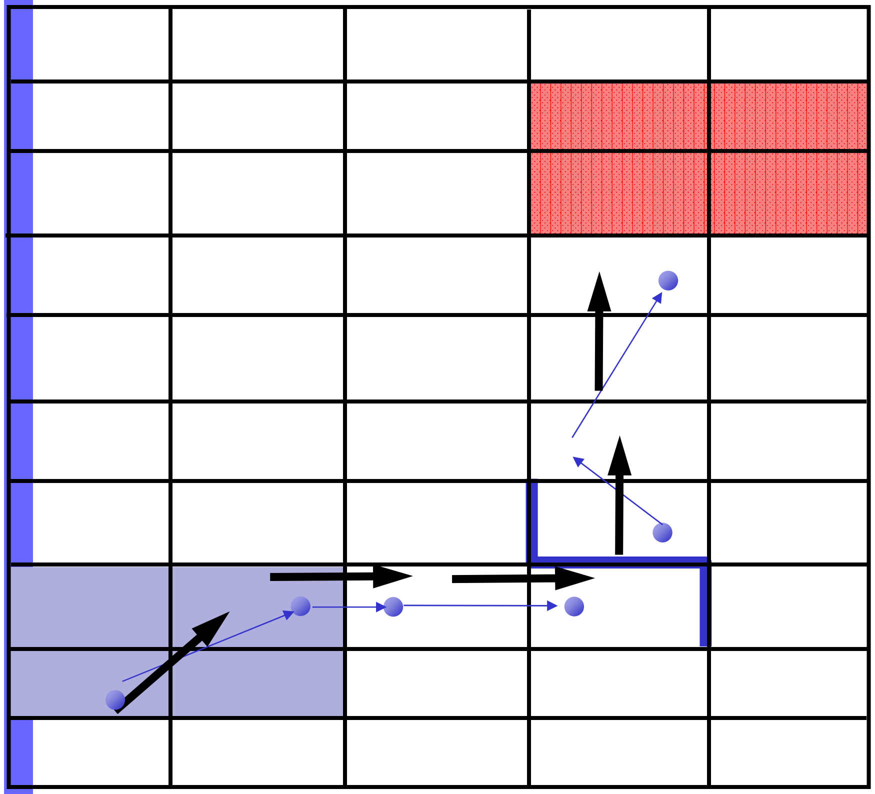


Solution

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

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

Iterative Abstraction-Refinement



Solution

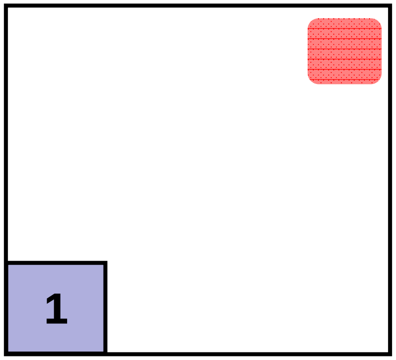
Use spurious **counterexamples** to **refine** abstraction

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

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

Build-and-Search

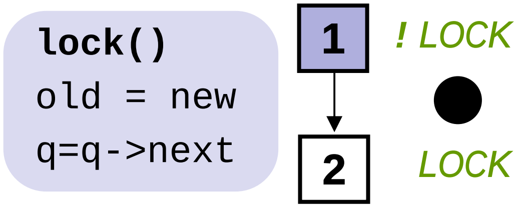
```
Example ( ) {
1: do{
       lock();
       old = new;
       q = q->next;
2:     if (q != NULL){
3:        q->data = new;
          unlock();
          new ++;
       }
4:}while(new != old);
5: unlock ();
}
```



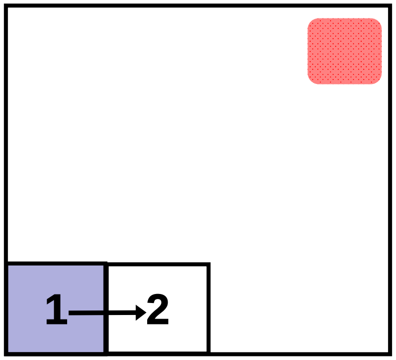
Predicates: *LOCK*

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 ();
}
```



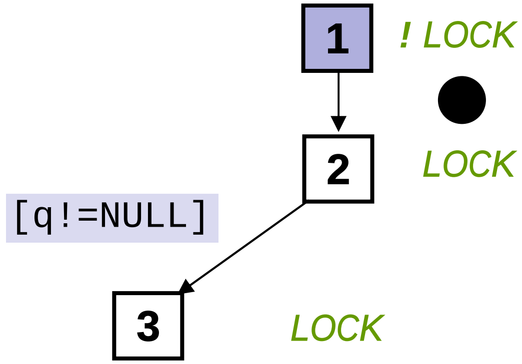
```
lock()
old = new
q=q->next
```

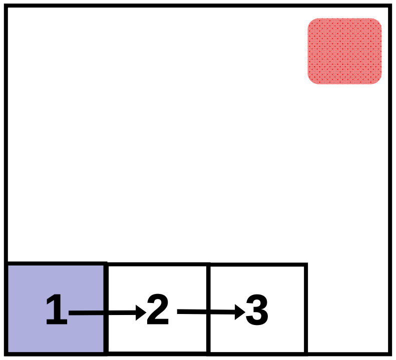


Predicates: *LOCK*

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 ();
}
```

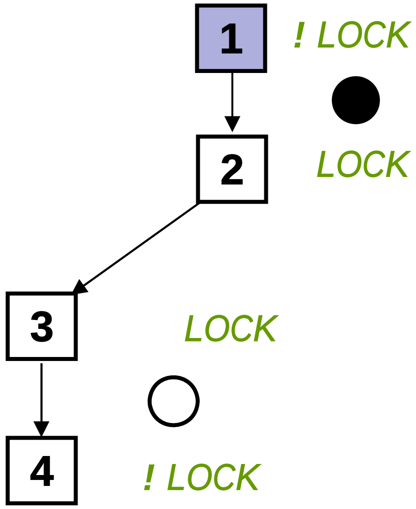




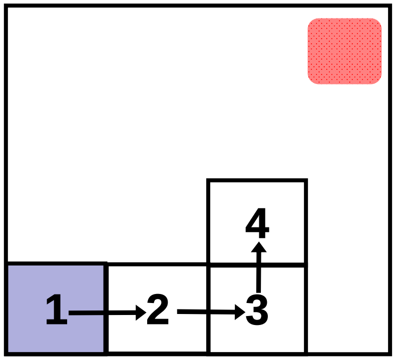
Predicates: *LOCK*

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 ();
}
```



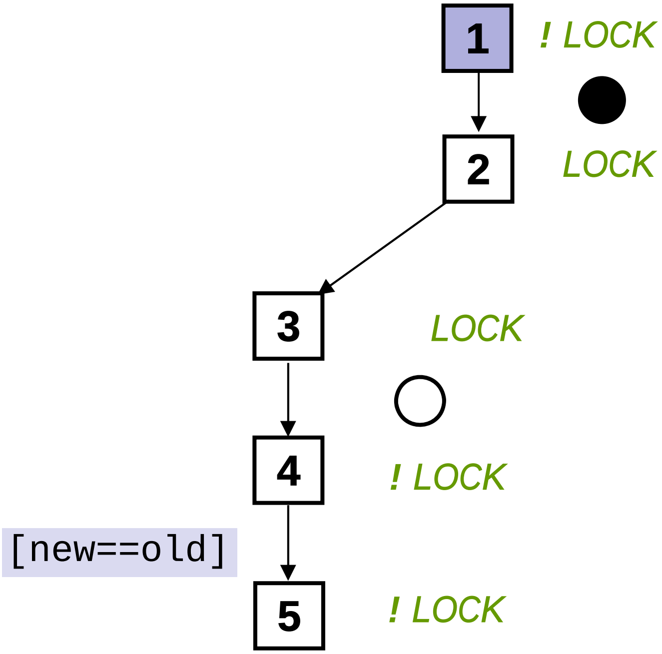
```
q->data = new
unlock()
new++
```

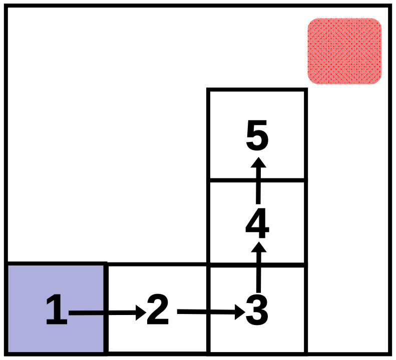


Predicates: *LOCK*

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 ();
}
```

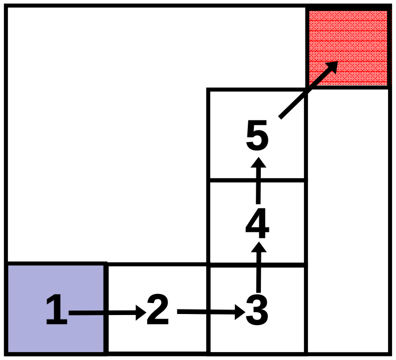


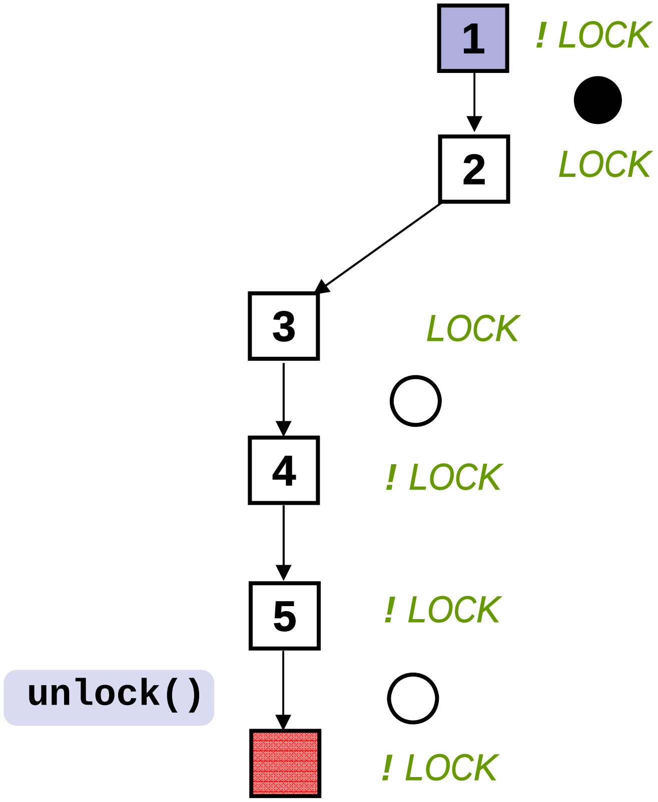


Predicates: LOCK

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 ();
}
```

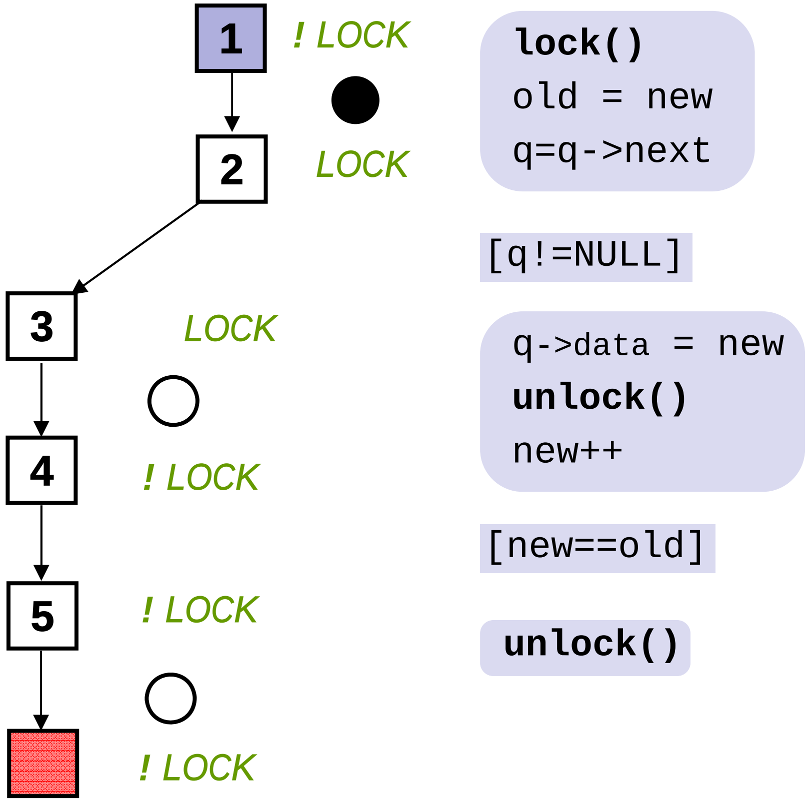




Predicates: LOCK

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

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

LOCK

`[q!=NULL]`

LOCK

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

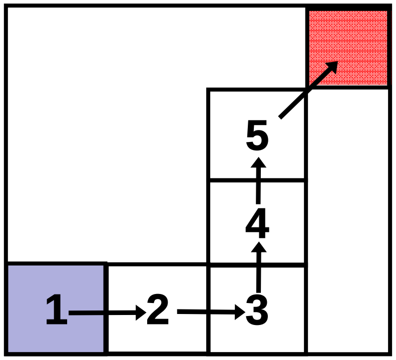
! LOCK

`[new==old]`

! LOCK

`unlock()`

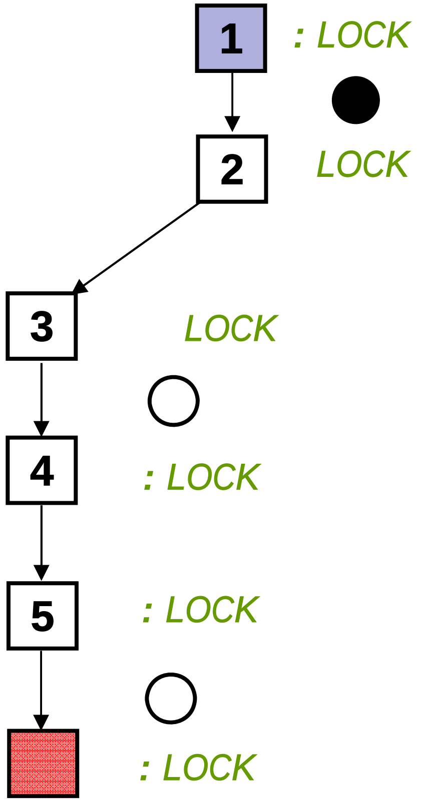
! LOCK



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 ();
}
```



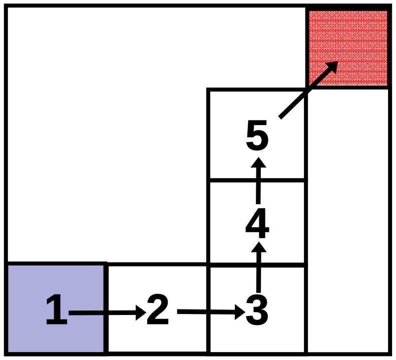
old = new

new++

[new==old]

Inconsistent

new == old

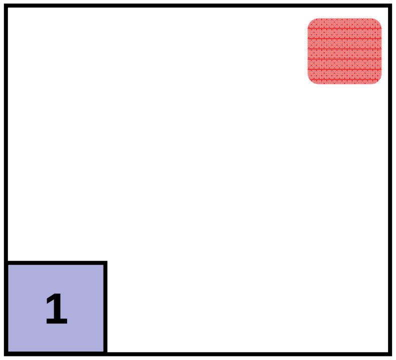


Predicates: *LOCK*

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 ();
}
```

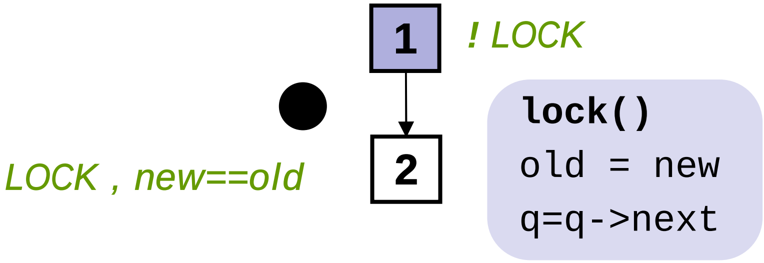
1 : LOCK

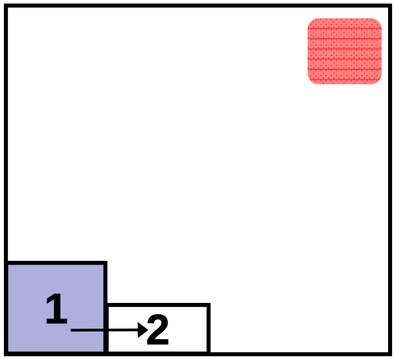


Predicates: *LOCK, new==old*

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 ();
}
```

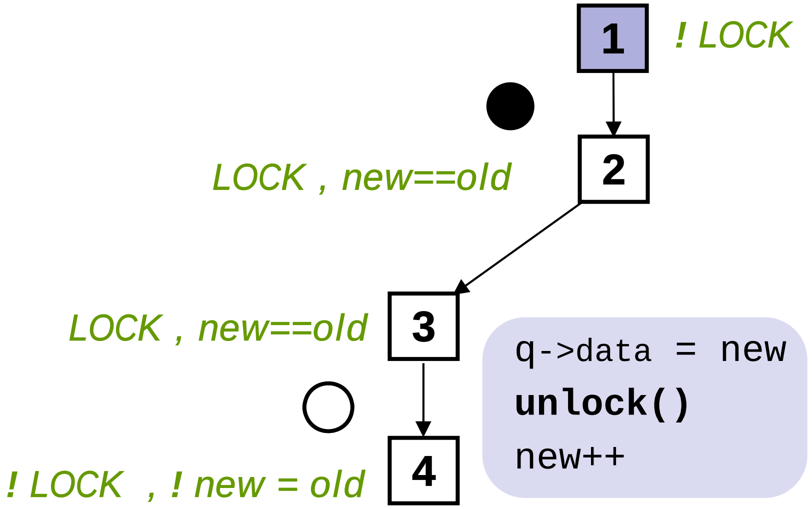




Predicates: *LOCK, new==old*

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 ();
}
```



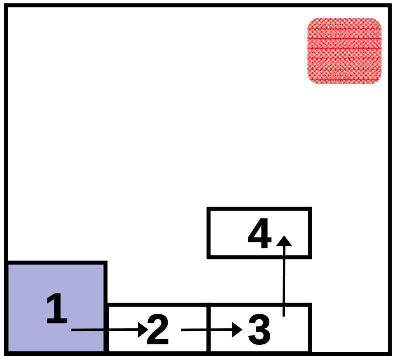
! LOCK

LOCK , new==old

LOCK , new==old

! LOCK , ! new = old

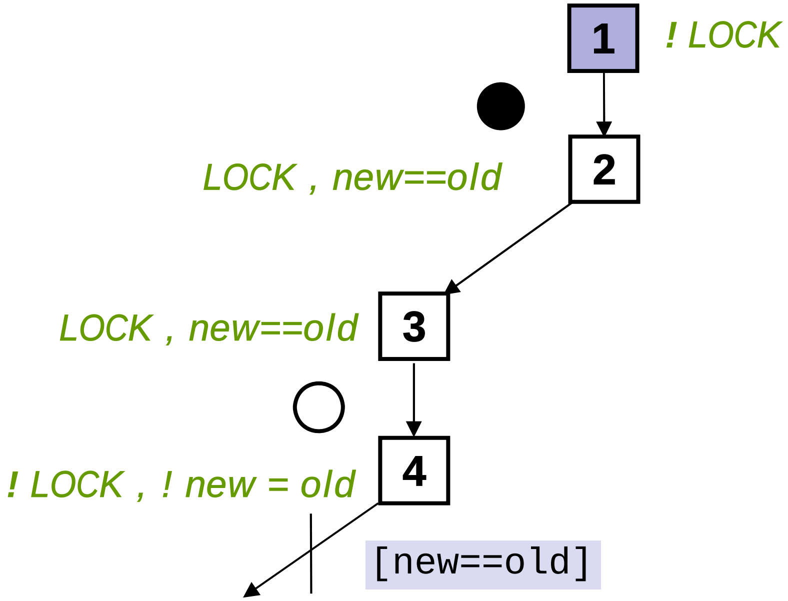
```
q->data = new
unlock()
new++
```

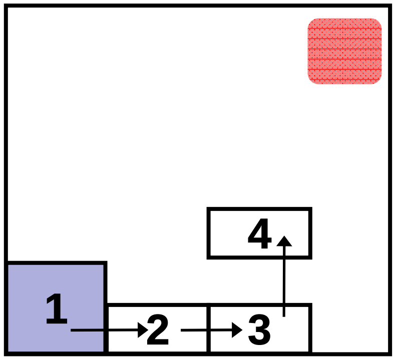


Predicates: LOCK, new==old

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 ();
}
```

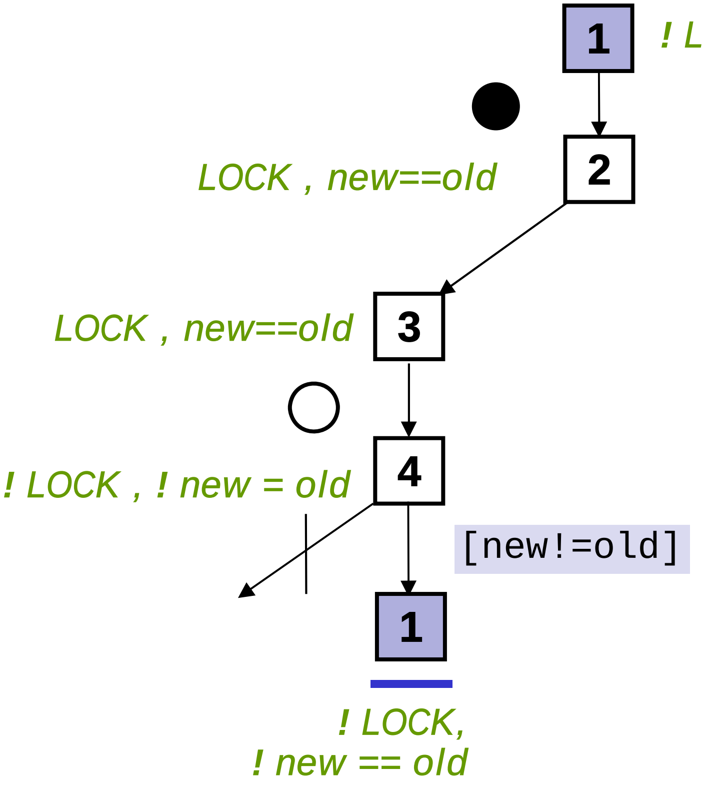


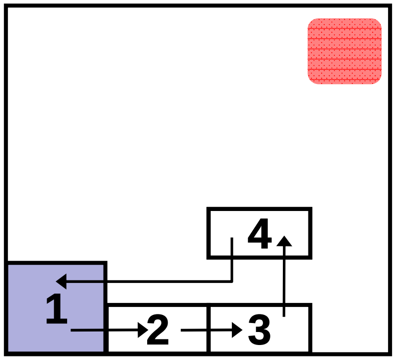


Predicates: LOCK, new==old

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 ();
}
```

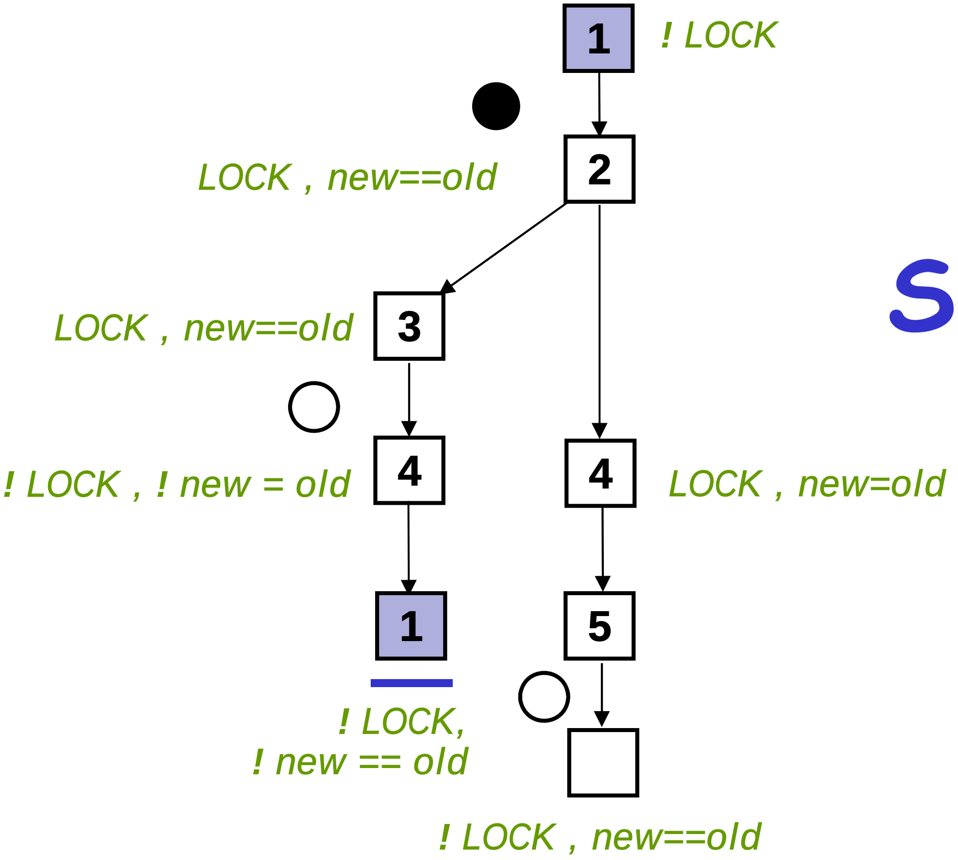




Predicates: LOCK, new==old

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 ();
}
```



! LOCK

LOCK , new==old

LOCK , new==old

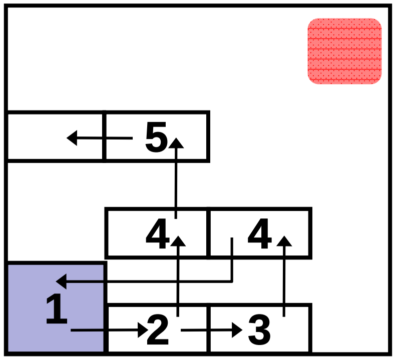
! LOCK , ! new = old

LOCK , new=old

! LOCK,
! new == old

! LOCK , new==old

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

Is ERROR reachable?

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

yes

Abstract model has a path leading to error state

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)
```

Concrete trace

Check using a SAT solver

Not possible

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

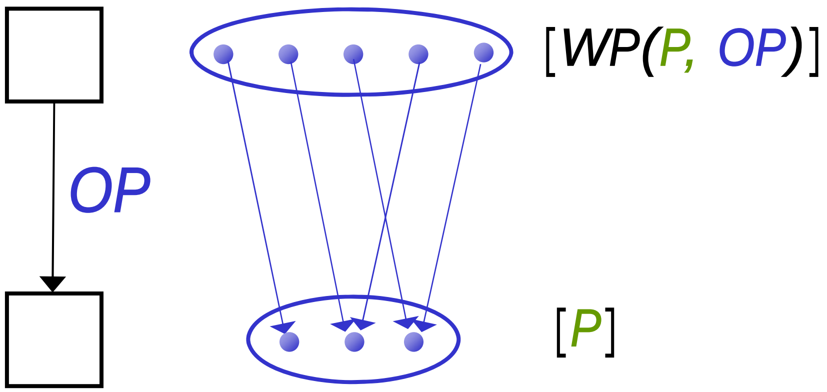
Weakest Preconditions

WP(P,OP)

Weakest formula P′ s.t.

if P′ is true before OP

then P is true after OP



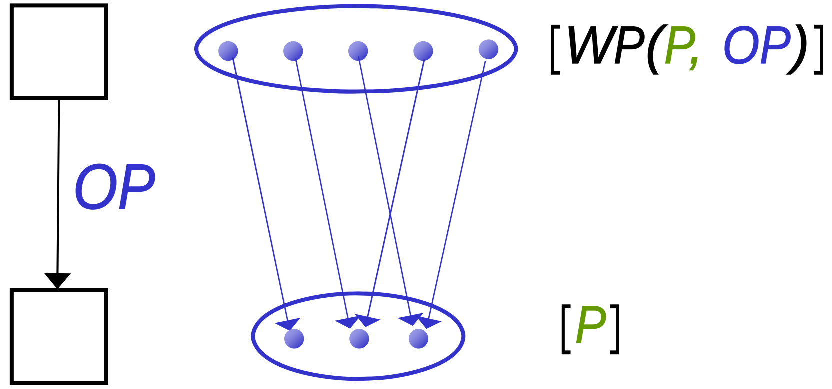
Weakest Preconditions

$WP(P, OP)$

Weakest formula P' s.t.

if P' is true before OP

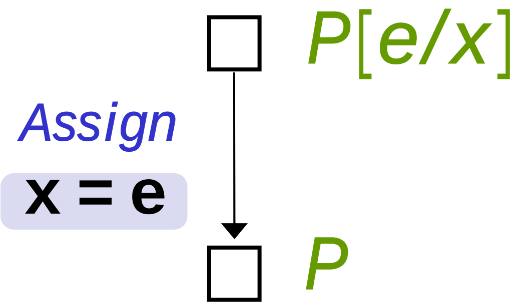
then P is true after OP



OP

$[WP(P, OP)]$

$[P]$

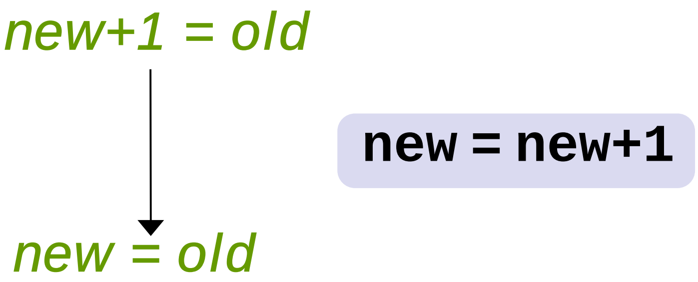


$P[e/x]$

Assign

x = e

P



new+1 = old

new = new+1

new = old

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

```
x = ctr;
y = ctr + 1;
assume(x == i-1)
assume (y != i)
```

boolean b2 : ctr + 1 ! = i

boolean b3: ctr == i -1

```
b1 = b3;
b0 = b2;
if (b1){
   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!

```
b1 = b3;
b0 = b2;
if (b1){
  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