Property 1: Double Locking

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

Calls to lock and unlock must alternate.
Property 2: Drop Root Privilege

"User applications must not run with root privilege"

When `execv` is called, must have `suid \neq 0`

Property 3: IRP Handler

[Chen-Dean-Wagner '02]

[Fahndrich]
This Lecture

- **Model Checking**
  - Path sensitive program analysis

- **Lazy abstraction**
  - Compute abstraction on demand

- **Counterexample refinement**
  - Use false positives to refine abstraction

Example

```c
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;
}  
```

Diagram:

```
lock -> unlock

1. lock
2. unlock
```

11/29/17
What a program really is...

State Transition

Example ( ) {
1: do {
2: lock();
3: unlock();
new++;
4: } ...
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
Representing States as *Formulas*

\[
[F] \quad \text{states satisfying } F \{ s \mid s \models F \} \\
F \quad \text{FO formula over prog. vars}
\]

\[
[F_1] \text{intersect} [F_2] \\
F_1 \land F_2
\]

\[
[F_1] \text{union} [F_2] \\
F_1 \lor F_2
\]

\[
\text{complement} [F] \\
\neg F
\]

\[
[F_1] \text{subseteqq} [F_2] \\
F_1 \Rightarrow F_2
\]

**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

\[ pc \mapsto 3 \]
lock \mapsto \bullet
old \mapsto 5
new \mapsto 5
q \mapsto 0x133a

\[ \text{3: unlock(); new++;} \]
\[ \text{4:} \]

\[ pc \mapsto 4 \]
lock \mapsto 0
old \mapsto 5
new \mapsto 6
q \mapsto 0x133a

\[ \text{Theorem Prover} \]

lock
old=new

Abstract States and Transitions

Abstract States and Transitions

State

\[ pc \mapsto 3 \]
lock \mapsto \bullet
old \mapsto 5
new \mapsto 5
q \mapsto 0x133a

\[ \text{3: unlock(); new++;} \]
\[ \text{4:} \]

\[ pc \mapsto 4 \]
lock \mapsto 0
old \mapsto 5
new \mapsto 6
q \mapsto 0x133a

\[ \text{Theorem Prover} \]

lock
old=new

Existential Lifting

Existential Lifting

Abstract States and Transitions

Abstract States and Transitions

State

\[ pc \mapsto 3 \]
lock \mapsto \bullet
old \mapsto 5
new \mapsto 5
q \mapsto 0x133a

\[ \text{3: unlock(); new++;} \]
\[ \text{4:} \]

\[ pc \mapsto 4 \]
lock \mapsto 0
old \mapsto 5
new \mapsto 6
q \mapsto 0x133a

\[ \text{Theorem Prover} \]

lock
old=new

Existential Lifting

Existential Lifting
Abstraction

State

pc \mapsto 3
lock \mapsto \bullet
old \mapsto 5
new \mapsto 5
q \mapsto 0x133a

3: unlock();
new++;
4} ...

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

lock
old=new

! lock
! old=new

Analyze Abstraction

Analyze finite graph

Over Approximate:
Safe \implies System Safe
No false negatives

Problem
Spurious counterexamples
Solution

Use spurious counterexamples to refine abstraction!

Idea 2: Counterexample-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

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

The Big Picture

Program
Property
predicate abstraction
refinement predicates
boolean program
symbolic reachability
counterexamples
error path
path feasibility & predicate discovery

[Kurshan et al. '93]
[Clarke et al. '00]
[Ball, Rajamani '01]
**Problem:** Abstraction is Expensive

Reachable States

<table>
<thead>
<tr>
<th>Problem</th>
<th>Observe</th>
</tr>
</thead>
<tbody>
<tr>
<td>#abstract states = $2^{#predicates}$</td>
<td>Fraction of state space reachable</td>
</tr>
<tr>
<td>Exponential Thm. Prover queries</td>
<td>$#Preds \sim 100's$, $#States \sim 2^{100}$, $#Reach \sim 1000's$</td>
</tr>
</tbody>
</table>

**Solution 1:** Only Abstract Reachable States

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>#abstract states = $2^{#predicates}$</td>
<td>Build abstraction <em>during</em> search</td>
</tr>
<tr>
<td>Exponential Thm. Prover queries</td>
<td>Safe</td>
</tr>
</tbody>
</table>
Solution 2: Don’t Refine Error-Free Regions

Problem
#abstract states = 2^#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 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.

**SAFE**

**S1:** Only abstract reachable states
**S2:** Don’t refine error-free regions
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();
}
Example() {
    1: do {
        lock();
        old = new;
        q = q->next;
        2: if (q != NULL) {
            q->data = new;
            unlock();
            new++;
            3
        } while (new != old);
        5: unlock();
    } while (new != old);
}

Reachability Tree

Predicates: LOCK

---

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

Reachability Tree

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

Reachability Tree
Analyze Counterexample

Example()
1: do {
2:   lock();
3:   old = new;
4:   q = q->next;
5: if (q != NULL) {
6:     q->data = new;
7:     unlock();
8:     new++;
9:   }
10: } while (new != old);
11: unlock();

Predicates: LOCK

Reachability Tree

Inconsistent
new == old

Analyze Counterexample

Example()
1: do {
2:   lock();
3:   old = new;
4:   q = q->next;
5: if (q != NULL) {
6:     q->data = new;
7:     unlock();
8:     new++;
9:   }
10: } while (new != old);
11: unlock();

Predicates: LOCK

Reachability Tree

Inconsistent
new == old
Repeat Build-and-Search

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

Reachability Tree

Predicates: \textbf{LOCK}, \textbf{new==old}

Repeat Build-and-Search

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

Reachability Tree

Predicates: \textbf{LOCK}, \textbf{new==old}
Repeat Build-and-Search

Example() {
I: do {
  lock();
  old = new;
  q = q->next;
  if (q != NULL) {
    q->data = new;
    unlock();
    new++;
  }
} while (new != old);
unlock();
}

Predicates: LOCK, new==old

Reachability Tree

Repeat Build-and-Search

Example() {
I: do {
  lock();
  old = new;
  q = q->next;
  if (q != NULL) {
    q->data = new;
    unlock();
    new++;
  }
} while (new != old);
unlock();
}

Predicates: LOCK, new==old

Reachability Tree
Repeat Build-and-Search

Example() {
    do {
        lock();
        old = new;
        q = q->next;
        if (q != NULL) {
            q->data = new;
            unlock();
            new++;
        }
    } while (new != old);
    unlock();
}

Reachability Tree

Predicates: LOCK, new==old

Repeat Build-and-Search

Example() {
    do {
        lock();
        old = new;
        q = q->next;
        if (q != NULL) {
            q->data = new;
            unlock();
            new++;
        }
    } while (new != old);
    unlock();
}

Reachability Tree

Predicates: LOCK, new==old

SAFE
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.

Error Free

SAFE

S1: Only Abstract Reachable States
S2: Don’t refine error-free regions

Two handwaves

Example:

```c
1: do {
    lock();
    old = new;
    q = q->next;
2: if (q != NULL) {
3:     q->data = new;
4:     unlock();
new++;
5: } while (new != old);
6: unlock();
```
Q. How to compute “successors”?

Q. How to find predicates?

Refinement

Predicates: LOCK, new==old
Example() {
  do {
    lock();
    old = new;
    q = q->next;
    unlock();
    new++;
  } while (new != old);
  unlock();
} 

Q. How to compute “successors”?

Q. How to find predicates?

Refinement

Predicates: $LOCK, new == old$

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 \( OP \)

\[ \begin{align*}
    WP(P, OP) & \quad \text{OP} \\
    \text{Assign} & \\
    x = e & \\
    P'[e/x] & \\
    \text{new+1 = old} & \\
    \text{new = new+1} & \\
    \text{new = old} & \\
\end{align*} \]

How to Compute Successor?

Example:

```c
void Example() {
    for (x = 0; x < n; x++) {
        if (\( x < n \)) {
            // Some code...
        }
    }
}
```

For each \( p \)
- Check if \( p \) is true (or false) after \( OP \)

Q: When is \( p \) true after \( OP \)?
- If \( WP(p, OP) \) is true before \( OP \)
- We know \( F \) is true before \( OP \)
- Thm. Prover Query: \( F \Rightarrow WP(p, OP) \)

Predicates: \( LOCK, new=old \)
**Lazy Abstraction**

Program \rightarrow Abstract \rightarrow Yes \rightarrow Proof

Property \rightarrow Refine \rightarrow No \rightarrow Counter example

**Problem: Abstraction is Expensive**

**Solution:** 1. Only abstract reachable states,
2. Don’t refine error-free regions

**Key Idea:** Reachability Tree

---

**Property: IRP Handler**

[Fahndrich]
Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines*</th>
<th>Time(mins)</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbfiltr</td>
<td>12k</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>floppy</td>
<td>17k</td>
<td>7</td>
<td>93</td>
</tr>
<tr>
<td>diskprf</td>
<td>14k</td>
<td>5</td>
<td>71</td>
</tr>
<tr>
<td>cdaudio</td>
<td>18k</td>
<td>20</td>
<td>85</td>
</tr>
</tbody>
</table>

* Pre-processed

Lazy Abstraction: Main Ideas

- **Predicates:**
  - Abstract infinite program states

- **Counterexample-guided Refinement:**
  - Find predicates tailored to program, property

1. **Abstraction**: Expensive
   - Reachability Tree

2. **Refinement**: Find new predicates