Predicate abstraction and interpolation

Many pictures and examples are borrowed from

The Software Model Checker

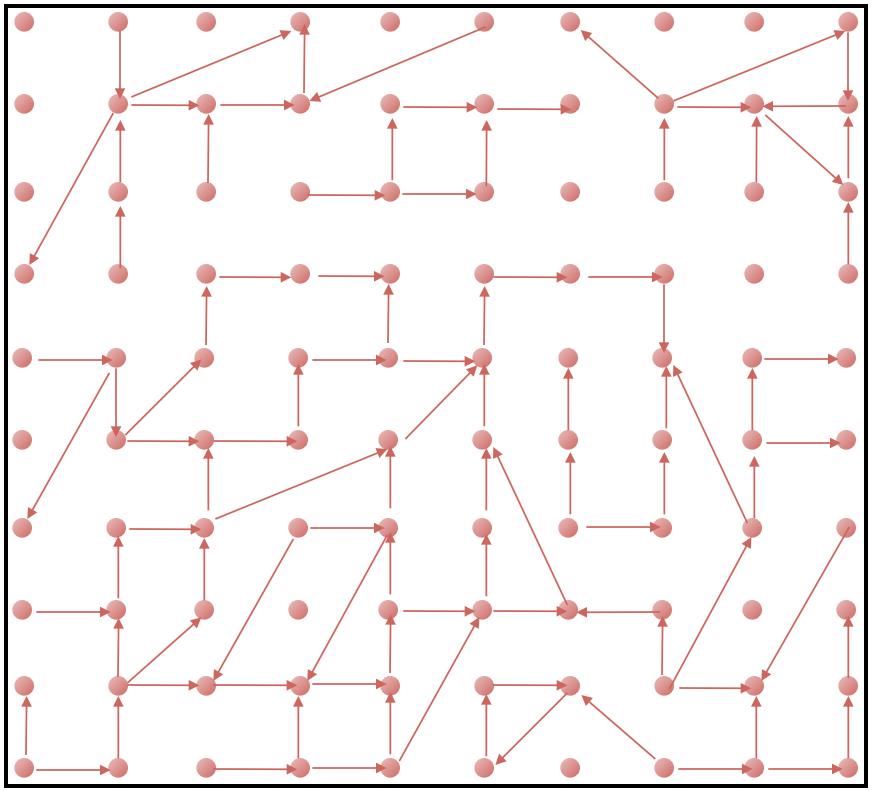
BLAST

presentation.

Outline

1. Predicate abstraction – the idea in pictures
2. Counter-example guided refinement
3. wp, sp for predicate discovery
4. Interpolation

The task



Given a program, we wish to check it satisfies some property:

- never divide by zero
- variable x is always positive
- never call lock() twice in a row
- …

The problem:

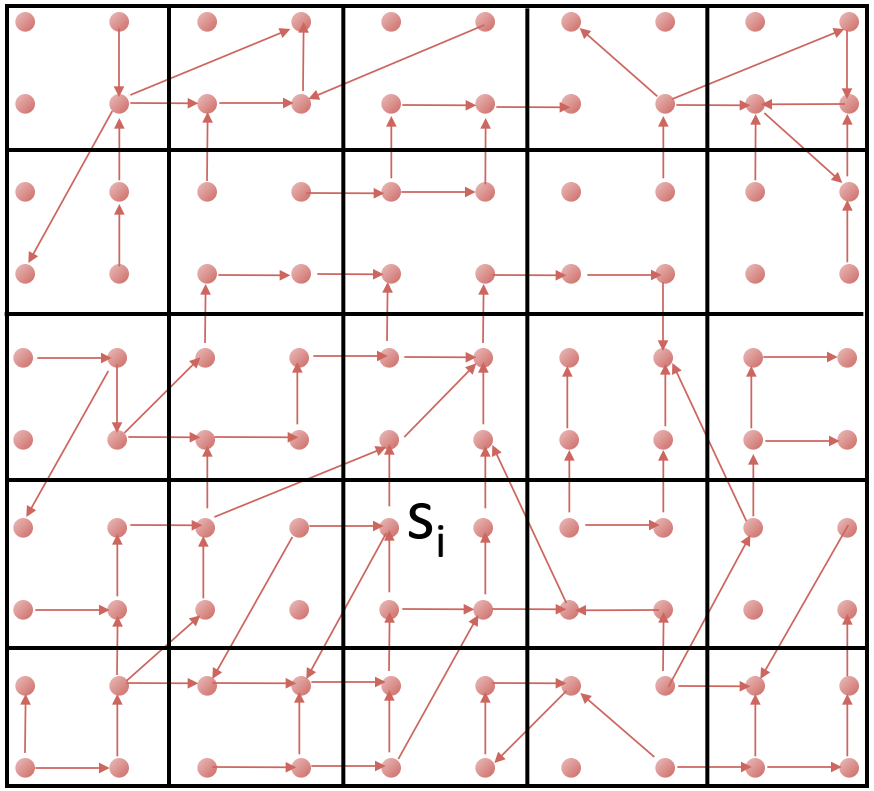
In general, programs have large or (for practical purposes) infinite state spaces:

- many variables
- integers

Let's say we have two 32bit integer variables, the number of states is

18446744065119617025

Predicate abstraction



Group concrete states that satisfy a certain property together.

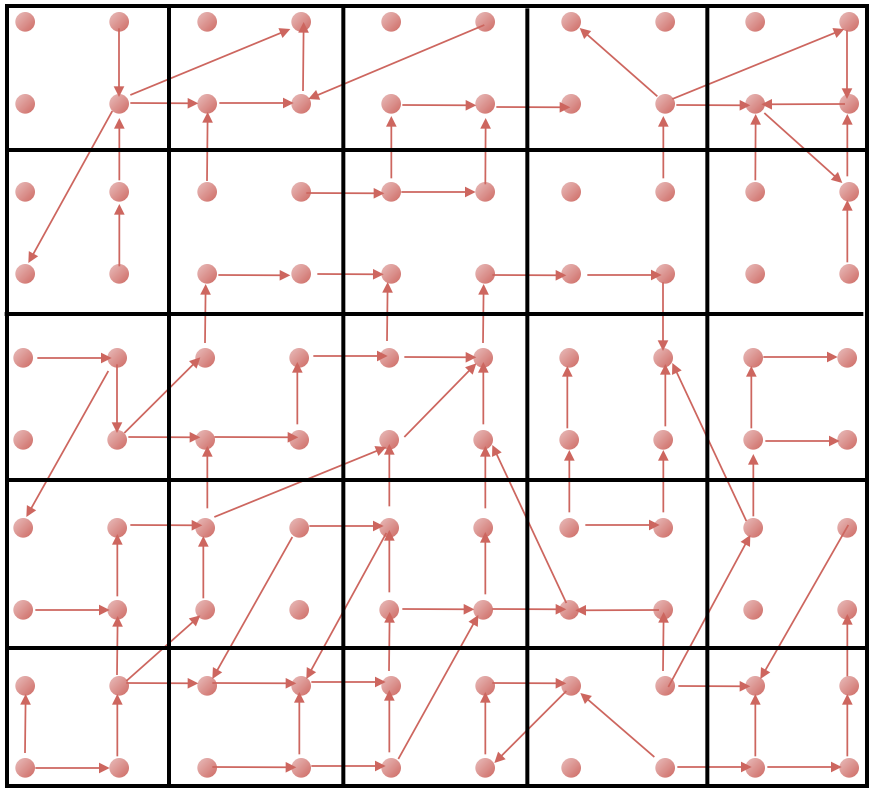
→ finitely many abstract states, labeled by predicates

Each such concrete state $\mathbf{s}_i$ consists of

- program counter
- state of variables

Hence, one node in the CFG can correspond to many different program states.

Predicate abstraction



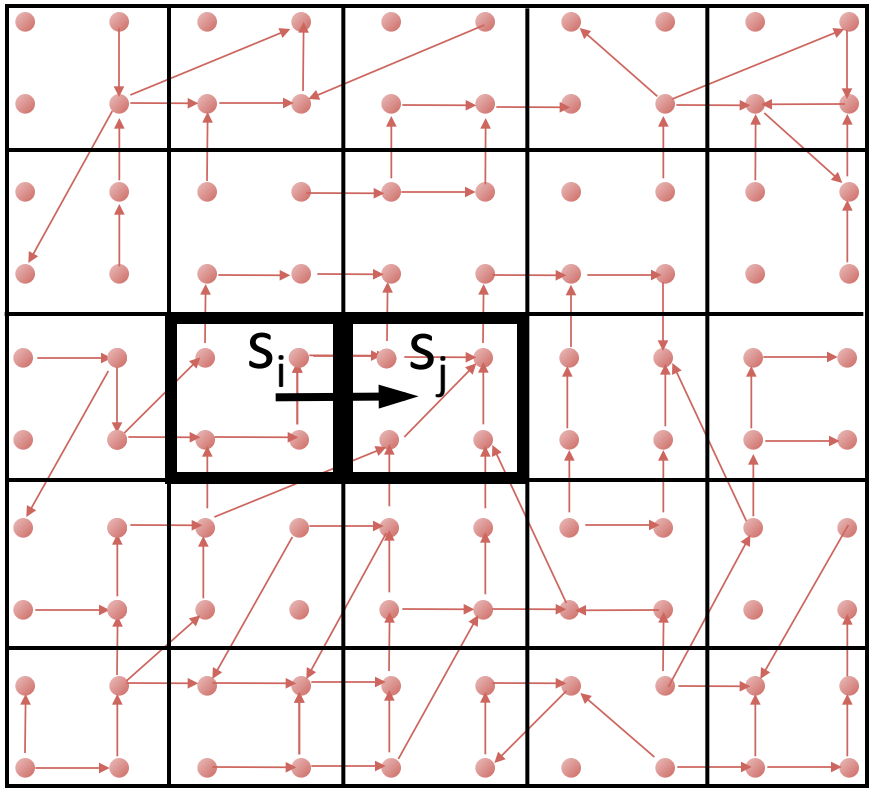
We are given the concrete relation with transitions $(s_i, s_j) \in r$, i.e whenever we have an edge in the CFG.

Using some abstraction function β we get corresponding abstract states

$a_i = \beta(s_i)$ and $a_j = \beta(s_j)$

and we merge those states whose predicates are the same.

Predicate abstraction



$a_i = \beta(s_i)$ and $a_j = \beta(s_j)$

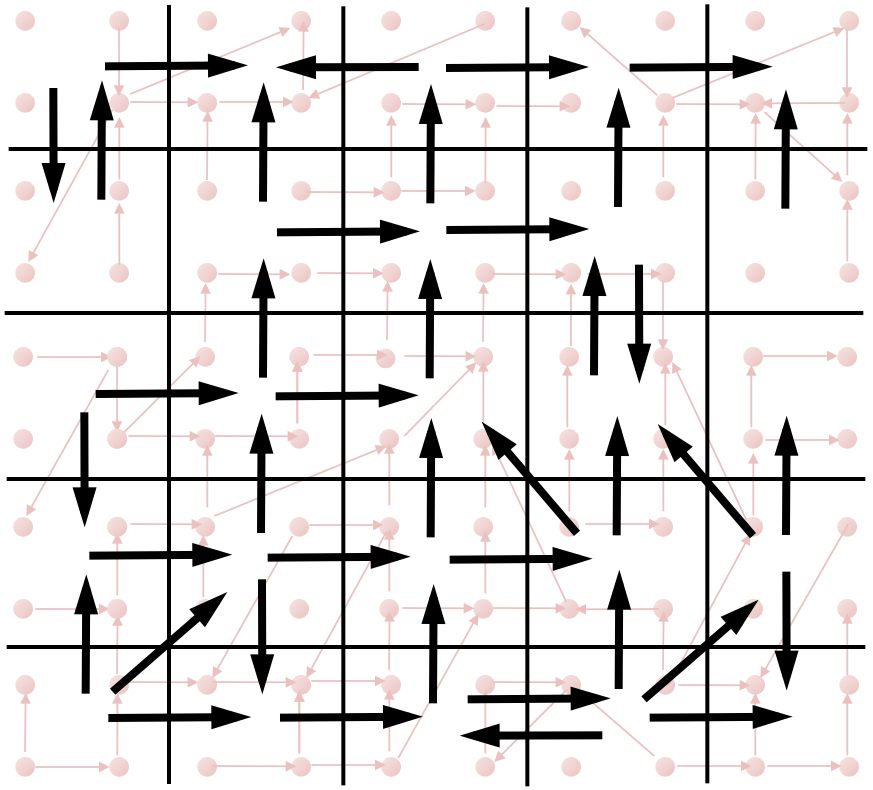
Then, if

$(s_i, s_j) \in r$

we require

$(a_i, a_j) \in a$.

Predicate abstraction



$a_i = \beta(s_i)$ and $a_j = \beta(s_j)$

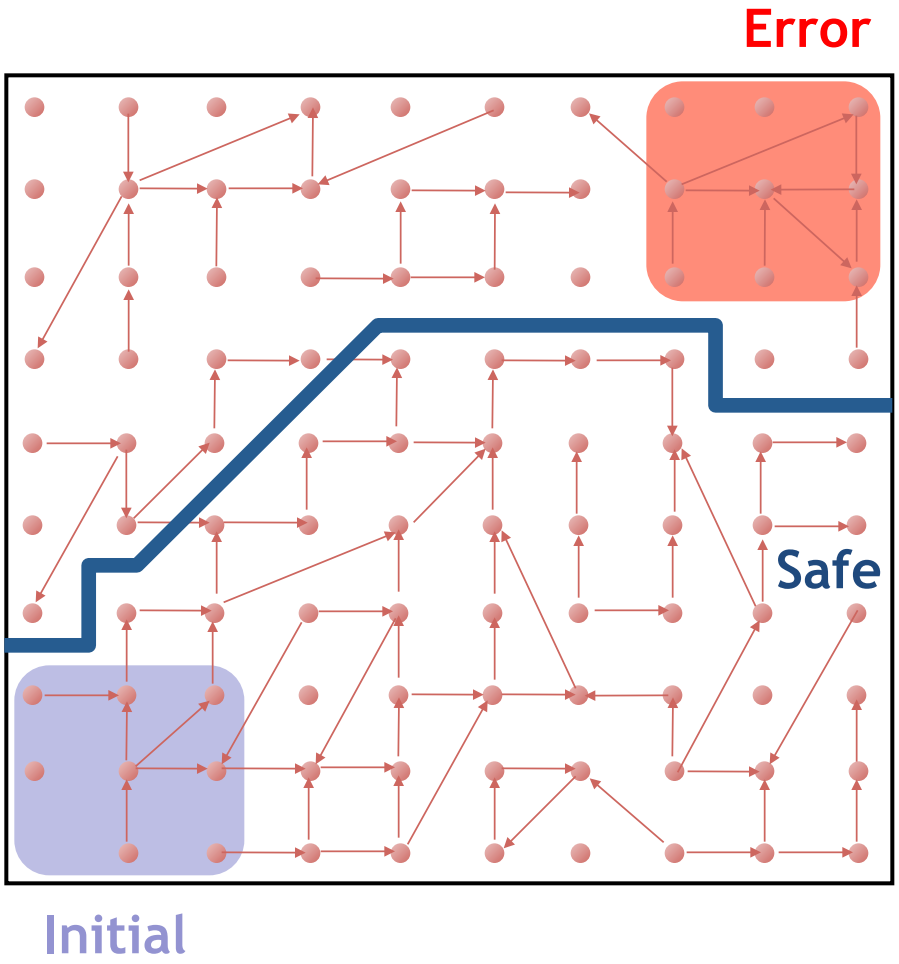
Then, if

$(s_i, s_j) \in r$

we require

$(a_i, a_j) \in a$.

Predicate abstraction

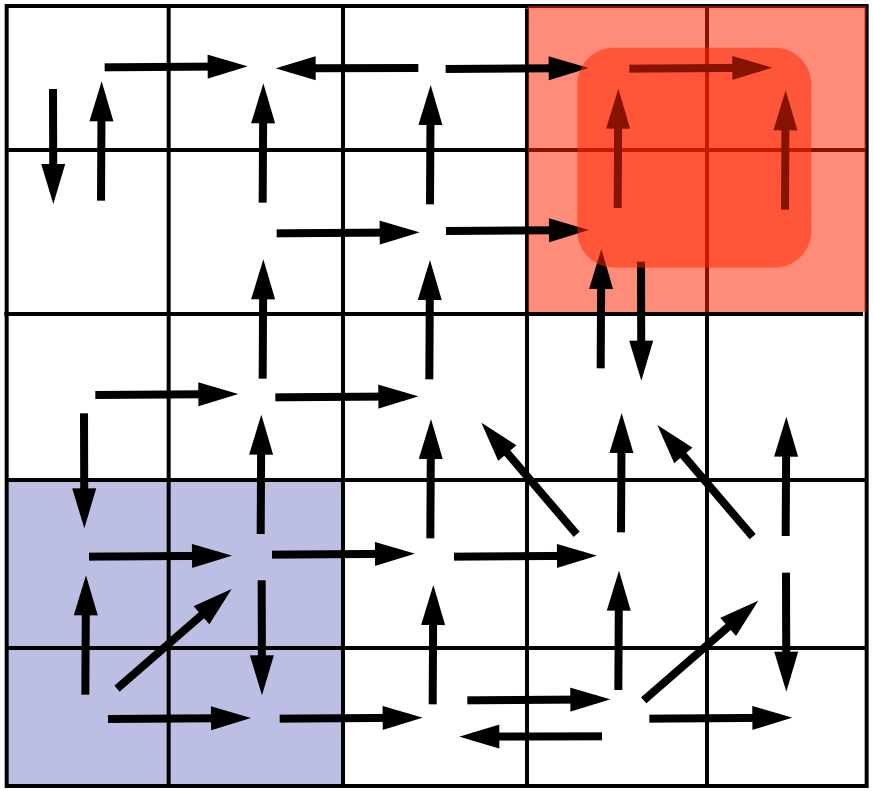


Error states are bad states where the property to check does not hold.

Reachability question:

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

Predicate abstraction



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

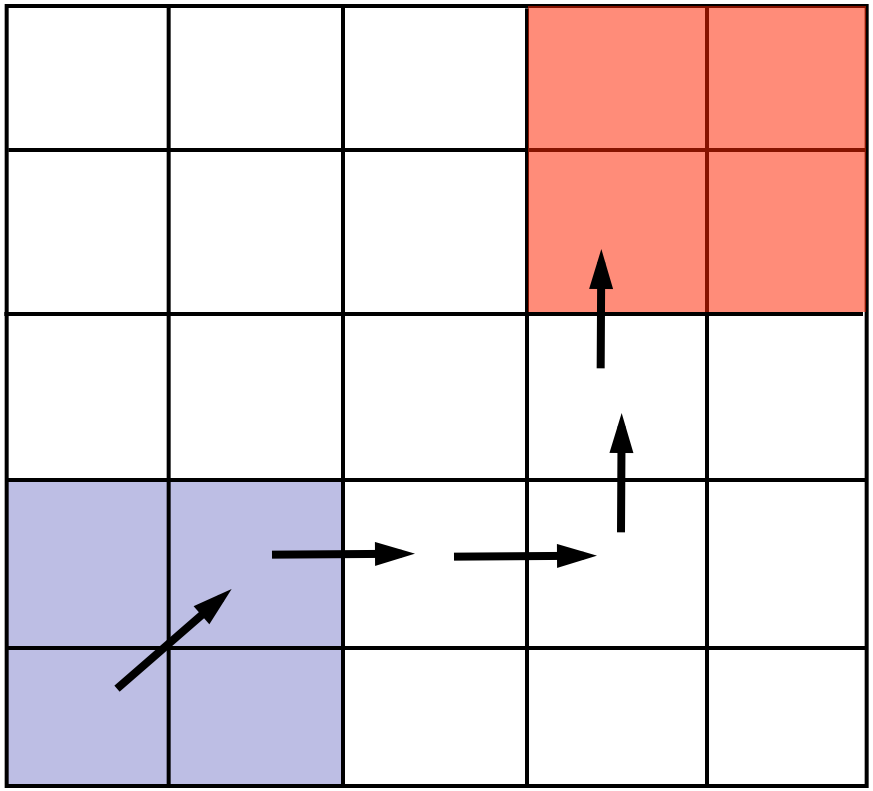
We are guaranteed to not get any false negatives:

if a state is unreachable in abstraction, it is unreachable in the concrete state space.

Outline

1. Predicate abstraction – the idea in pictures
2. Counter-example guided refinement
3. wp, sp for predicate discovery
4. Interpolation

False positives



Suppose we find a path to some error state.

Have we found a true bug in the program?

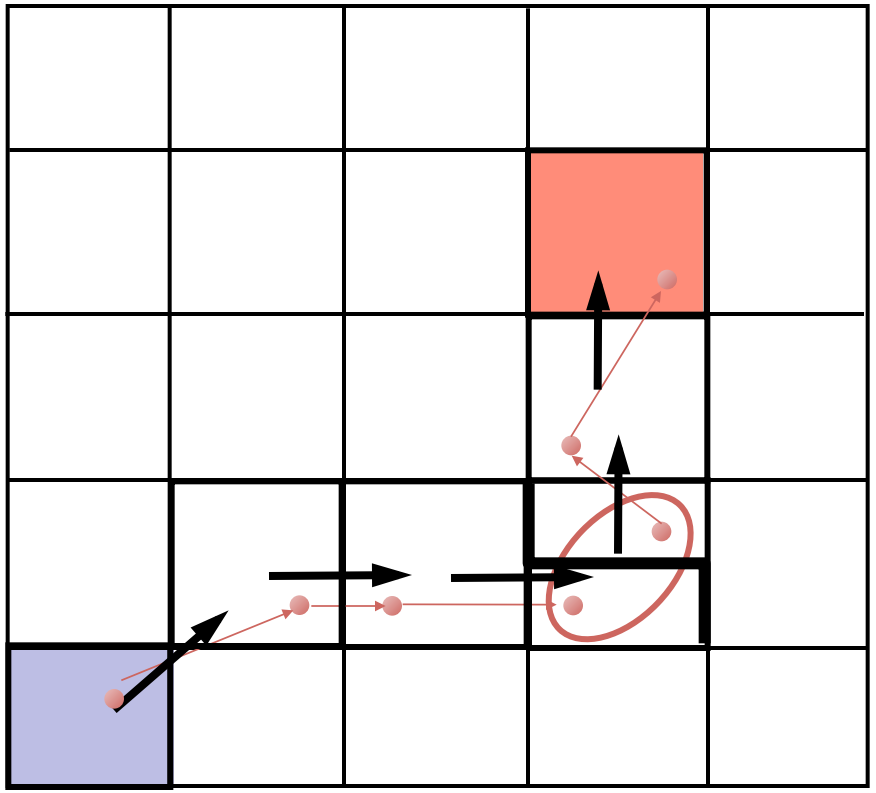
Maybe, or we just found a spurious counterexample.

How to check:

- take the concrete path through the program and construct the formula describing its relation
- feed this formula to a theorem prover
  - path feasible: true bug found, report and finish
  - path infeasible: no bug, refine abstraction

Note: how we get the concrete path will become obvious later.

Counter-example guided refinement



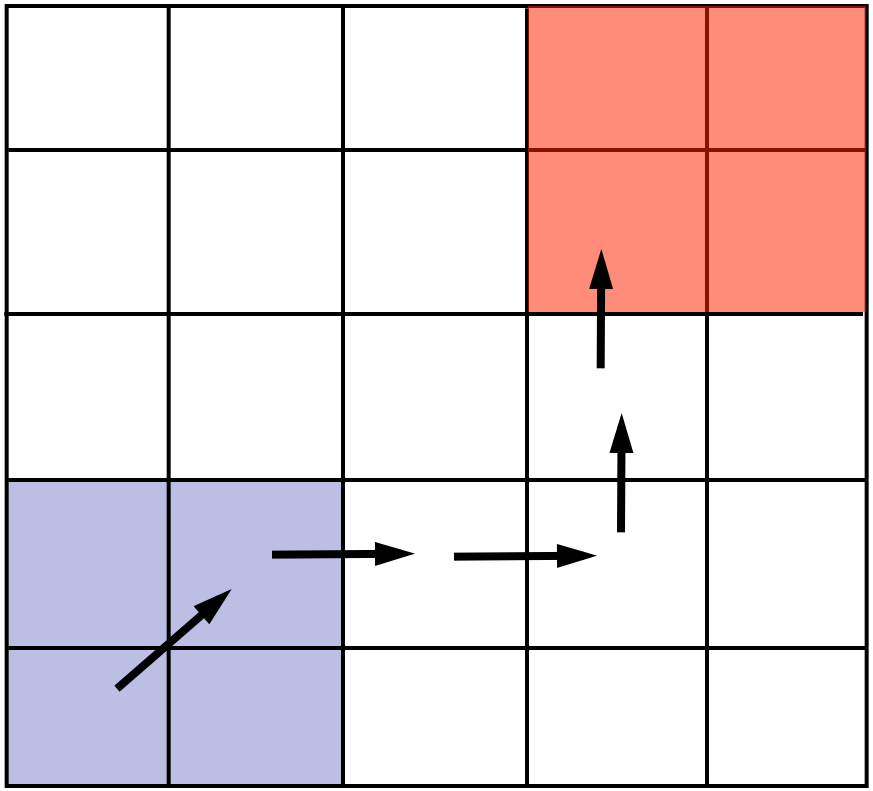
If path is infeasible, add more predicates to distinguish paths and rule out this particular one.

Idea: use infeasible path to generate predicates such that when added, this path will not appear any more.

Repeat until

- find a true counterexample
- system is proven safe
- timeout

Counter-example guided refinement



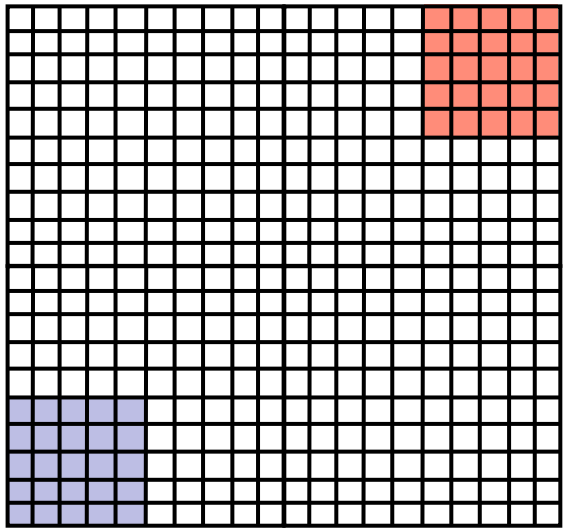
Suppose we have a black-box tool that can provide us with the missing predicates.

P_2 P_3

P_1

We're done, right?

Lazy abstraction

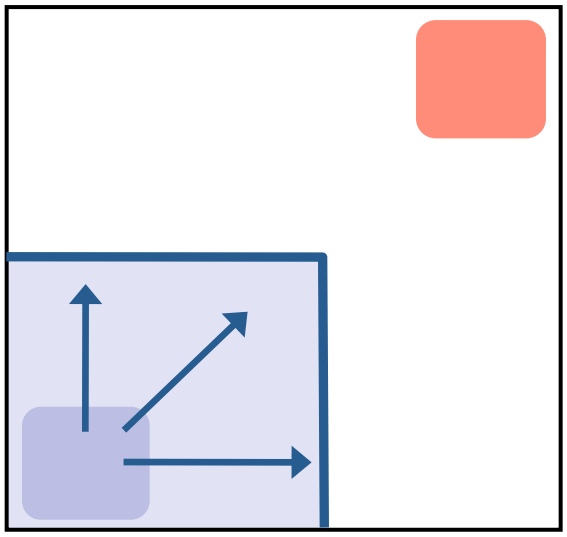


Not quite...

Abstraction is expensive:

abstract states is finite, but still too large:

$$2^{\#\ predicates}$$

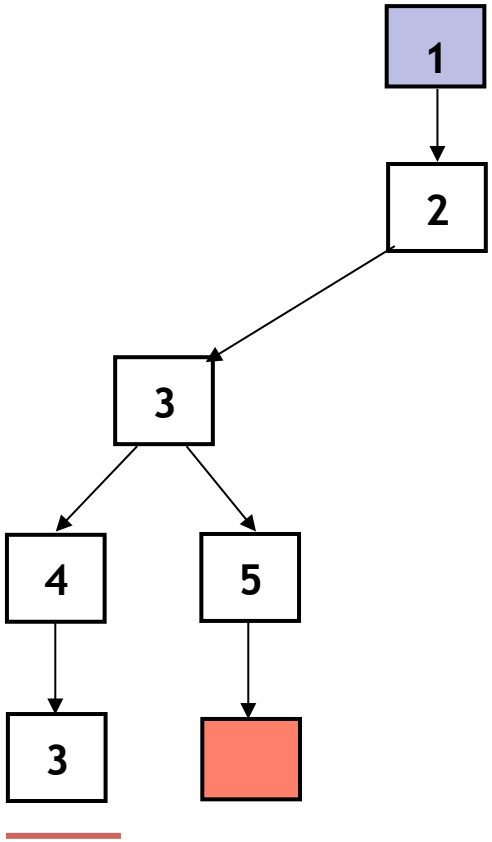


Observation:

- not all predicates are needed everywhere
- only a fraction of states is reachable

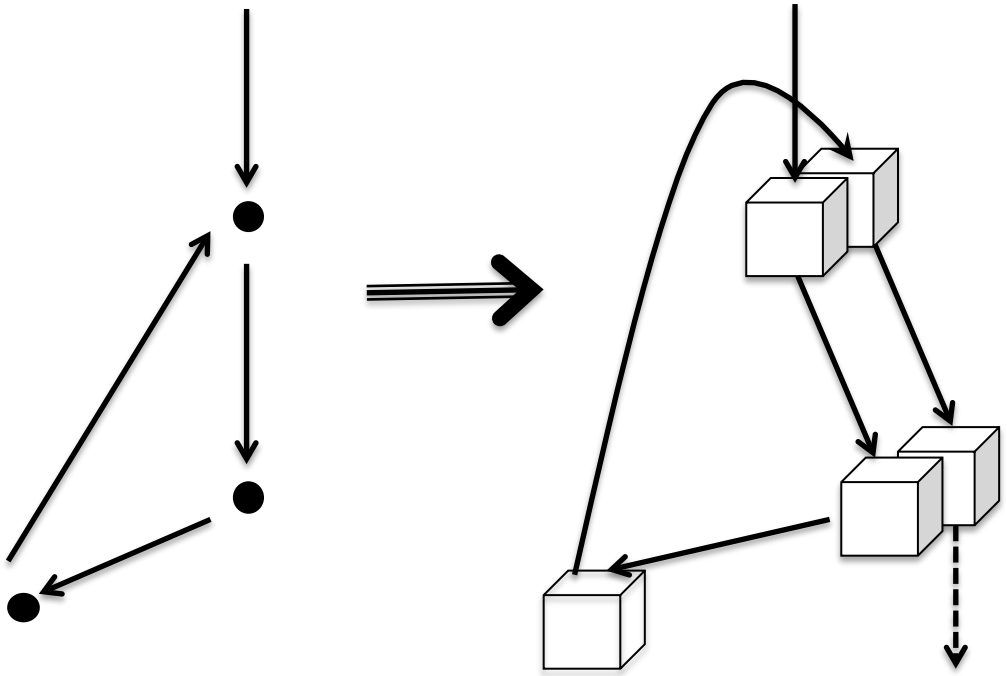
Abstract reachability tree

Initial



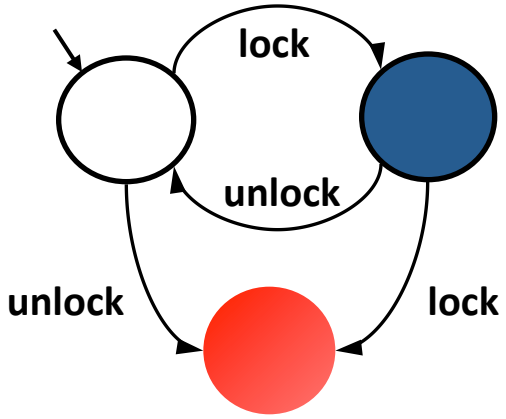
Unroll the CFG:

- pick a tree node
- add children
- if we revisit a state already seen, cut off



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;
}
```

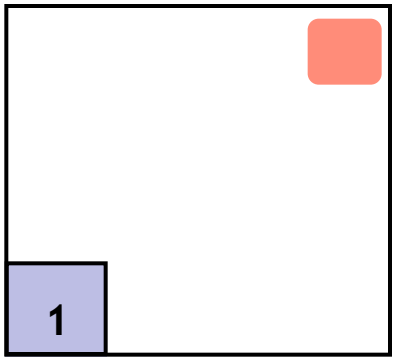


"An attempt to re-acquire an acquired lock or release a released lock will cause a deadlock."

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

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 ();
}
```

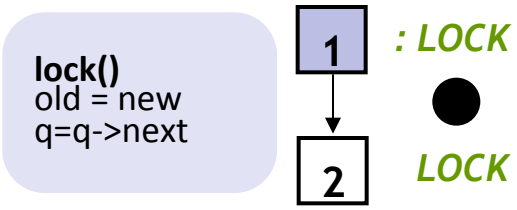


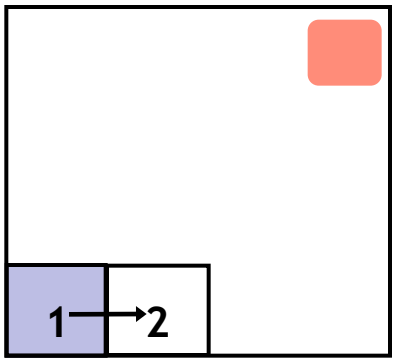
Predicates: *LOCK*

Reachability Tree

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 ();
}
```



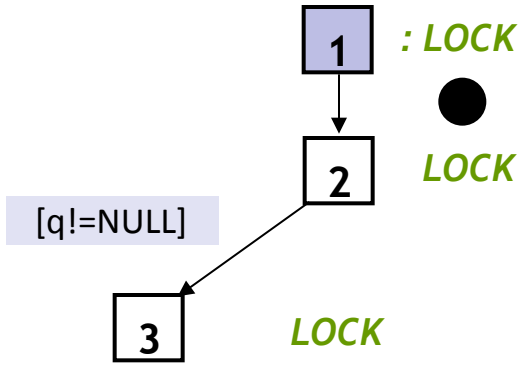


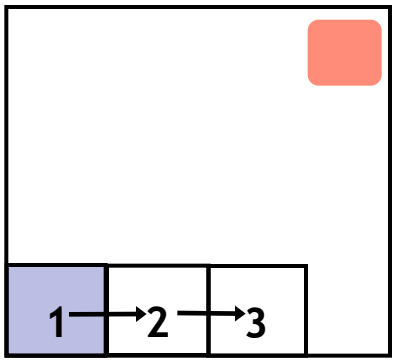
Predicates: *LOCK*

Reachability Tree

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 ();
}
```



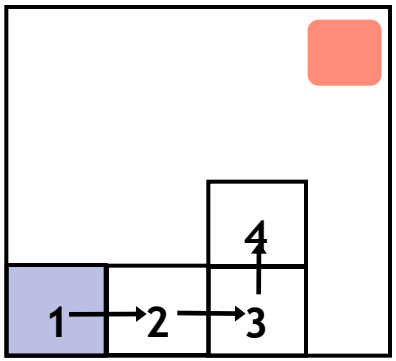


Predicates: *LOCK*

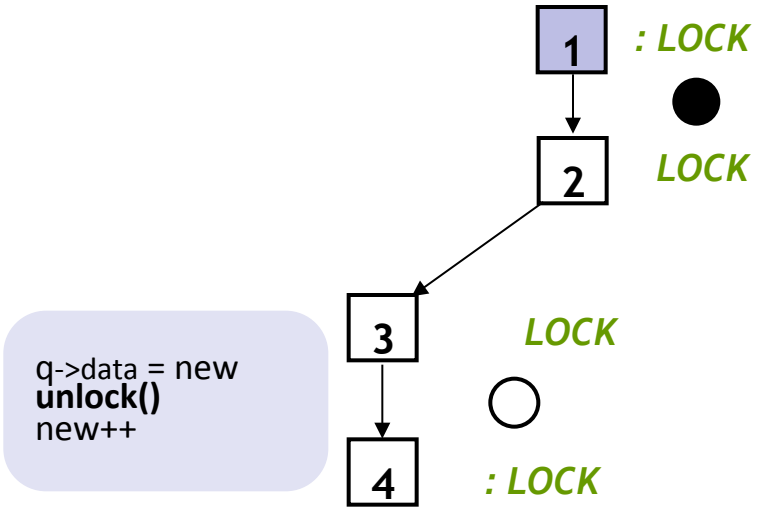
Reachability Tree

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 ();
}
```



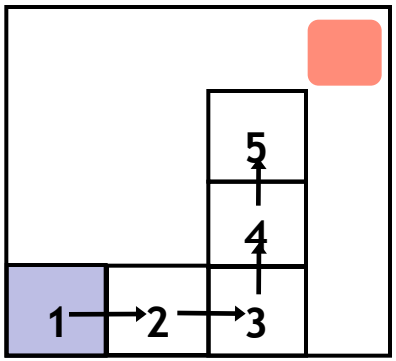
Predicates: *LOCK*



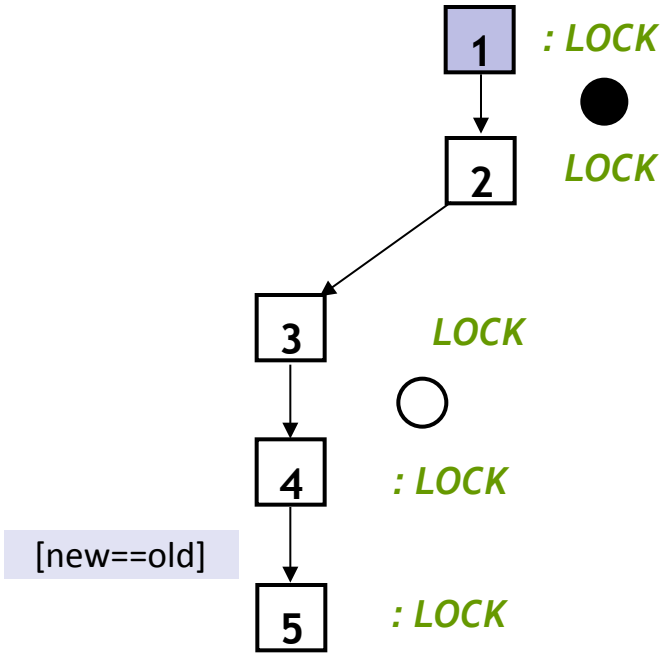
Reachability Tree

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 ();
}
```



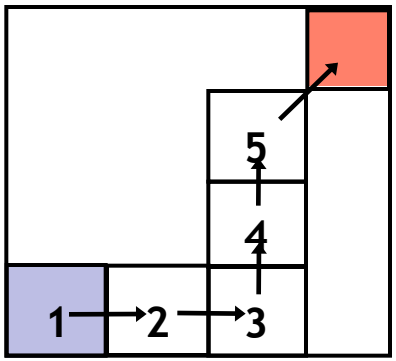
Predicates: *LOCK*



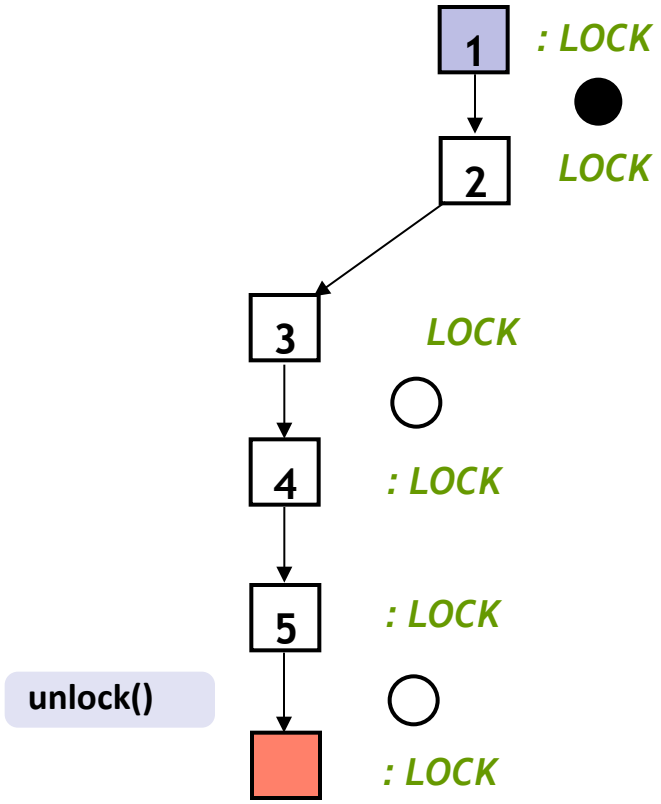
Reachability Tree

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 ();
}
```



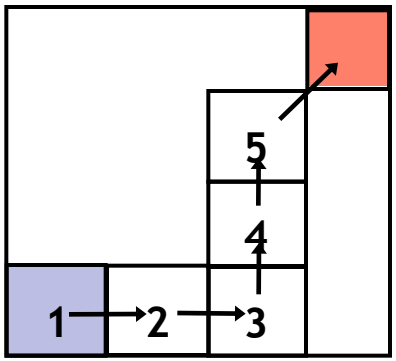
Predicates: *LOCK*



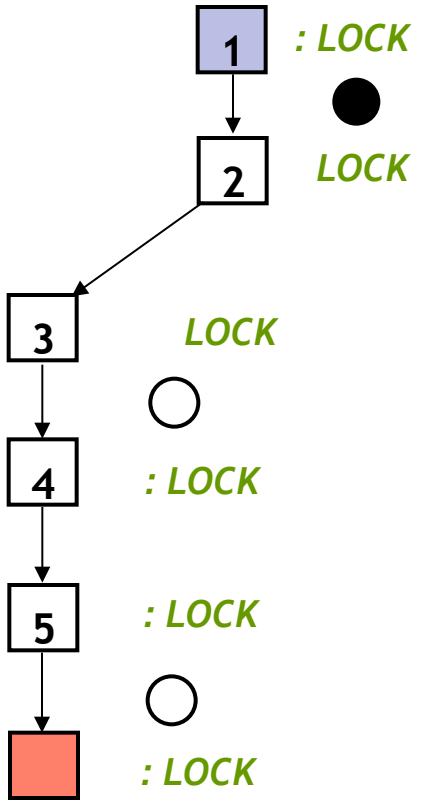
Reachability Tree

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 ();
}
```



Predicates: *LOCK*



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

[q!=NULL]

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

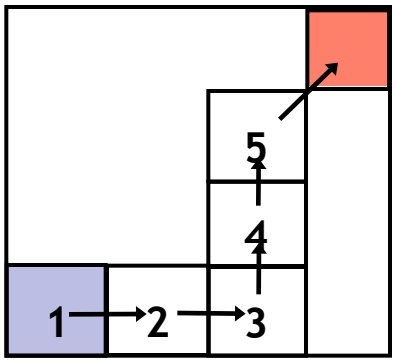
[new==old]

unlock()

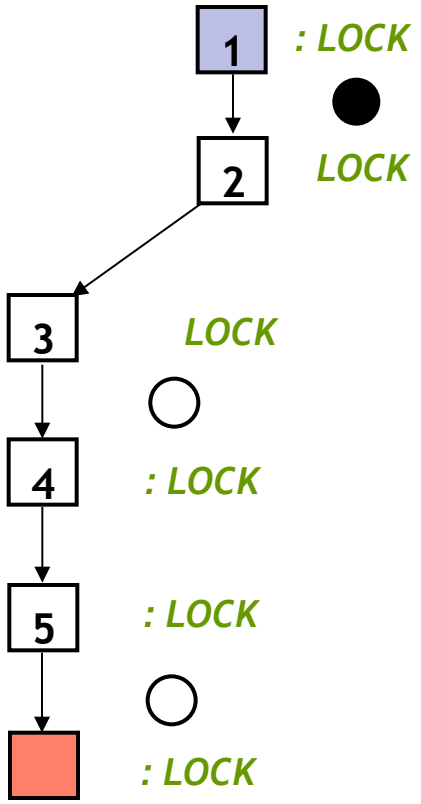
Reachability Tree

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 ();
}
```



Predicates: *LOCK*



Reachability Tree

old = new

new++

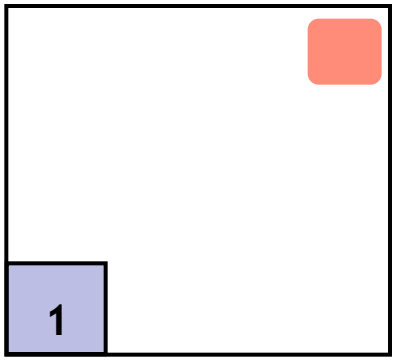
[new==old]

Inconsistent

new == old

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 ();
}
```

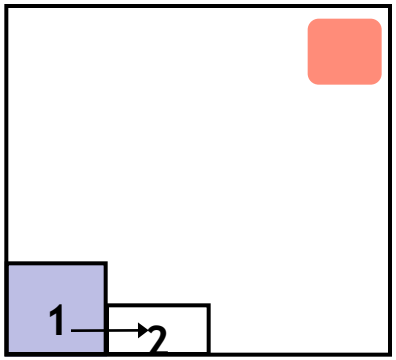


Predicates: *LOCK, new==old*

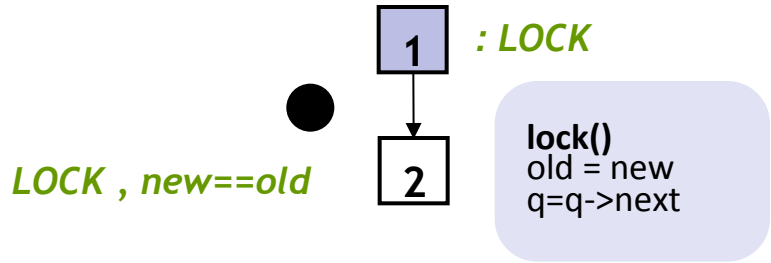
Reachability Tree

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 ();
}
```



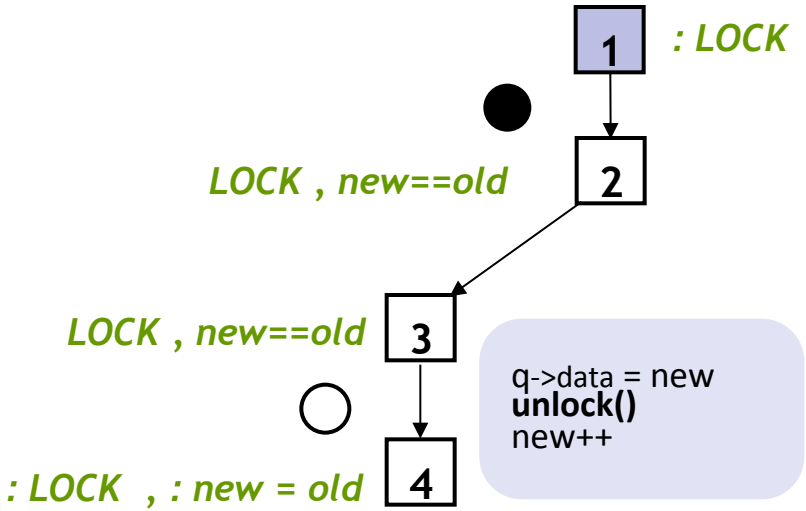
Predicates: *LOCK, new==old*

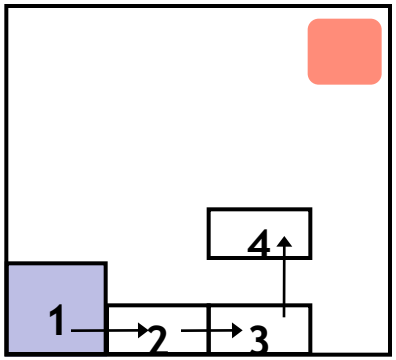


Reachability Tree

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 ();
}
```



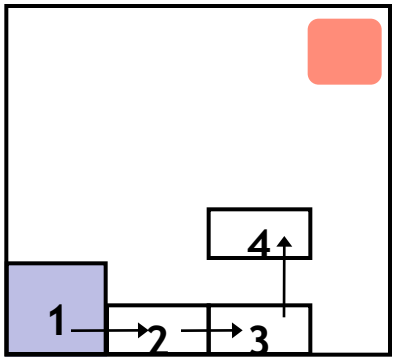


Predicates: *LOCK, new==old*

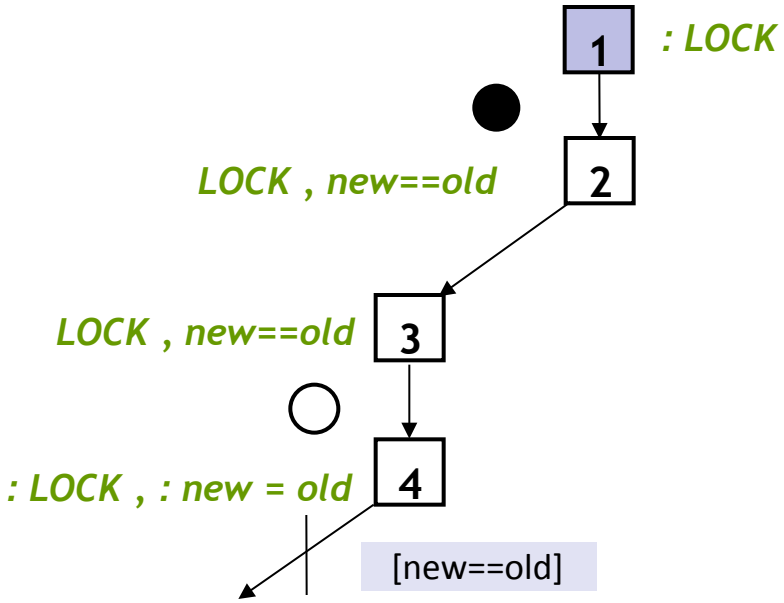
Reachability Tree

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 ();
}
```



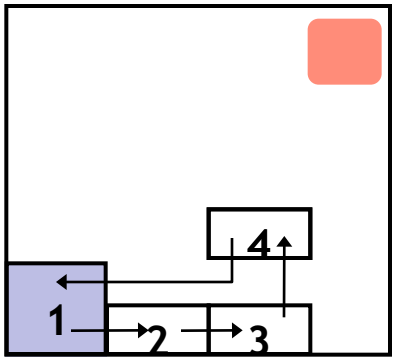
Predicates: *LOCK, new==old*

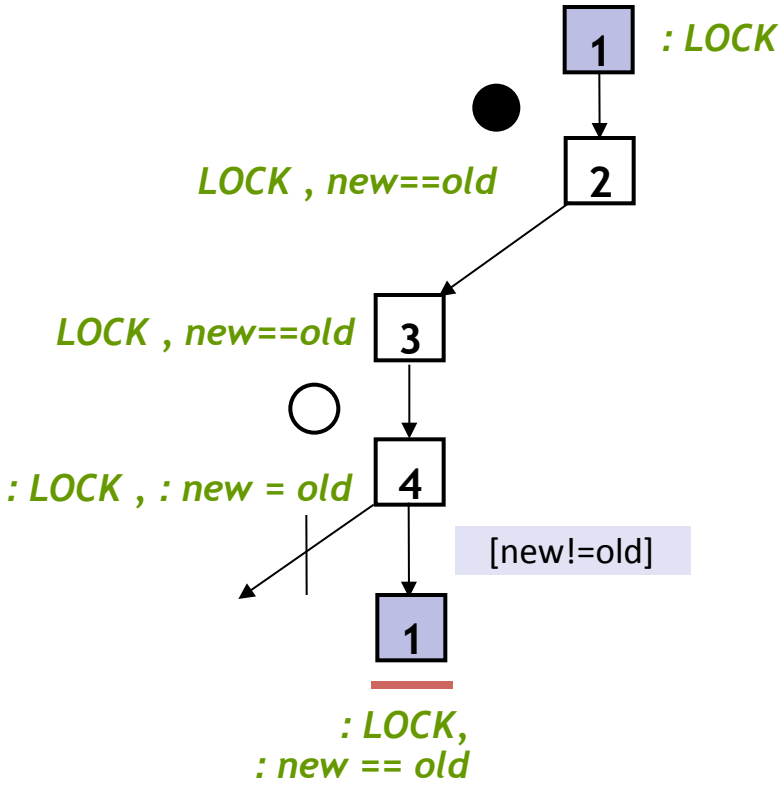


Reachability Tree

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 ();
}
```

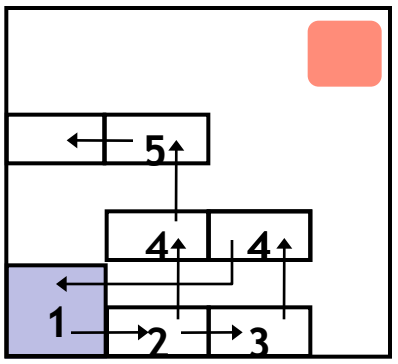




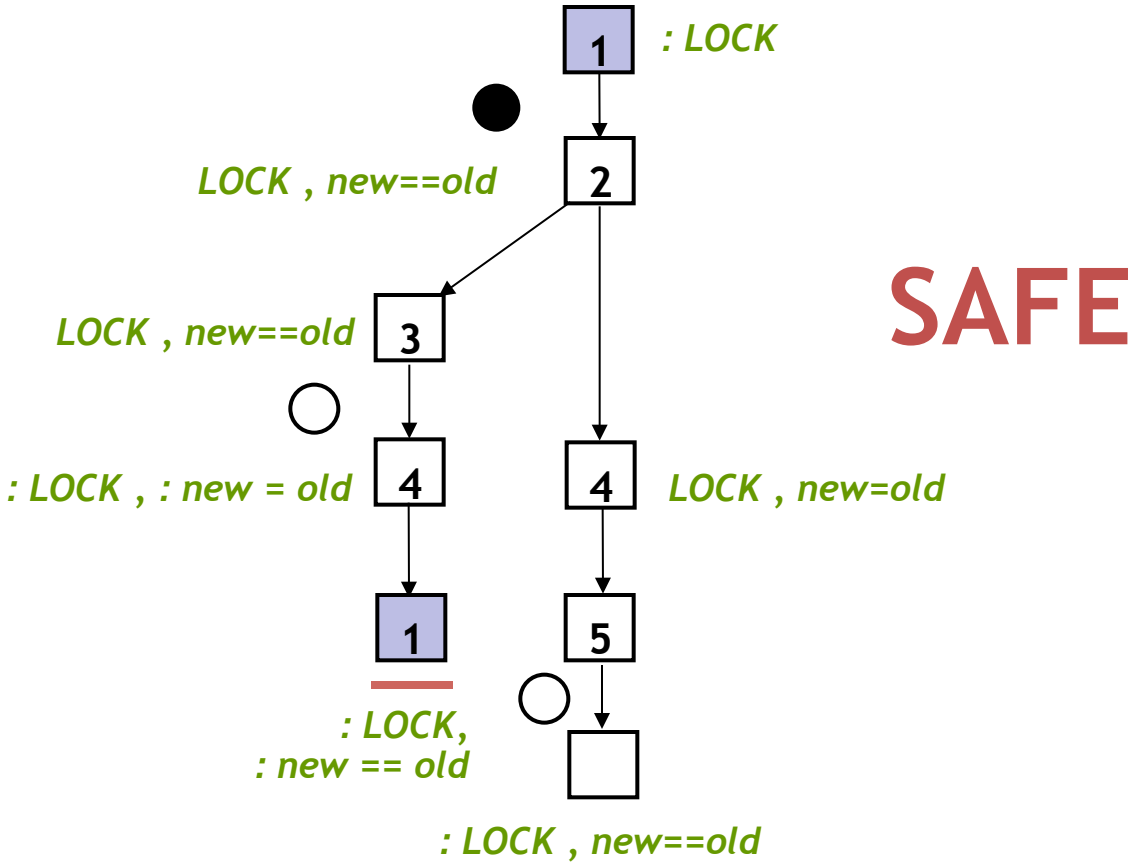
Predicates: *LOCK, new==old*

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 ();
}
```



Predicates: *LOCK, new==old*



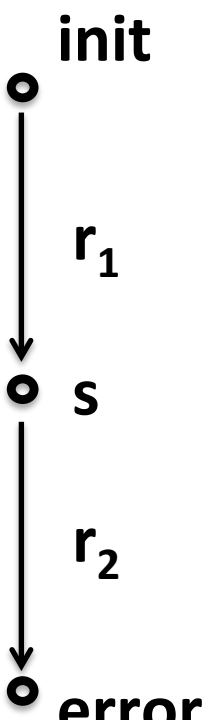
SAFE

Reachability Tree

Outline

1. Predicate abstraction – the idea in pictures
2. Counter-example guided refinement
3. wp, sp for predicate discovery
4. Interpolation

How to find the predicates: wp, sp



We have a path

$$\text{init}(x_1) \wedge r_1(x_1, x_2) \wedge r_2(x_2, x_3)$$

that is infeasible, i.e. set of states at position **error** is empty.

$$\forall x, x'.\neg\big(P(x) \wedge r(x,x') \wedge Q(x')\big)$$

$$\forall x, x'.P(x) \wedge r(x,x') \rightarrow \neg Q(x') \quad \Leftrightarrow \quad \{P\}\, r\, \{\neg Q\}$$

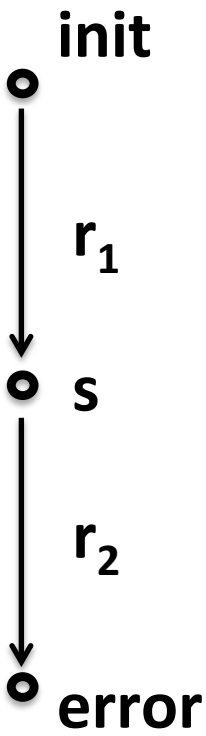
The ‘P’ is what we are looking for, hence use

$$\text{wp}(r_2, \text{false})$$

to derive predicates for position **s**.

We effectively compute the weakest condition such that the error state is not reached.

How to find the predicates: wp, sp



$wp(r_2, false)$

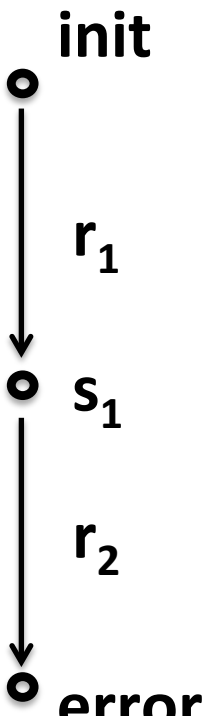
Propagate backwards through the ART to compute predicates for all positions.

Alternatively, use sp(init, r) to compute predicates forwards.

However,

- wp, sp introduce quantifiers
- formulae can become quite complex

What kind of predicates are needed?



Suppose our path consists of states s_1, $s_{2, \ldots,}$ s_n.

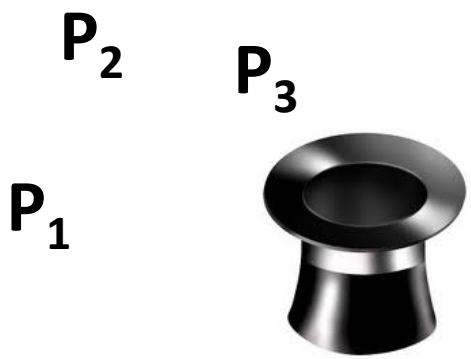
What we want are predicates P_i (corresponding to s_i), such that

P_1 -> P_2 -> … P_{n-1} and

P_{n-1} and P_n are inconsistent.

→ the path has been ruled out.

Note: it is always sound to pick predicates at random!



Outline

1. Predicate abstraction – the idea in pictures
2. Counter-example guided refinement
3. wp, sp for predicate discovery
4. Interpolation

So what is the magic?

Definition:

Given two formulas F and G, such that |= F -> G, an **interpolant** for (F, G) is a formula H such that:

1. |= F -> H
2. |= H -> G
3. H only contains free variables common to both F and G

Craig's interpolation theorem (1957):

Let F and G be formulas in first-order logic. If F -> G is valid, then an interpolant for (F, G) always exists.

(… but it can contain quantifiers.)

Examples

The examples are all in propositional logic:

F: $\left(P \vee \left(Q \wedge R\right)\right)$ H:

G: $\left(P \vee \neg \neg Q\right)$

F: $\left(P \wedge \neg P\right)$ H:

G: Q

F: Q H:

G: $\left(P \vee \neg P\right)$

F: $\neg(P \wedge Q) \rightarrow (\neg R \wedge Q)$ H:

G: $(T \rightarrow P) \vee (T \rightarrow \neg R)$

Examples

The examples are all in propositional logic:

F: $\left(P \vee \left(Q \wedge R\right)\right)$ H: $P \vee Q$

G: $\left(P \vee \neg \neg Q\right)$

F: $\left(P \wedge \neg P\right)$ H: $\bot$

G: Q

F: Q H: T

G: $\left(P \vee \neg P\right)$

F: $\neg(P \wedge Q) \rightarrow (\neg R \wedge Q)$ H: $(\mathrm{P} \vee \neg \mathrm{R})$

G: $(T \rightarrow P) \vee (T \rightarrow \neg R)$

Two simple ways of computing an interpolant

Suppose F -> G.

Let

$$H_{min} \equiv elim(\exists p_1, p_2, ..., p_n.\ F) \text{ where } \{p_1, p_2, ..., p_n\} = FV(F) \backslash FV(G)$$
$$H_{max} \equiv elim(\forall q_1, q_2, ..., q_m.\ G) \text{ where } \{q_1, q_2, ..., q_m\} = FV(G) \backslash FV(F)$$

and let I(F, G) be the set of all interpolants for (F, G):

$$\mathcal{I}(F, G) = \{H \mid H \text{ is interpolant for } (F, G)\}$$

Theorem:

The following properties hold for H_{min}, H_{max}, I(F, G) defined above:

$$(1)\ H_{min} \in \mathcal{I}(F, G)$$
$$(2)\ \forall H \in \mathcal{I}(F, G).\ \models (H_{min} \rightarrow H)$$
$$(3)\ H_{max} \in \mathcal{I}(F, G)$$
$$(4)\ \forall H \in \mathcal{I}(F, G).\ \models (H \rightarrow H_{max})$$

Effectively, H_{min} is the strongest interpolant and H_{max} is the weakest one.

Proof

WLOG, let F be over the variables x, y and G over y, z.
Then by assumption $\forall x, y, z. F(x, y) \rightarrow G(y, z)$ and for any interpolant H in $\mathcal{I}$ it holds

$$\forall x, y.\ F(x, y) \rightarrow H(y)$$

$$\forall y, z.\ H(y) \rightarrow G(y, z)$$

Now, for H_{min} to be an interpolant, it must hold $\forall x, y.\ F(x, y) \rightarrow \exists x_1.\ F(x_1, y)$
This statement is equivalent to $\forall y.\ (\exists x.\ F(x, y)) \rightarrow \exists x_1.\ F(x_1, y)$ which is trivially true.
Similarly $\forall y, z.(\exists x_1.\ F(x_1, y)) \rightarrow G(y, z) \quad \Leftrightarrow \quad \forall x_1, y, z.\ F(x_1, y) \rightarrow G(y, z)$
hence H_{min} is indeed an interpolant.
To show that it is the strongest interpolant consider $\forall x, y.\ F(x, y) \rightarrow H(y)$ which is equivalent to $\forall y.\ (\exists x.\ F(x, y)) \rightarrow H(y)$ which is what we wanted to show.
The proof for H_{max} follows similarly.

Remarks

- By the last theorem, if a theory has quantifies elimination, then it also has interpolants, e.g.
  - Presburger arithmetic
  - field of complex and real numbers
  - mixed linear and integer constraints

- But, interpolants may exist even if there is no quantifier elimination (e.g. FOL).
- There are also other ways of computing them.

- Some theories do not have interpolants, e.g. quantifier free theory of arrays

$$F: \quad M' = wr(M, x, y)$$
$$G: \quad (a \neq b) \wedge (rd(M, a) \neq rd(M', a)) \wedge (rd(M, b) \neq rd(M', b))$$

Since the interpolant cannot use x, y, a, b, it has to use quantifiers.

Alternatively

Instead of validity of implication, we can consider unsatisfiability.

Definition:

Given two formulas F and G, such that F ^ G is inconsistent, an interpolant for (F, G) is a formula H such that:

1. |= F -> H
2. H ^ G is inconsistent
3. H only contains free variables common to both F and G

Intuition:

H is an abstraction of F containing just the inconsistent information with G.

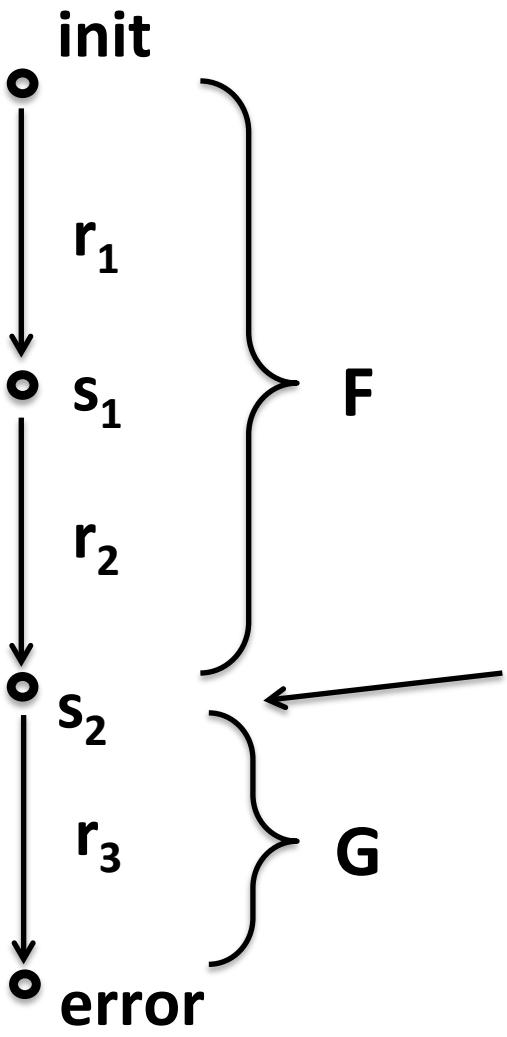
I.e. H is the *reason* why F and G are inconsistent.

Example:

F: $2z-1 \leq 0 \wedge y-z+2 \leq 0$ H: $x-y \leq 0 \wedge -3x+y+1 \leq 0$ Interpolant: $2y+3 \leq 0$

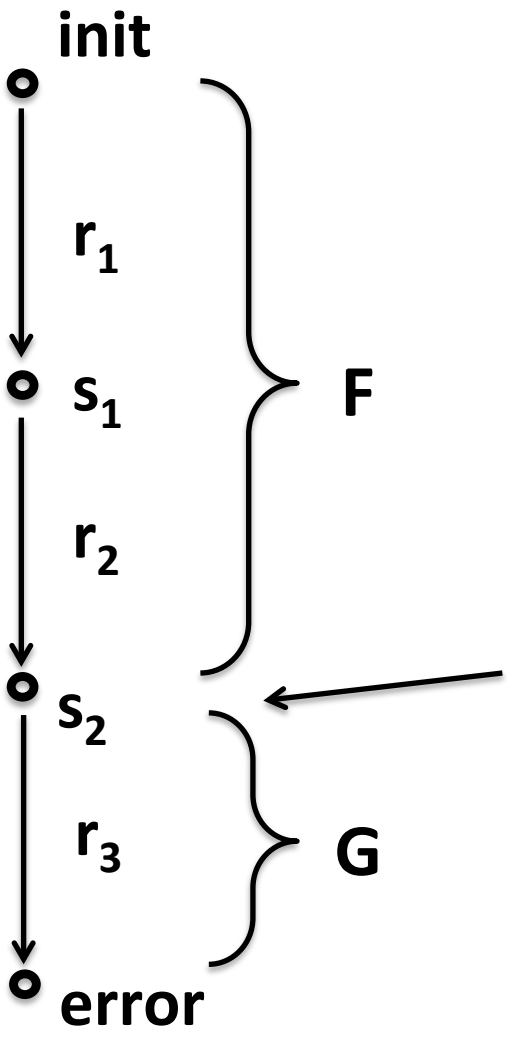
It is possible to extract interpolants from a proof of F ^ G being inconsistent.

Putting it all together



The information the executing program has at this point contains variables common to F and G.

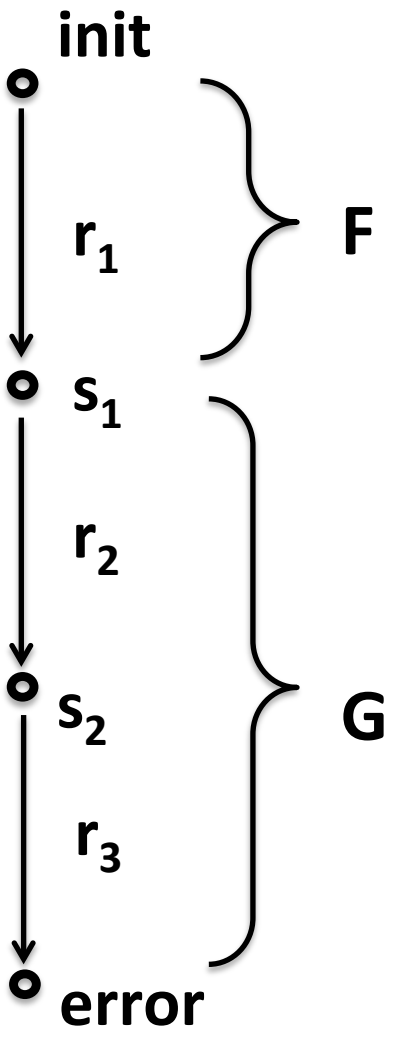
Putting it all together



Use the interpolant H here as additional predicate.

Note: there is way to compute interpolants from the proof of unsatisfiability of the path.

Putting it all together



Repeat for all nodes in the ART.