Software Model Checking with Abstraction Refinement

Computer Science and Artificial Intelligence Laboratory

MIT

Armando Solar-Lezama

With slides from Thomas Henzinger, Ranjit Jhala and Rupak Majumdar.
Used with permission.

Dec 08, 2011

Model checking so far

The promise of model checking

- Exhaustive exploration of the state space of a program
- Push-button verification of arbitrary temporal logic formulas
- Dramatic performance improvements from
  - State reduction techniques
  - Symbolic representations

But

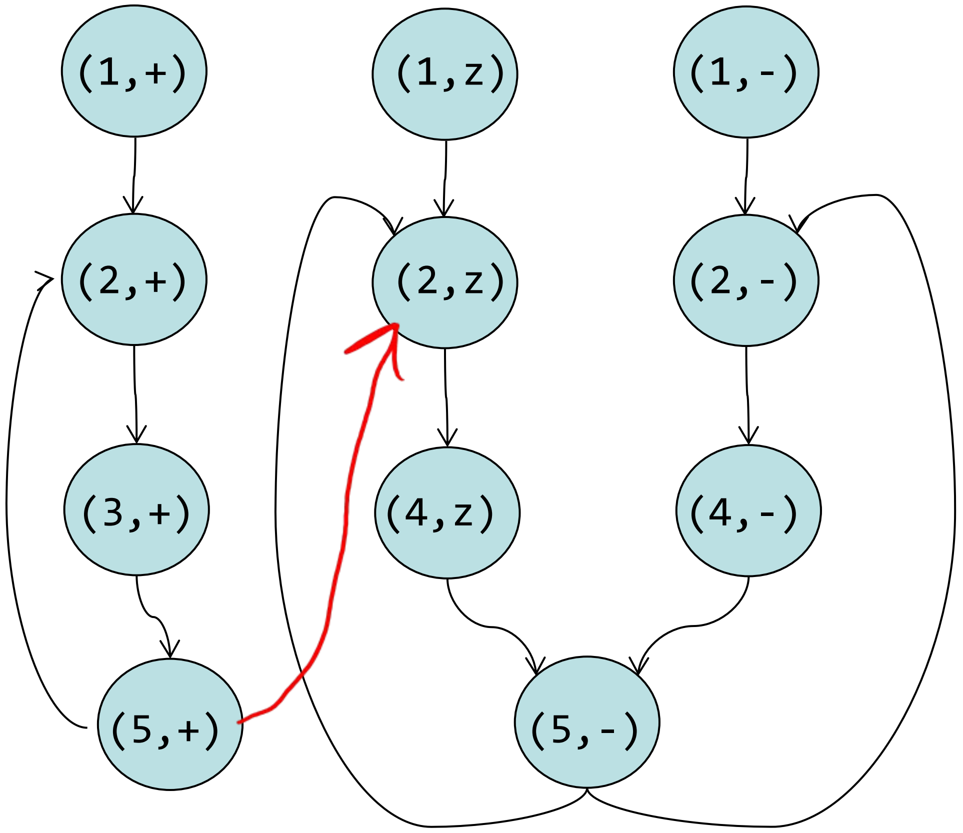
- It only works for programs with bounded state space

Abstraction to the rescue

We can abstract the infinite state space into a finite one

- Every abstract state corresponds to an infinite set of states
- Is this the same thing as abstract interpretation?

```
    void main(){
1:    int x = *;
      while(*){
2:      if(x>0)
3:        x = 2*x;
        else
4:        x = x-1;

5:      x = abs(*)/x;
      }
    }
```

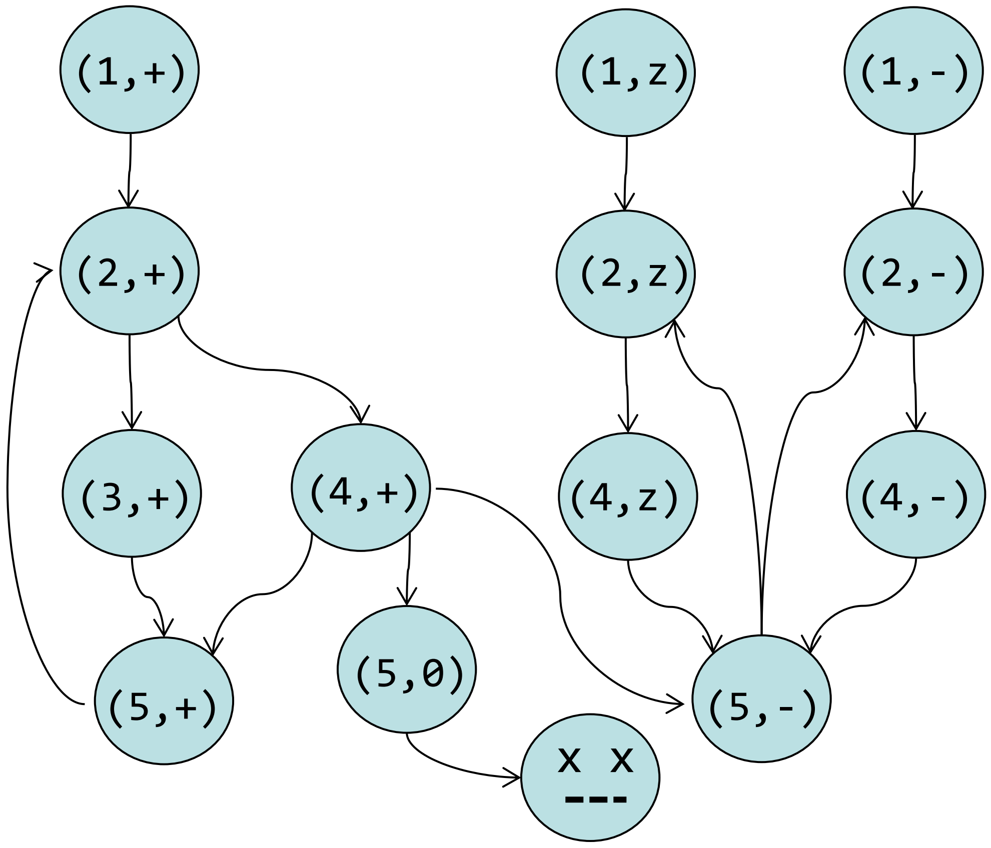


The problem with abstraction

Abstractions usually have to be tailored to the program and property of interest

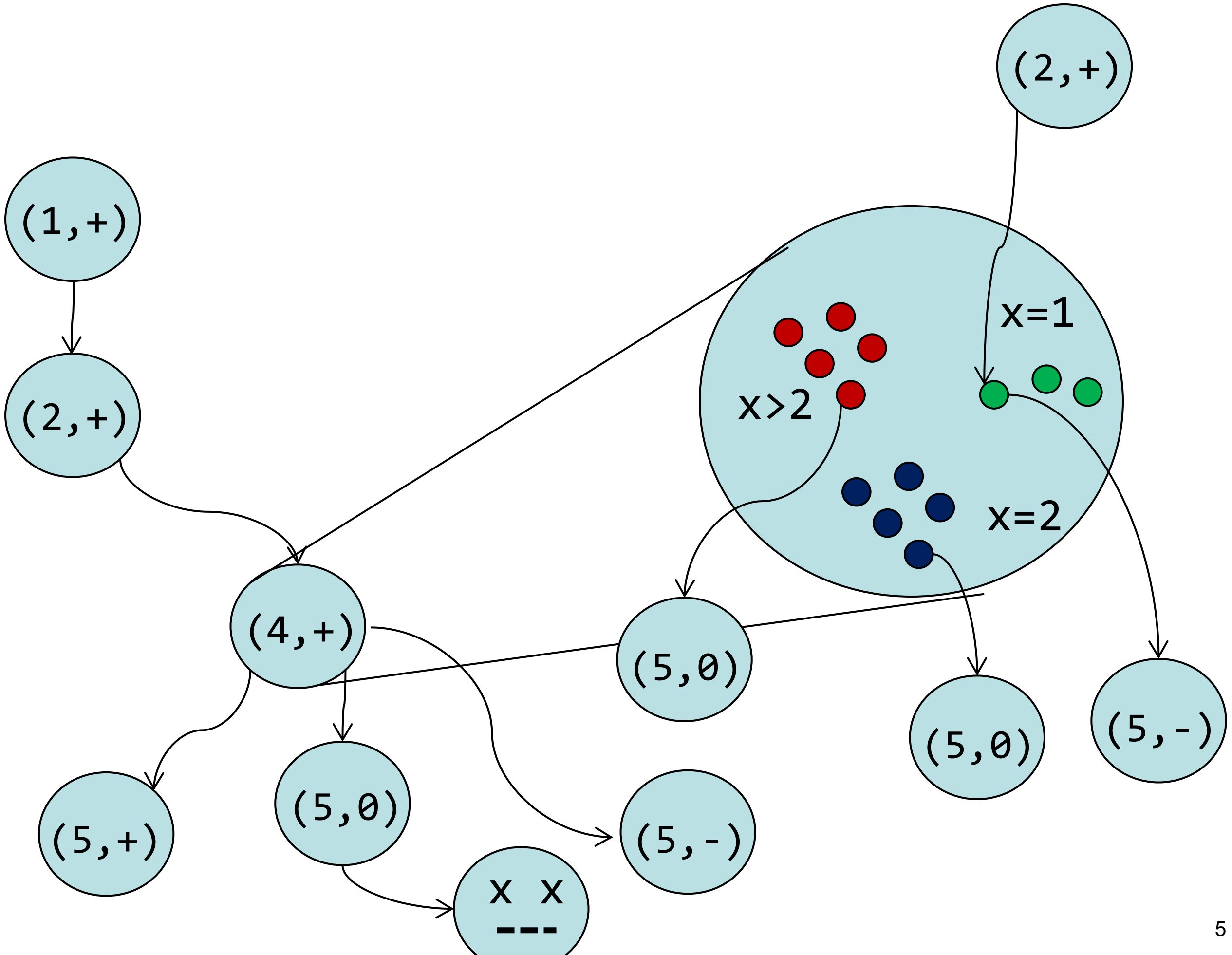
- Imprecision on the abstraction can lead to spurious paths

```
    void main(){
1:    int x = *;
      while(*){
2:      if(x>1)
3:        x = 2*x;
        else
4:        x = x-2;

5:      x = abs(*)/x;
      }
    }
```



Spurious path under the microscope

```
     void main(){
1:     int x = *;
       while(*){
2:       if(x>1)
3:         x = 2*x;
         else
4:         x = x-2;

5:       x = abs(*)/x;
       }
     }
```



2 Key ingredients for software MC

We need a simple way to come up with abstractions

Our abstractions must be flexible

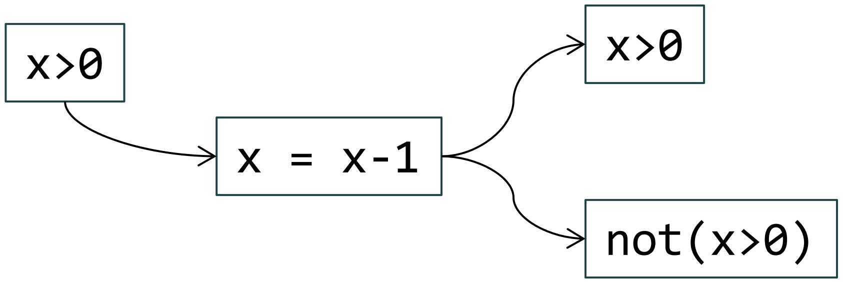
- We need to be able to refine them on demand
- This is how we identify spurious paths and eliminate them

Predicate Abstraction

Abstract state defined by a set of predicates

- Ex: x>0, p.next != null, p.next.val > 0

Transition function can be computed by a theorem prover

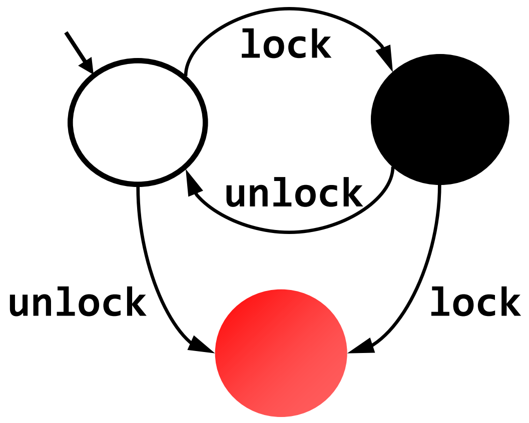


Big idea:

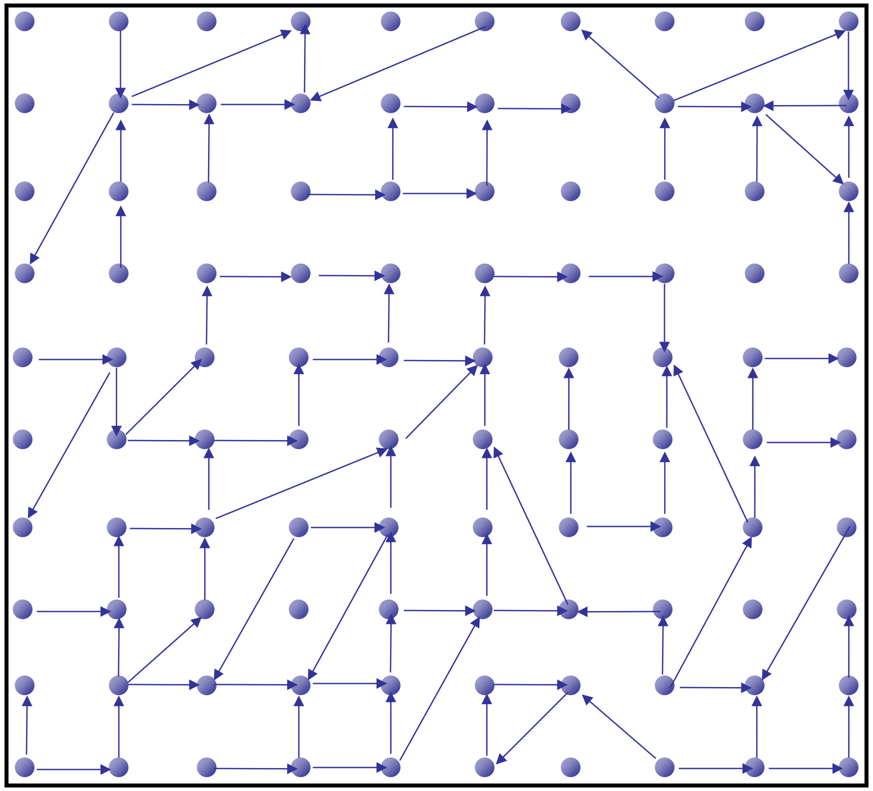
- We can refine the abstraction by introducing more predicates!

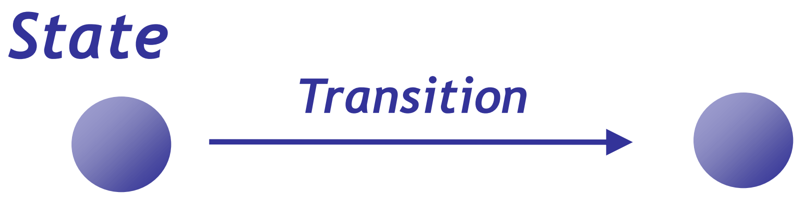
Example

```
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…



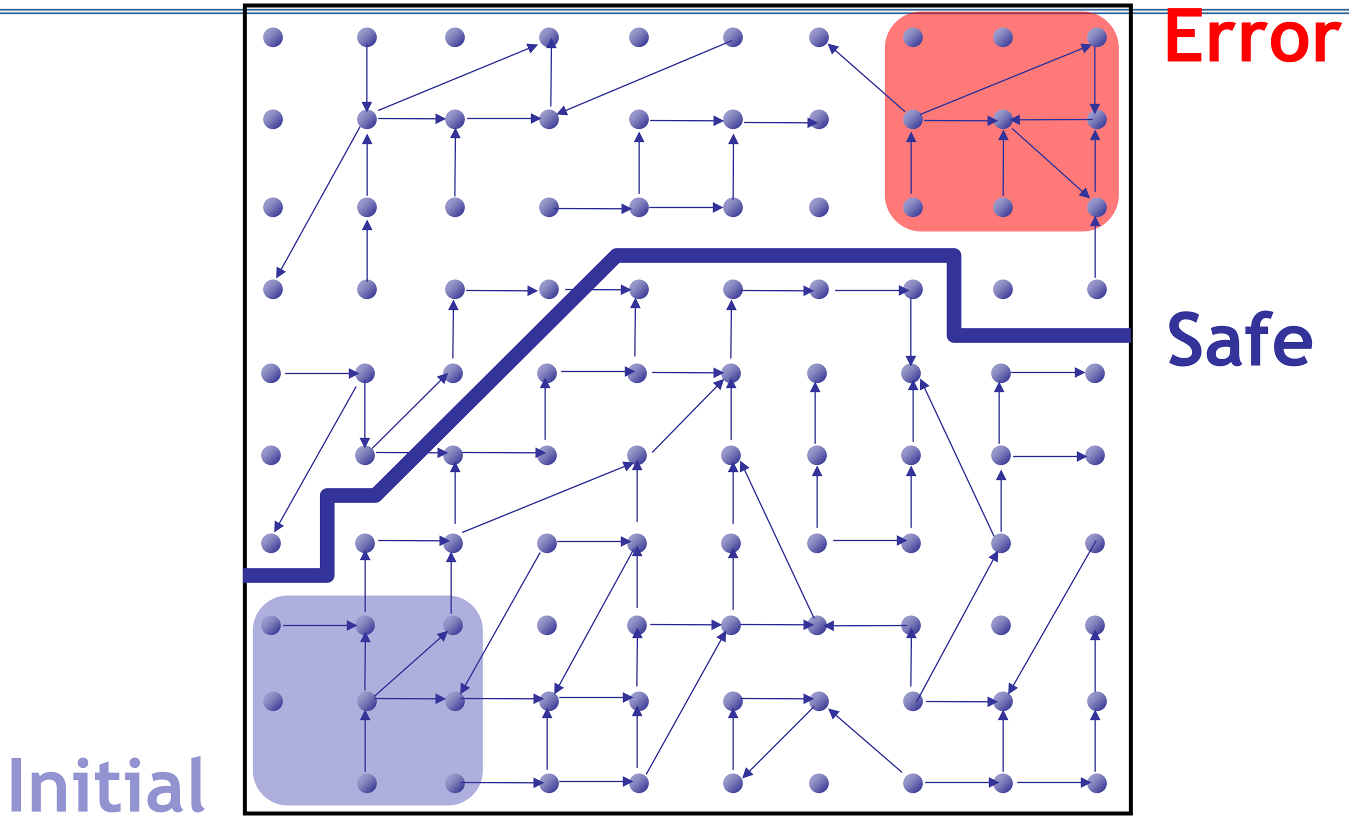


pc	↦ 3	*pc*	↦ 4
lock	↦ ●	lock	↦ ○
old	↦ 5	old	↦ 5
new	↦ 5	new	↦ 6
q	↦ 0x133a	q	↦ 0x133a

```
3: unlock();
   new++;
4:} …
```

```
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

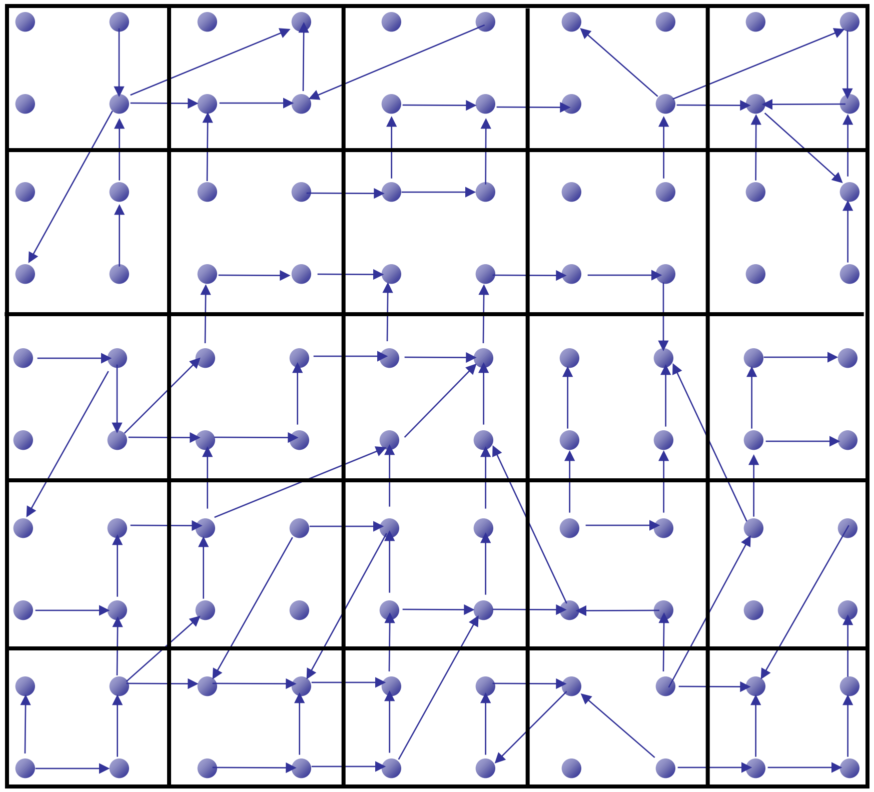


Is there a **path** from an **initial** to an **error** state ?

Problem: **Infinite** state graph

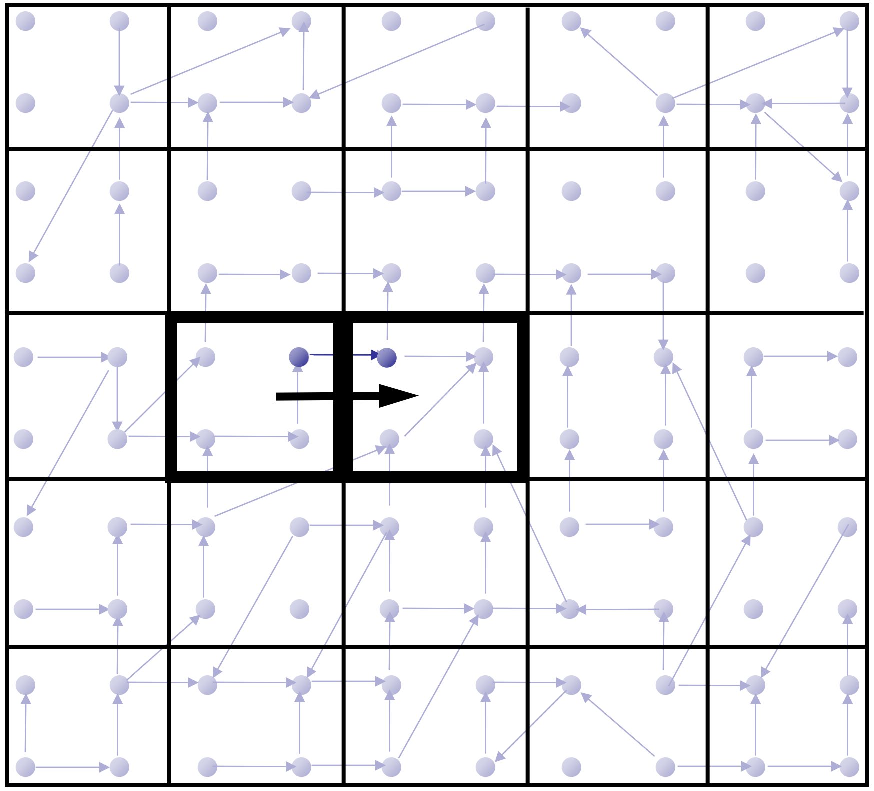
Solution : **Set** of states = logical **formula**

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

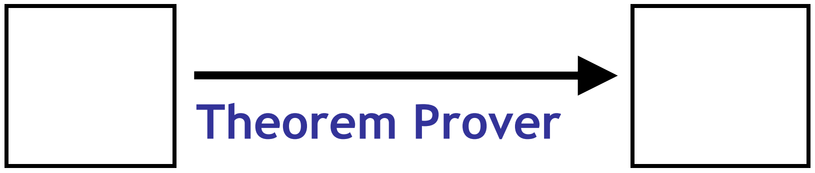


State

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

```
3: unlock();
   new++;
4:} …
```

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



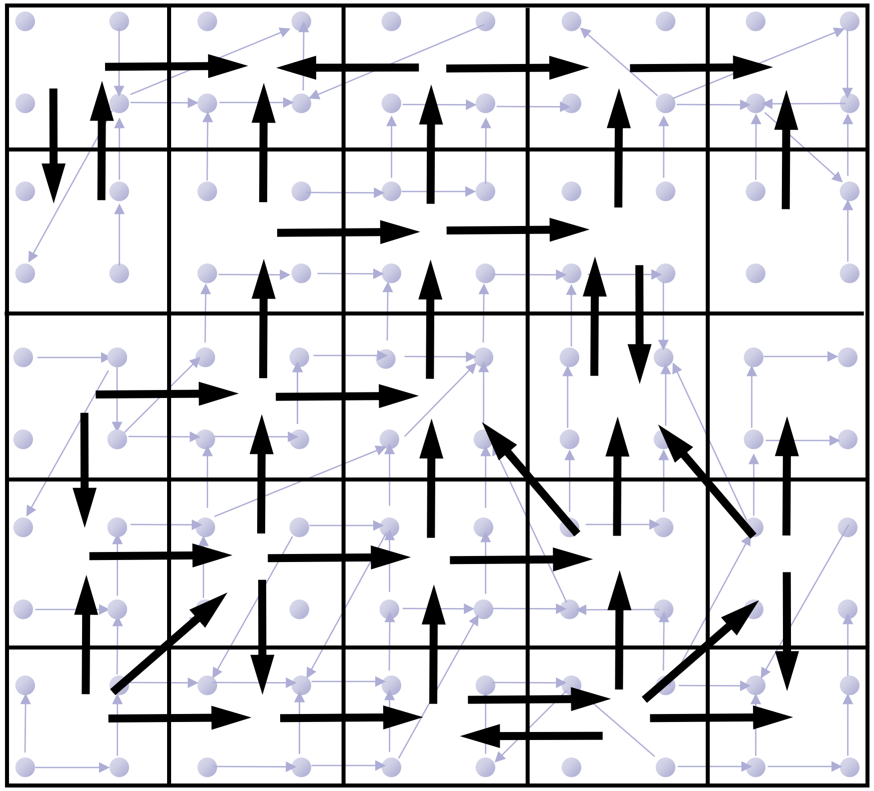
lock

old=new

!lock

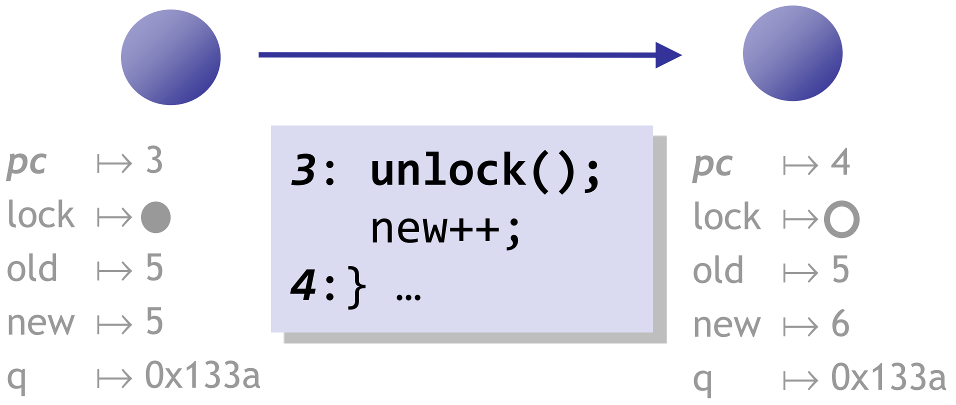
! old=new

Abstraction



Existential Lifting

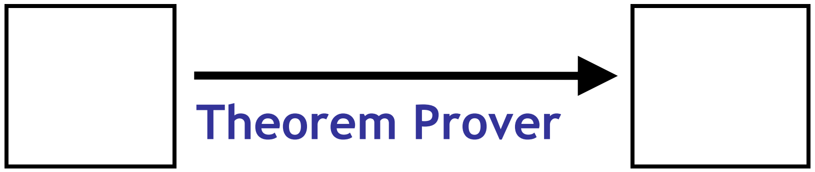
State



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

```
3: unlock();
   new++;
4:} …
```

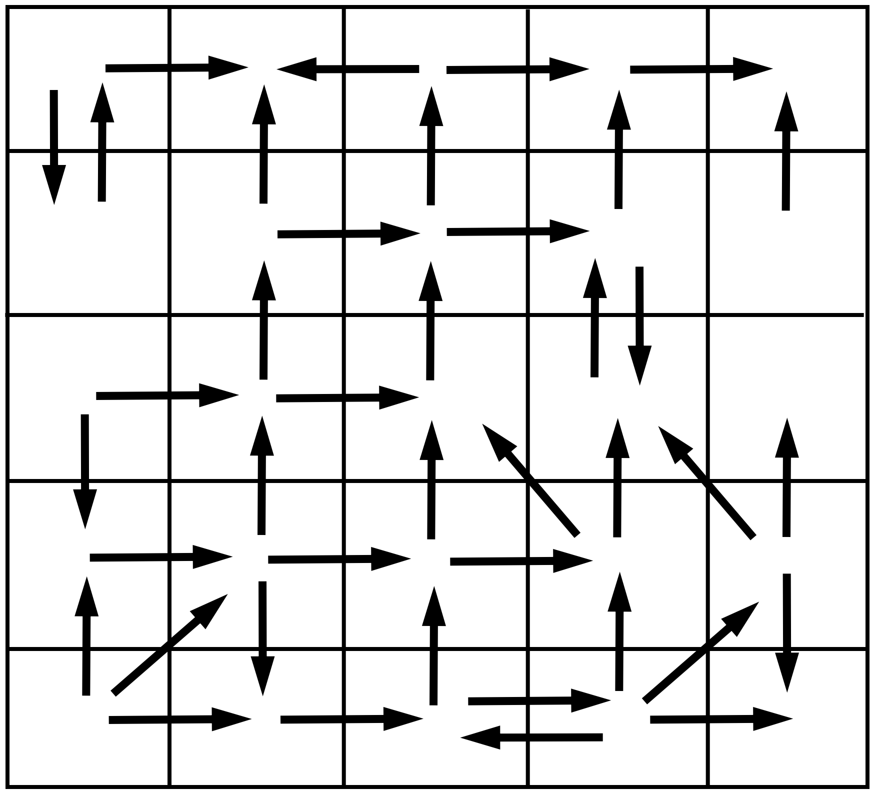
$pc \mapsto 4$
lock $\mapsto$ ○
old $\mapsto 5$
new $\mapsto 6$
q $\mapsto$ 0x133a

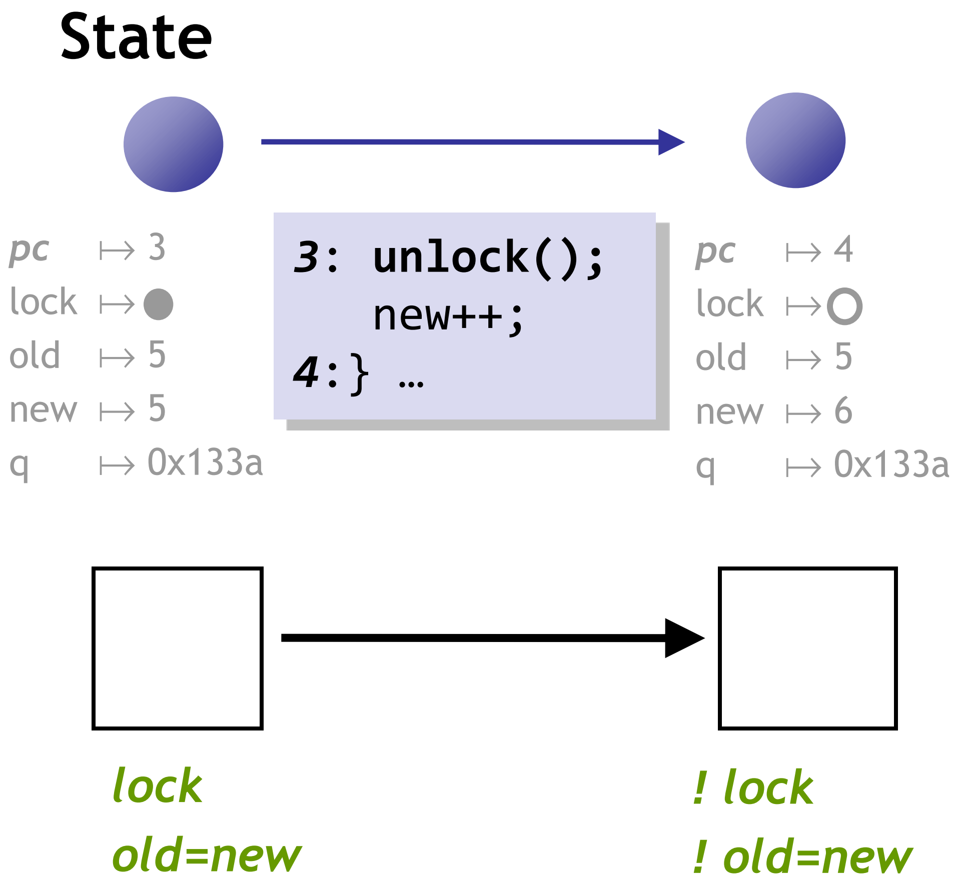


lock
old=new

! lock
! old=new

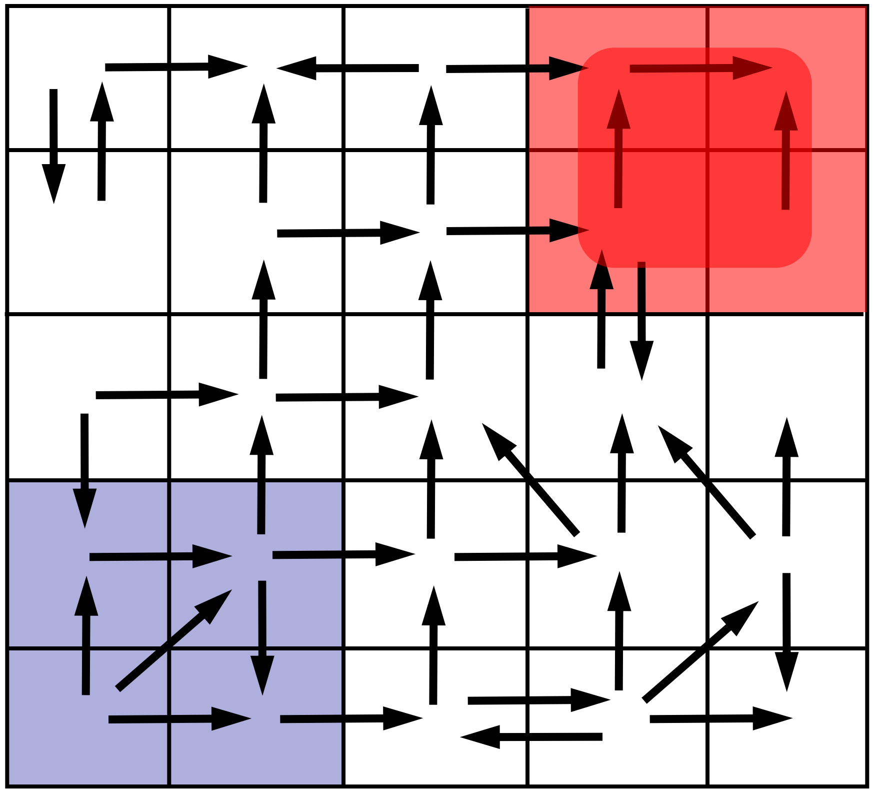
Abstraction





```
3: unlock();
   new++;
4:} …
```

Analyze Abstraction



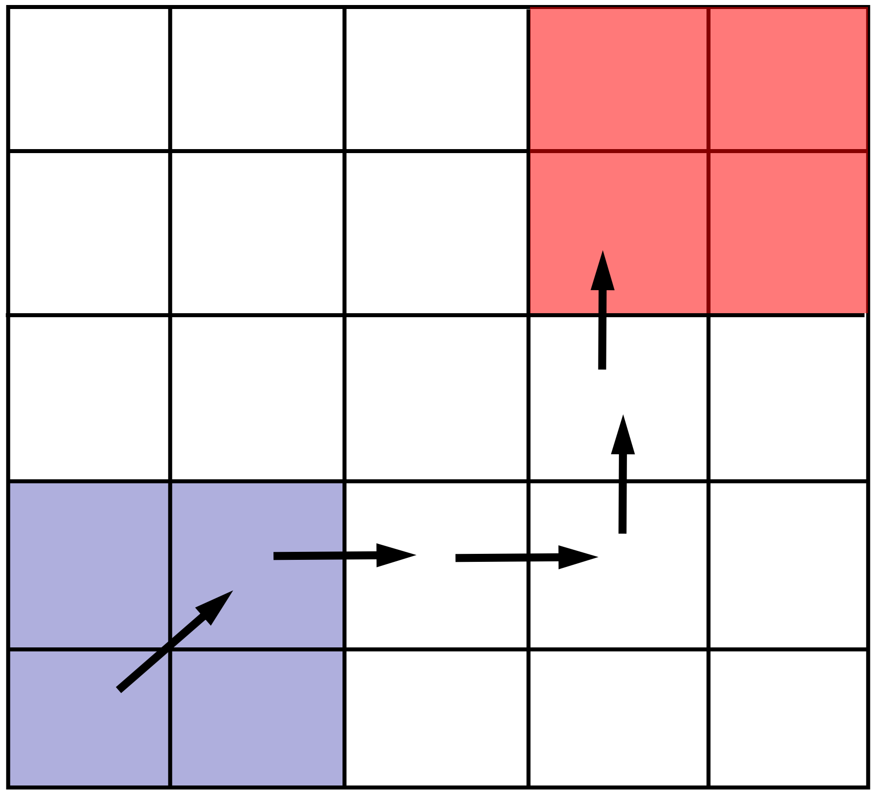
Analyze finite graph

No **false negatives**

Problem

Spurious **counterexamples**

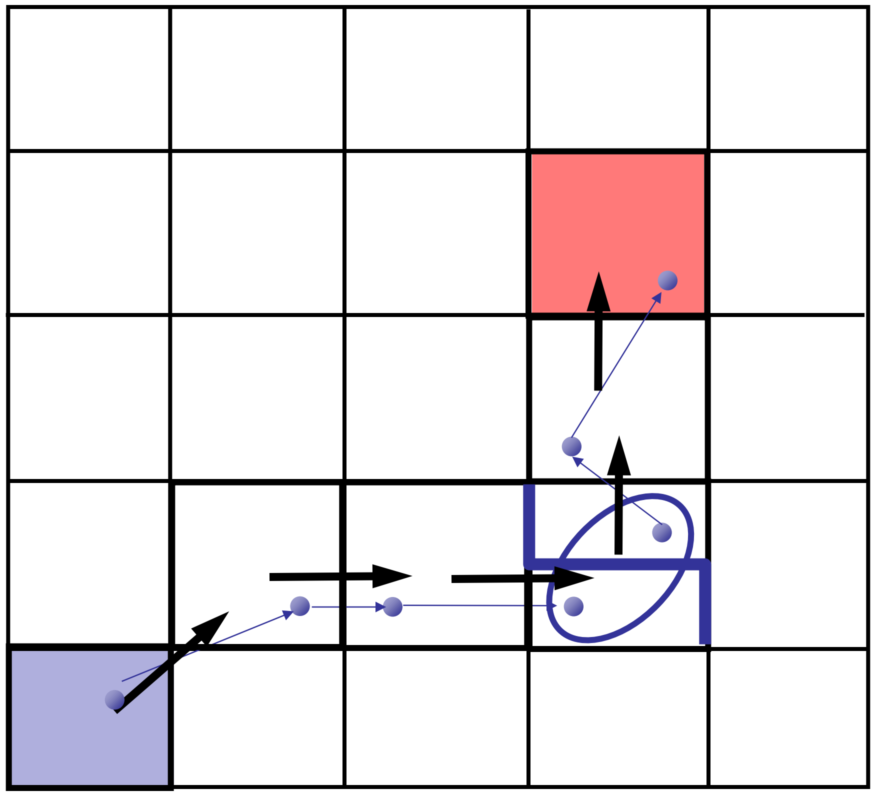
Idea 2: Counterex.-Guided Refinement



Solution

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

Idea 2: Counterex.-Guided Refinement



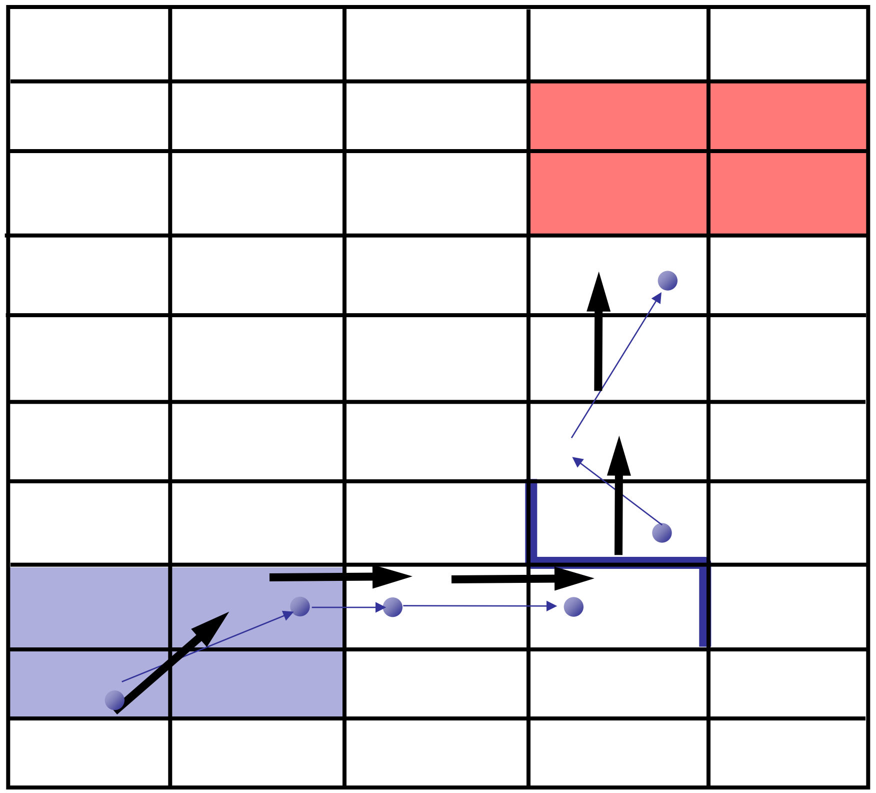
Solution

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]

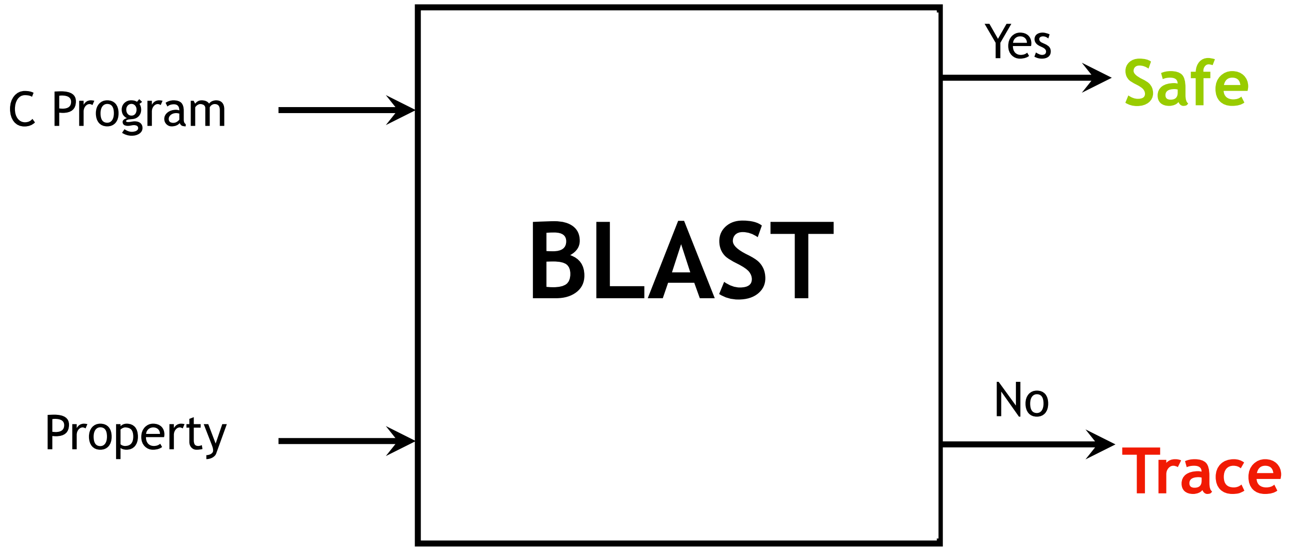
[Ball-Rajamani 01]

Solution

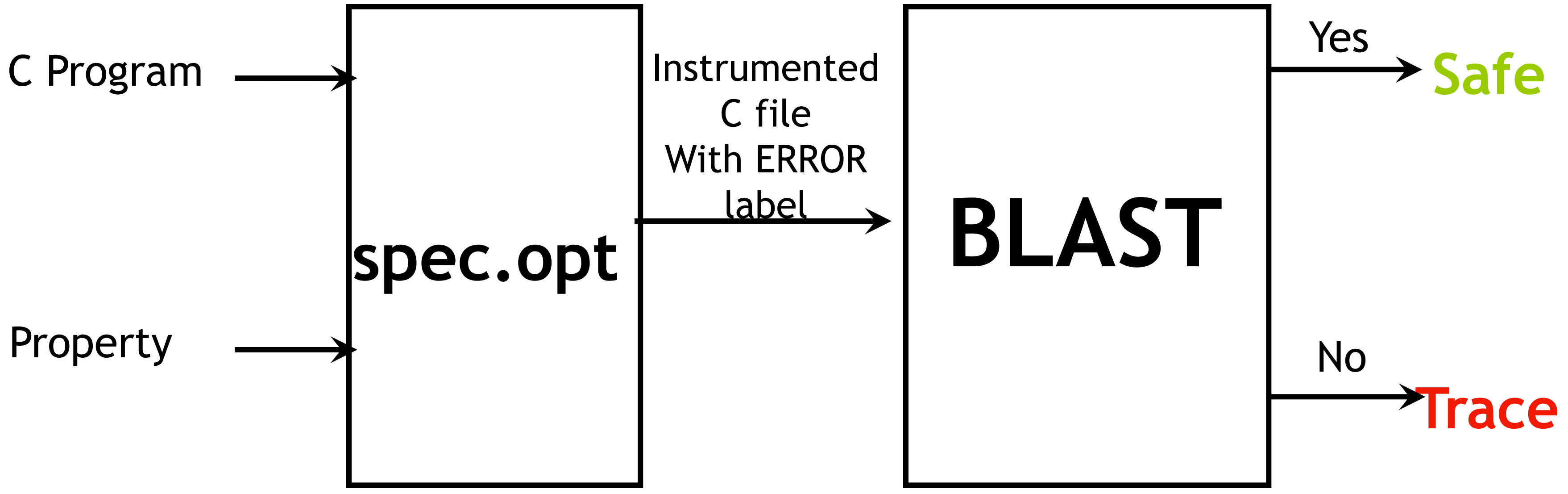
Use spurious **counterexamples** to **refine** abstraction

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

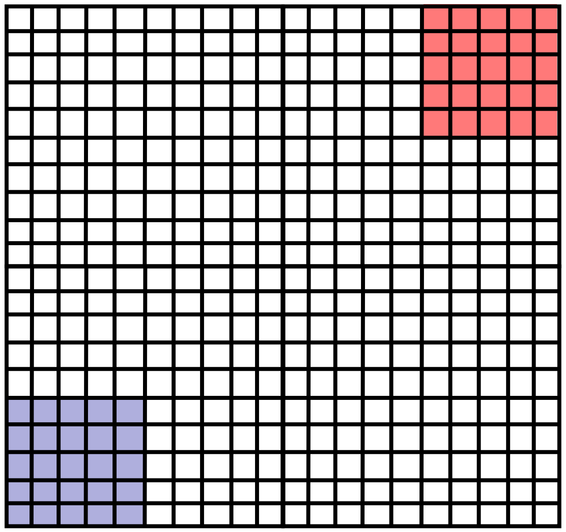
Lazy Abstraction

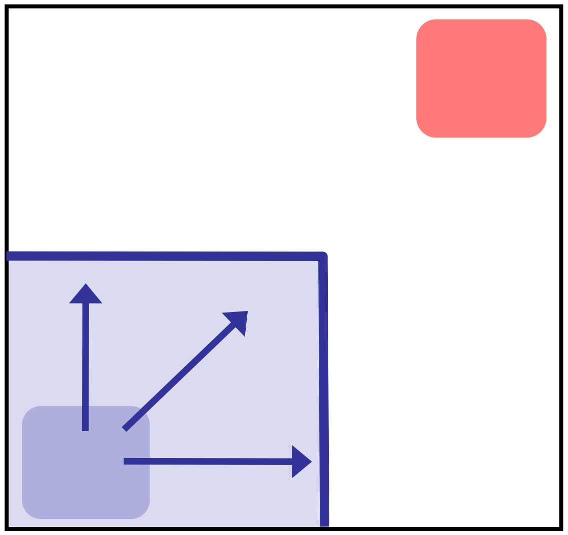


Lazy Abstraction



Problem: Abstraction is Expensive





Reachable

Problem

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

Exponential Thm. Prover queries

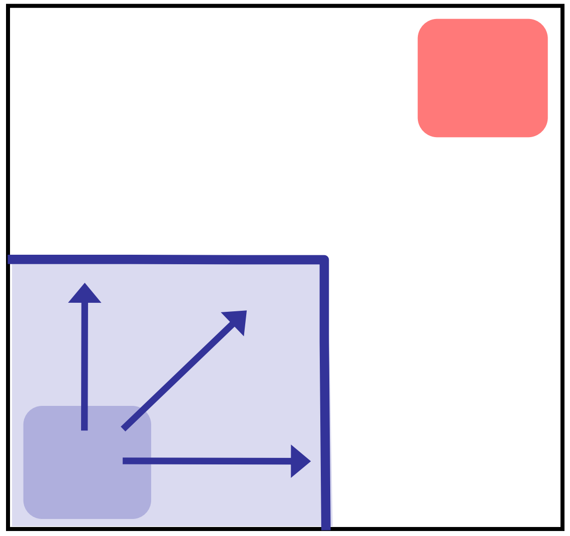
Observe

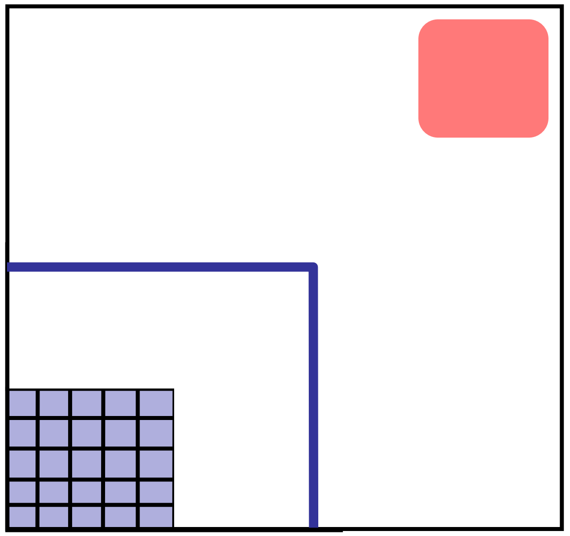
Fraction of state space reachable

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

#Reach ~ 1000's

Solution1: Only Abstract Reachable States





Problem

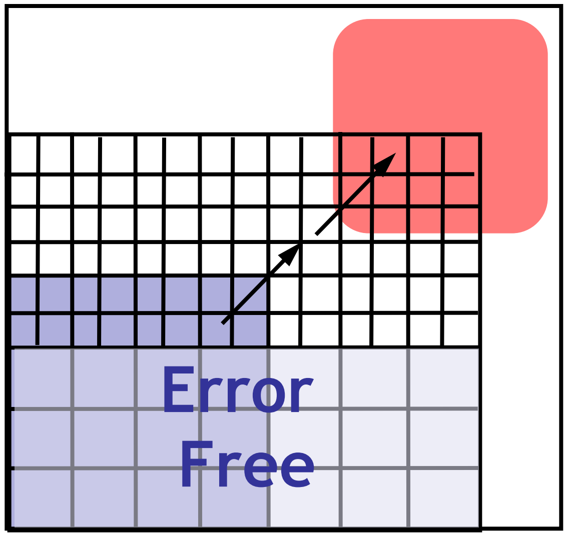
#abstract states = $2^{\#predicates}$

Exponential Thm. Prover queries

Solution

Build abstraction **during** search

Solution2: Don’t Refine Error-Free Regions



Problem

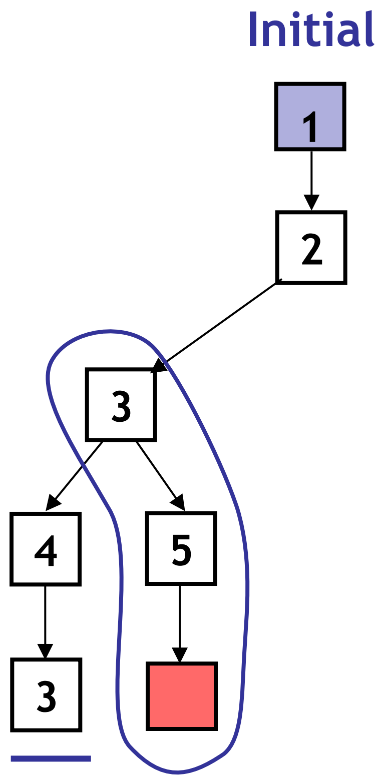
#abstract states = $2^{\#\text{predicates}}$

Exponential Thm. Prover queries

Solution

Don’t refine error-free regions

Key Idea: Reachability Tree



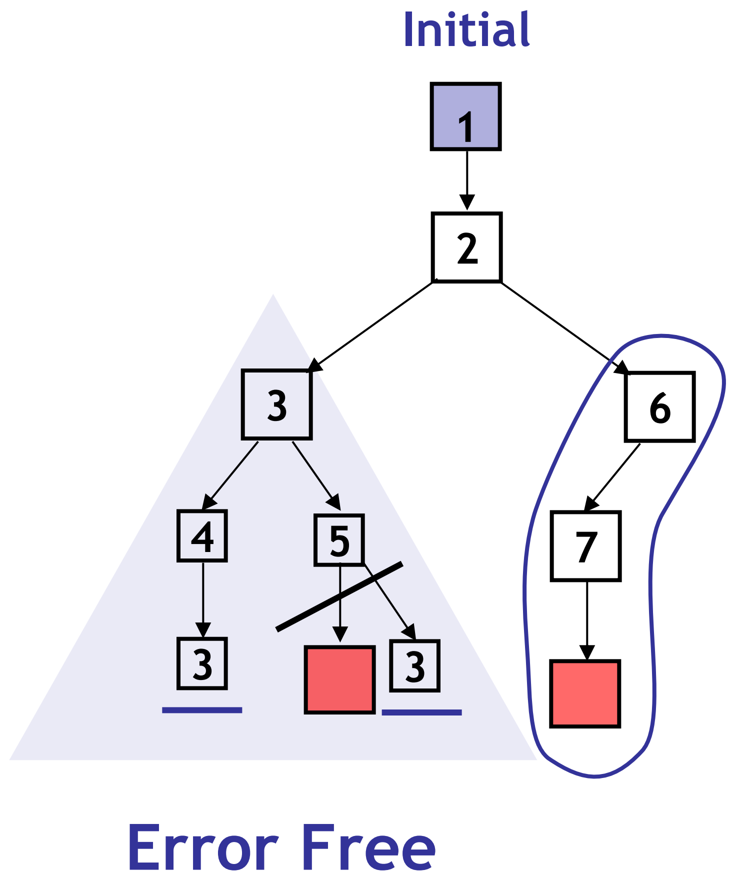
Unroll Abstraction

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



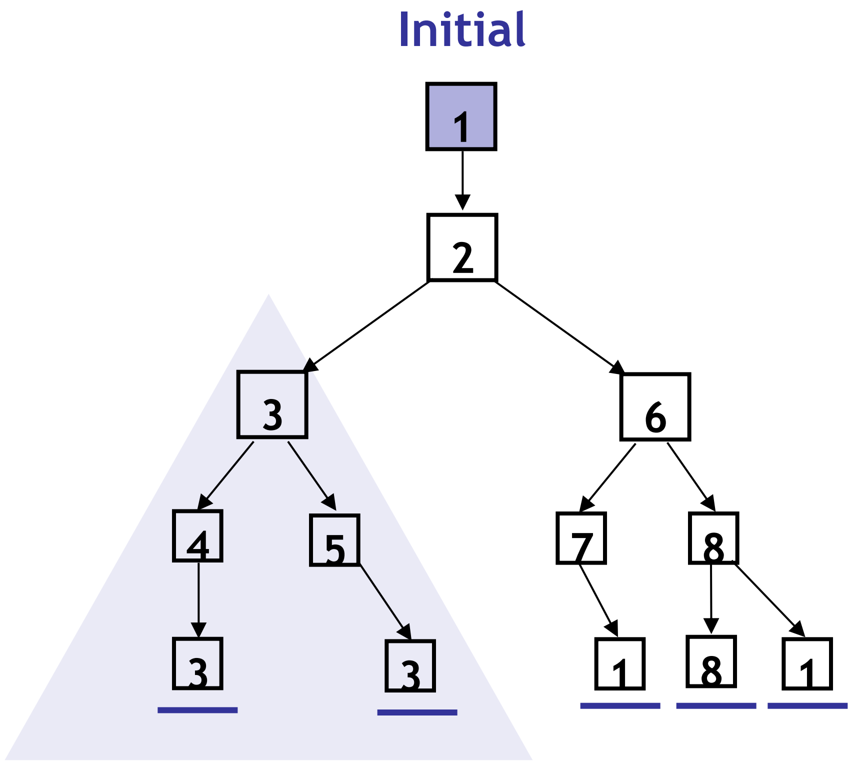
Unroll Abstraction

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.

Error Free

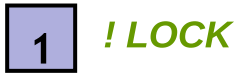
$\boxed{\text{SAFE}}$

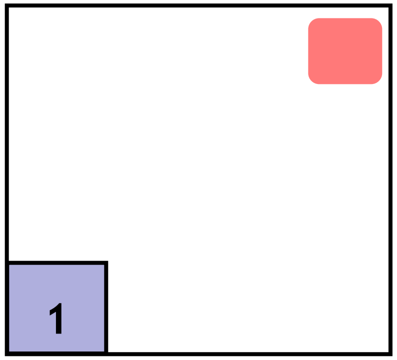
S1: Only Abstract Reachable States

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 ();
}
```



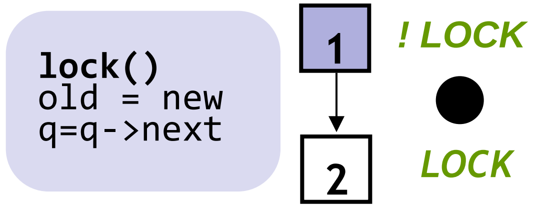


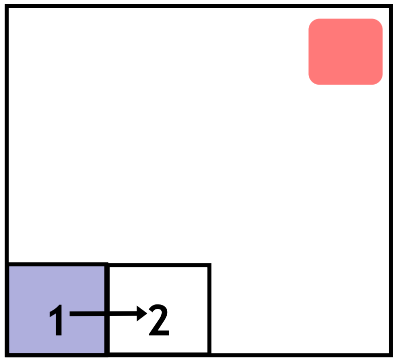
Predicates: *LOCK*

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 ();
}
```



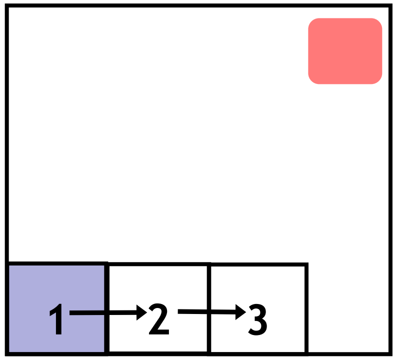


Predicates: *LOCK*

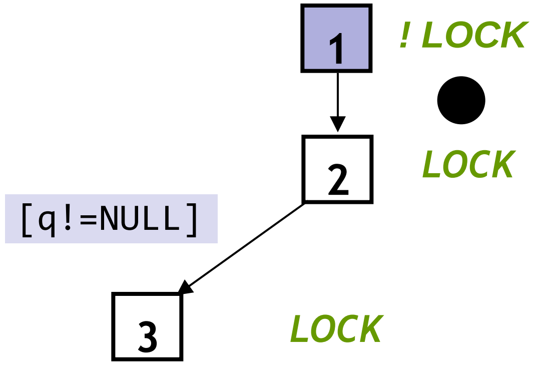
Reachability Tree

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 ();
}
```



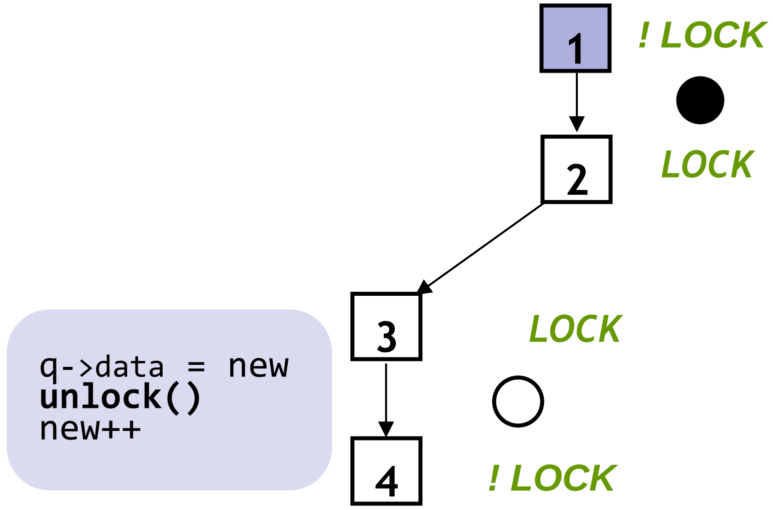
Predicates: *LOCK*

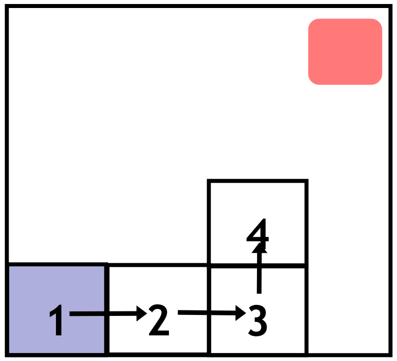


Reachability Tree

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 ();
}
```



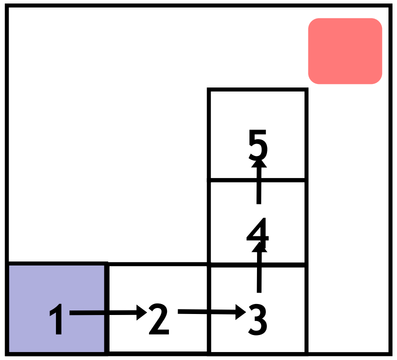


Predicates: *LOCK*

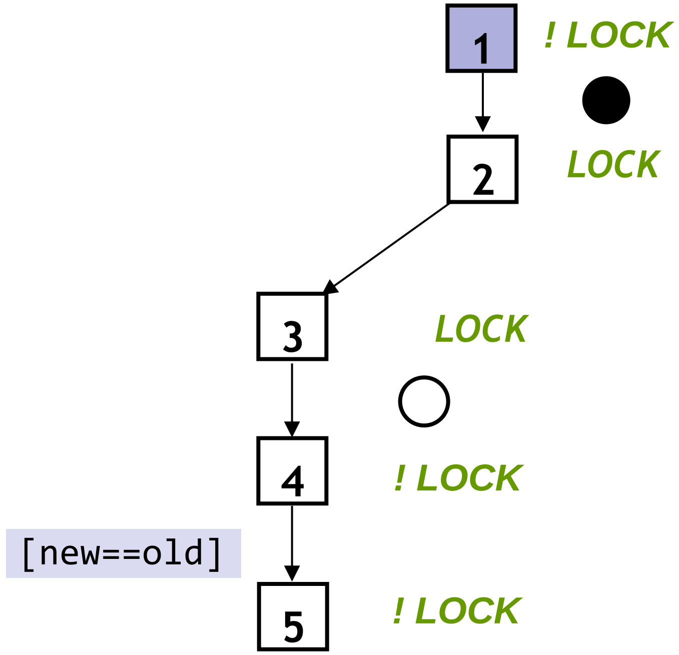
Reachability Tree

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 ();
}
```



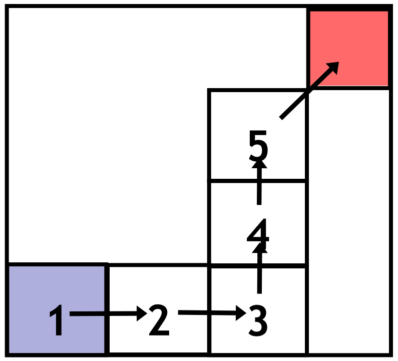
Predicates: *LOCK*



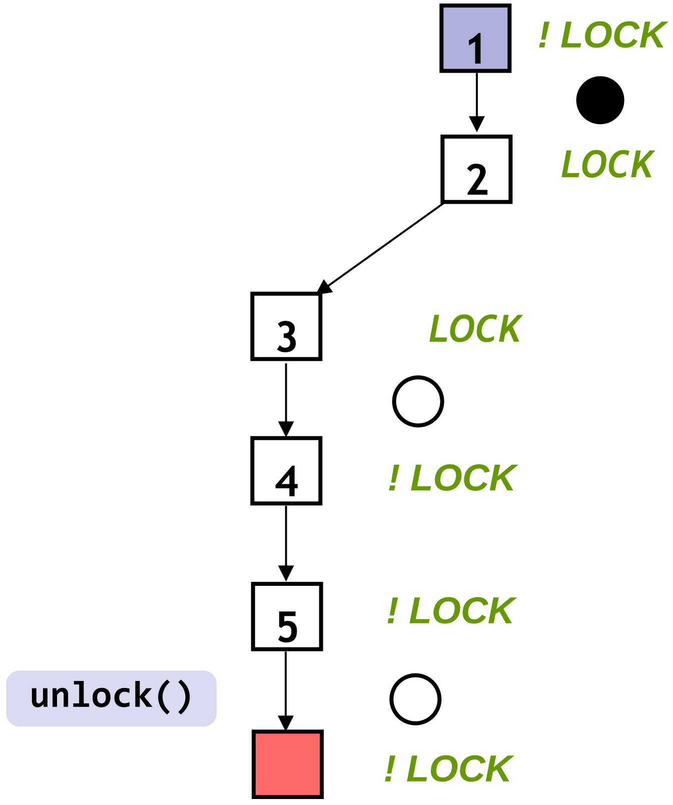
Reachability Tree

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 ();
}
```



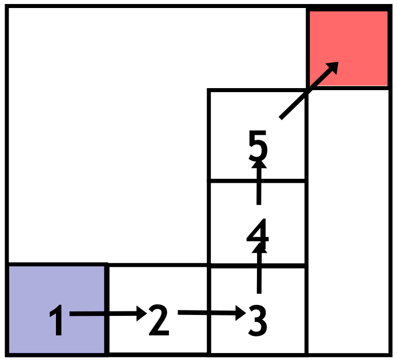
Predicates: *LOCK*



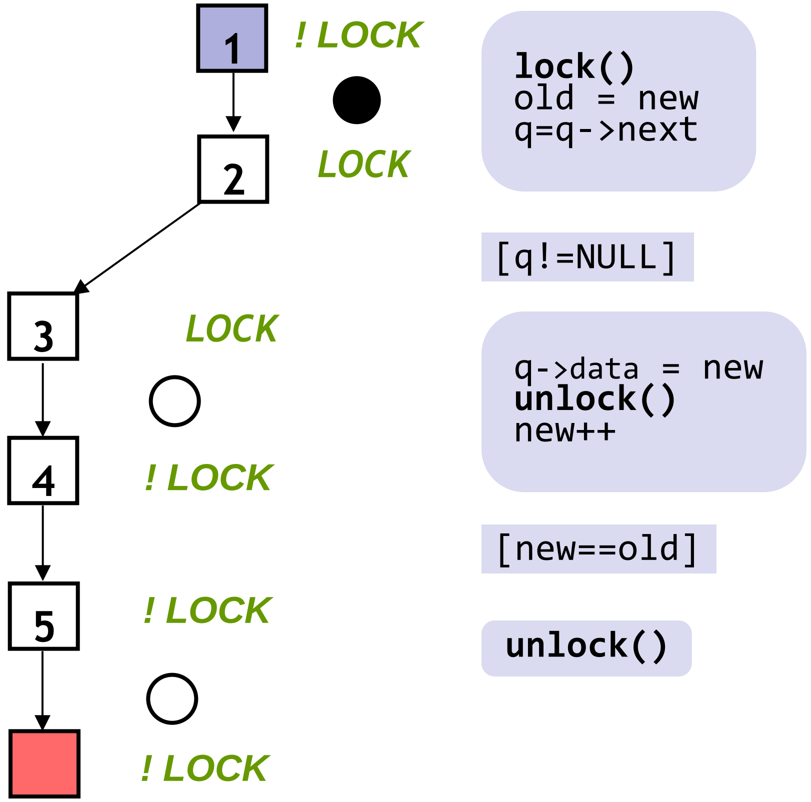
Reachability Tree

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 ();
}
```



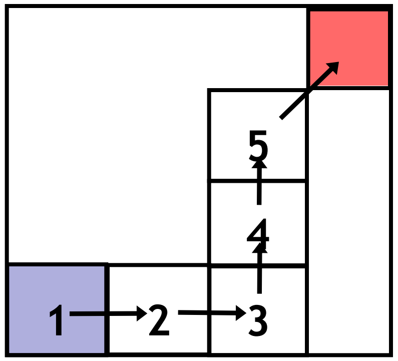
Predicates: *LOCK*



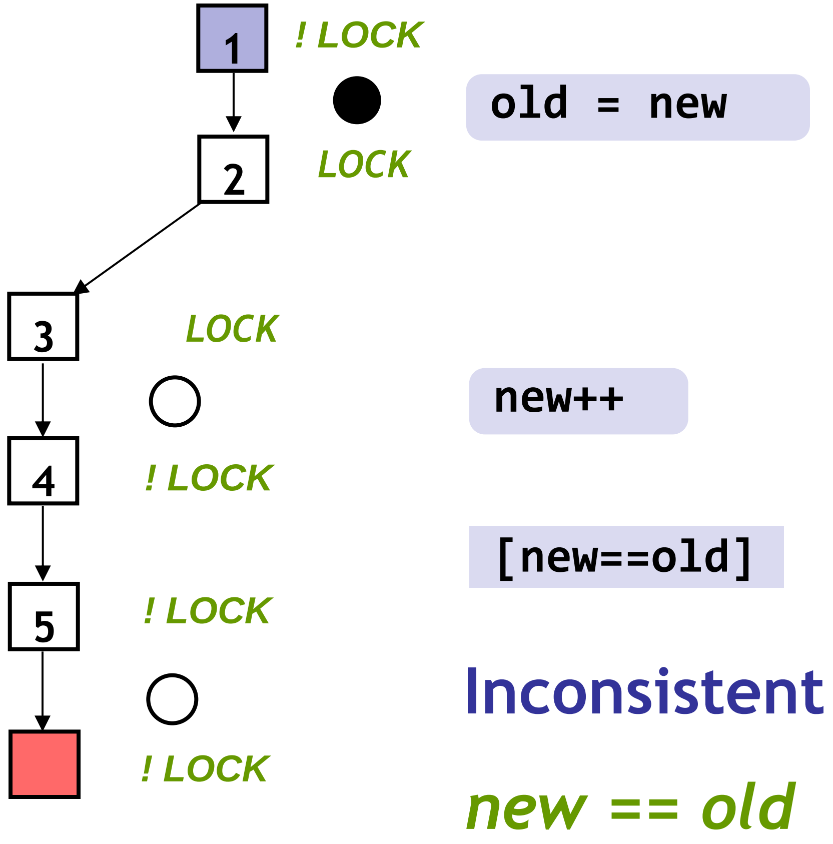
Reachability Tree

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 ();
}
```



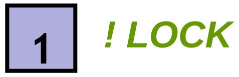
Predicates: *LOCK*

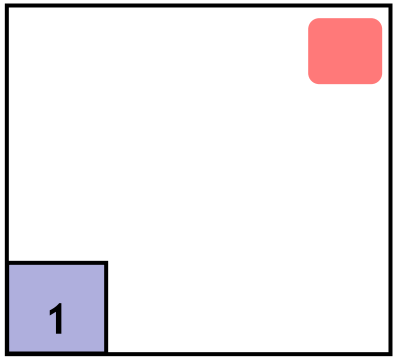


Reachability Tree

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 ();
}
```



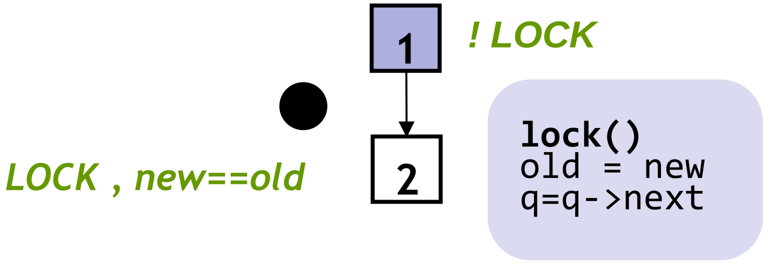


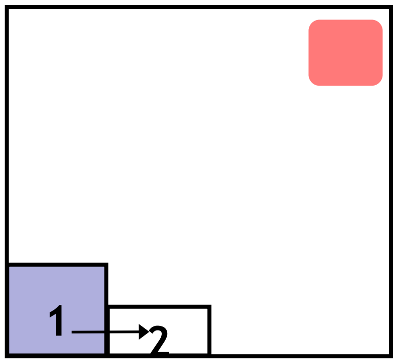
Predicates: *LOCK, new==old*

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 ();
}
```



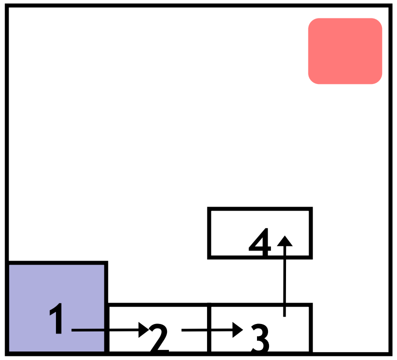


Predicates: *LOCK, new==old*

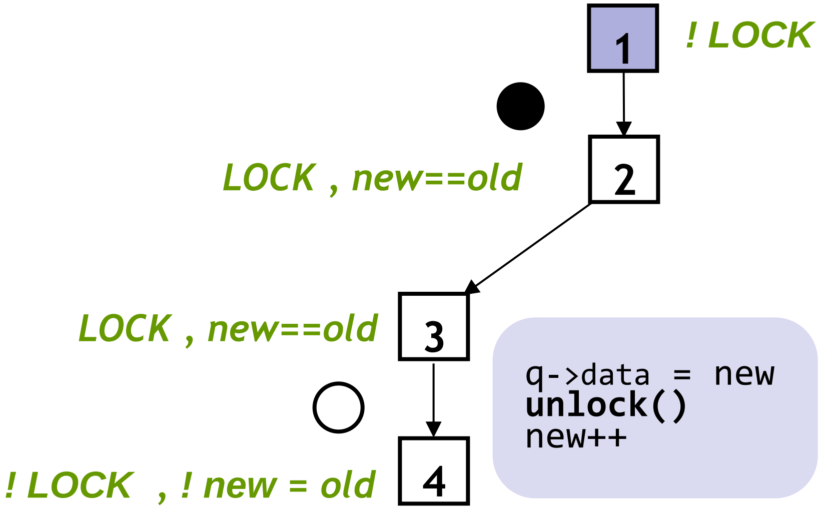
Reachability Tree

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 ();
}
```



Predicates: *LOCK, new==old*

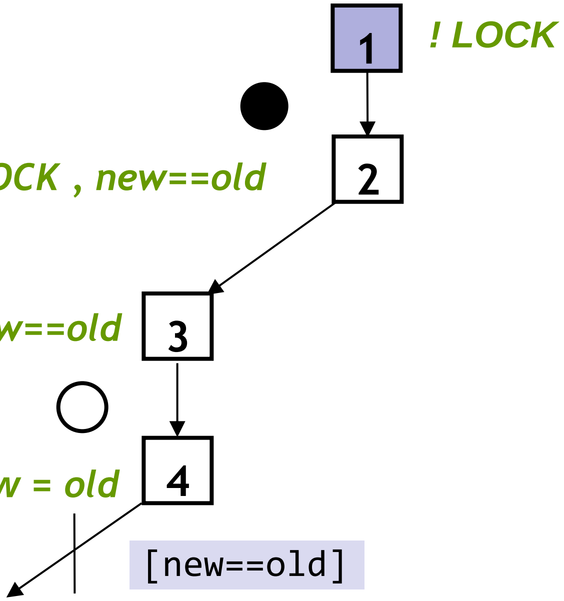


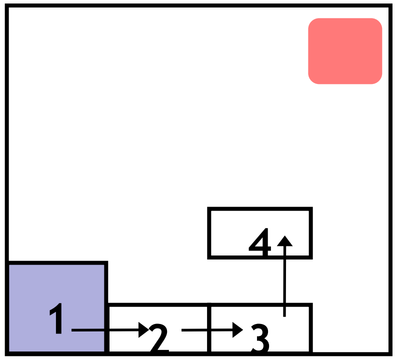
```
q->data = new
unlock()
new++
```

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 ();
}
```



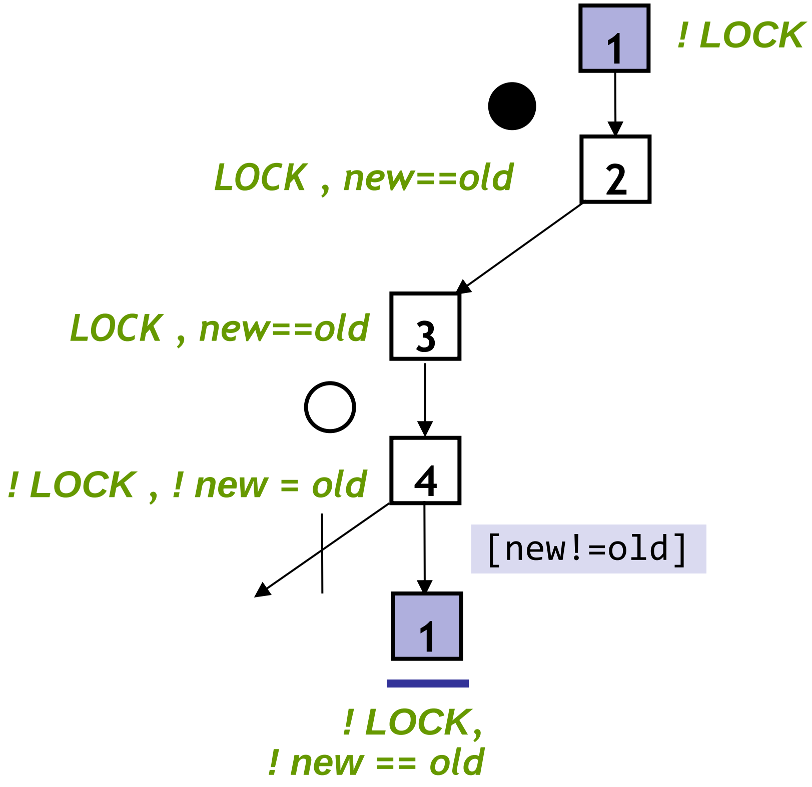


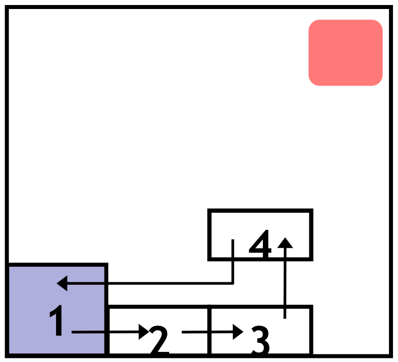
Predicates: *LOCK, new==old*

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 ();
}
```



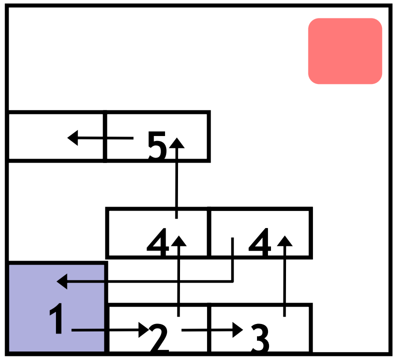


Predicates: *LOCK, new==old*

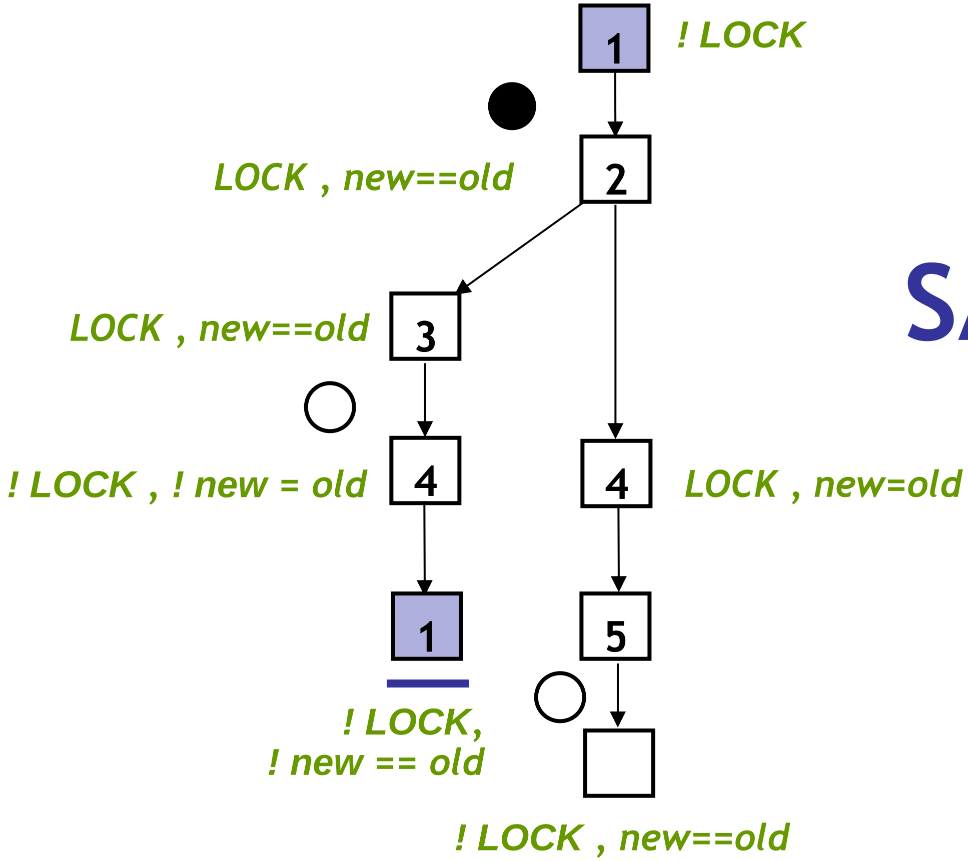
Reachability Tree

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 ();
}
```



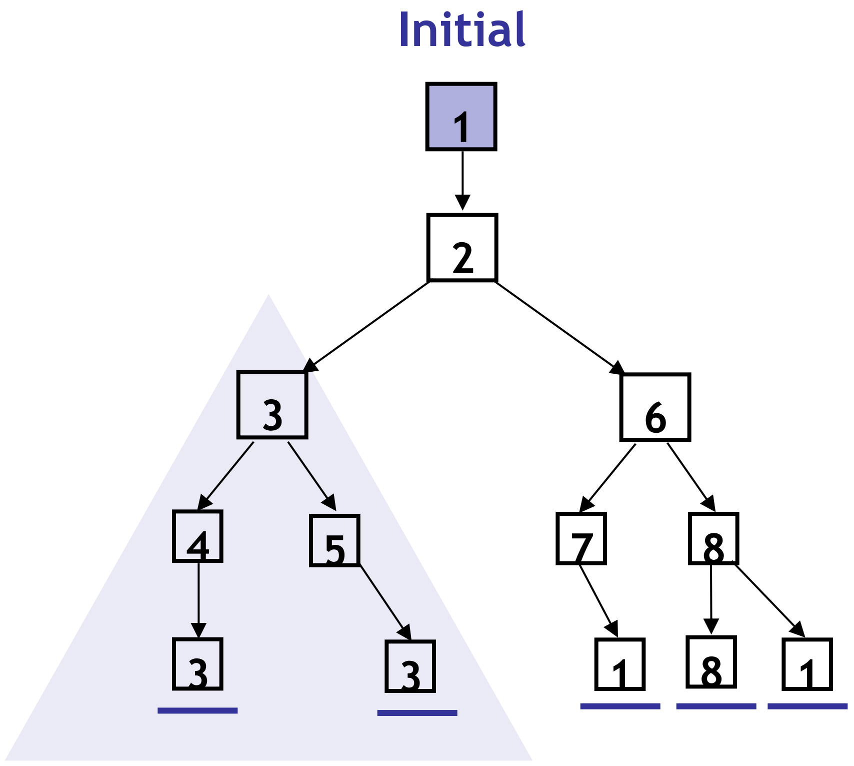
Predicates: *LOCK, new==old*



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.

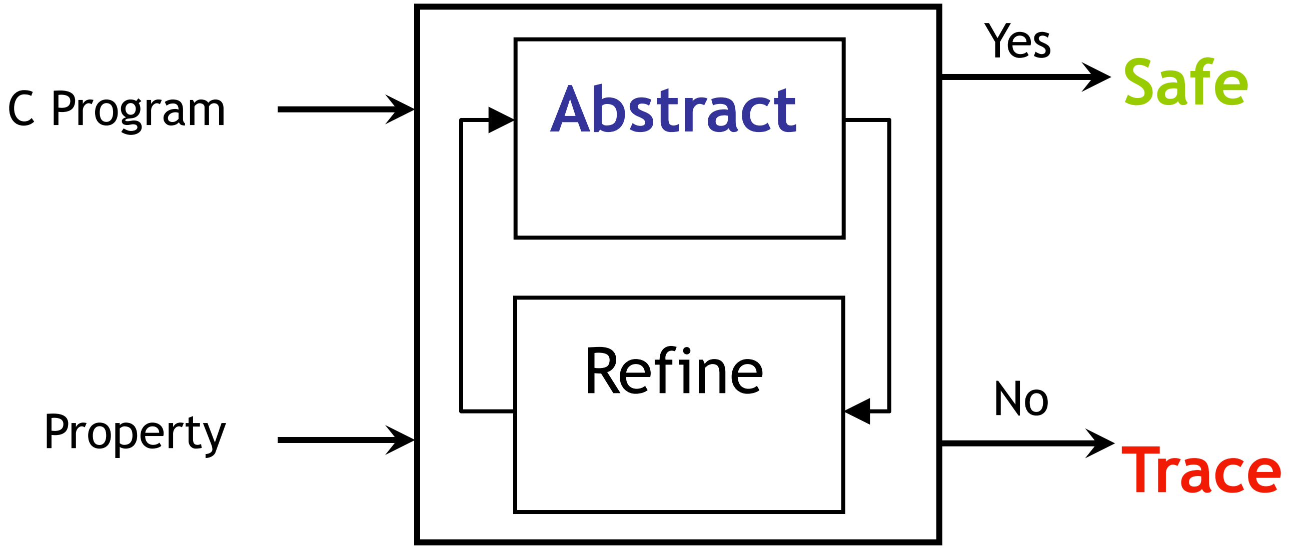
Error Free

SAFE

S1: Only Abstract Reachable States

S2: Don't refine error-free regions

Lazy Abstraction



Problem: Abstraction is Expensive

Solution: 1. Abstract reachable states,
2. Avoid refining error-free regions

Key Idea: Reachability Tree