Having a BLAST with SLAM

If anyone hits me with a snowball, I’ll hit him with 250 snowballs!

What if somebody hits you with 250 snowballs?

For example, I’ve cleared off this corner of my bed. Take a picture of me here, but crop out all the mess around me, so it looks like I keep my room tidy.

Is this even legal?

Wait, let me comb my hair and put on a tie.

...Sighhh...
Topic: Software Model Checking via Counter-Example Guided Abstraction Refinement

- There are easily two dozen SLAM/BLAST/MAGIC papers; I will skim.
SLAM Overview

- **INPUT:** Program and Specification
  - Standard C Program (pointers, procedures)
  - Specification = Partial Correctness
    - Given as a finite state machine (typestate)
    - “I use locks correctly”, *not* “I am a webserver”

- **OUTPUT:** Verified or Counterexample
  - Verified = program does not violate spec
    - Can come with proof!
  - Counterexample = concrete bug instance
    - A path through the program that violates the spec
Take-Home Message

• **SLAM** is a *software model checker*. It *abstracts* C programs to *boolean programs* and model-checks the boolean programs.

• No errors in the boolean program implies no errors in the original.

• An error in the boolean program *may* be a real bug. Or SLAM may *refine* the abstraction and start again.
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 \texttt{execv} is called, must have \texttt{suid} \neq 0

[Chen-Dean-Wagner '02]
Property 3: IRP Handler

IRP accessible

start NP

MPR3 \(\rightarrow\) synch \(\rightarrow\) MPR completion

MPR2 \(\rightarrow\) not pending returned \(\rightarrow\) CallDriver

MPR1 \(\rightarrow\) MPR completion

NP \(\rightarrow\) prop completion

SKIP1 \(\rightarrow\) CallDriver \(\rightarrow\) SKIP2

IPC \(\rightarrow\) CallDriver

DC \(\rightarrow\) N/A \(\rightarrow\) N/A

return child status

return not Pend

start P

Mark Pending

MPR3 \(\rightarrow\) synch

MPR2 \(\rightarrow\) not pending returned \(\rightarrow\) CallDriver

MPR1 \(\rightarrow\) MPR completion

NP \(\rightarrow\) prop completion

SKIP1 \(\rightarrow\) CallDriver \(\rightarrow\) SKIP2

IPC \(\rightarrow\) CallDriver

DC \(\rightarrow\) N/A \(\rightarrow\) N/A

return Pending

[Fahndrich]
Example ( ) {
  do{
    lock();
    old = new;
    q = q->next;
  } if (q != NULL){
  } q->data = new;
  unlock();
  new ++;
} while(new != old);
unlock();
return;
}
SLAM in a Nutshell

\[
\text{SLAM}(\text{Program } p, \text{ Spec } s) = \begin{align*}
\text{Program } q &= \text{incorporate_spec}(p, s); \\
\text{mutable PredicateSet } abs &= \{ \};
\end{align*}
\]

while true do

\[
\text{BooleanProgram } b = \text{abstract}(q, abs); \\
\text{match } \text{model_check}(b) \text{ with} \\
| \text{No Error} \rightarrow \text{printf("no bug"); exit(0)} \\
| \text{Counterexample}(c) \rightarrow \\
\hspace{1cm} \text{if } \text{is_valid_path}(c, p) \text{ then} \\
\hspace{2cm} \text{printf("real bug"); exit(1)} \\
\hspace{1cm} \text{else} \\
\hspace{2cm} abs \leftarrow abs \cup \text{new_preds}(c)
\]

\]
done
Incorporating Specs

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

Example ( ) {
1: do{
    if L=1 goto ERR;
    else L=1;
    old = new;
    q = q->next;
2:    if (q != NULL){
3:        q->data = new;
    if L=0 goto ERR;
    else L=0;
    new ++;
2:    } while(new != old);
3: if L=0 goto ERR;
    else L=0;
    return;
ERR: abort();
}

Original program violates spec iff new program reaches ERR
Program As Labeled Transition System

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 (old=1, old=2, old=...)

Solution: Set of states \( \models \) logical formula
Representing
[Sets of States] as *Formulas*

<table>
<thead>
<tr>
<th>([F]) states satisfying (F) ({s \mid s \models F})</th>
<th>(F) FO fmla over prog. vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>([F_1] \cap [F_2])</td>
<td>(F_1 \land F_2)</td>
</tr>
<tr>
<td>([F_1] \cup [F_2])</td>
<td>(F_1 \lor F_2)</td>
</tr>
<tr>
<td>(\overline{F})</td>
<td>(\neg F)</td>
</tr>
<tr>
<td>([F_1] \subseteq [F_2])</td>
<td>(F_1 \Rightarrow F_2)</td>
</tr>
</tbody>
</table>

i.e. \(F_1 \land \neg F_2\) unsatisfiable
Idea 1: Predicate Abstraction

- **Predicates** on program state:
  - `lock` \( (i.e., \ text{lock=\text{true}}) \)
  - `old = new`

- States satisfying **same** predicates are **equivalent**
  - Merged into one **abstract state**

- **#abstract states is finite**
  - Thus model-checking the abstraction will be feasible!
Abstract States and Transitions

State

3: unlock();
new++;
Abstraction

Existential Lifting
(i.e., $A_1 \rightarrow A_2$ iff $\exists c_1 \in A_1. \exists c_2 \in A_2. c_1 \rightarrow c_2$)

Theorem Prover

$3$: unlock();
new++;

$4$: }

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

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

State

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

Analyze finite graph

Over Approximate:
Safe $\Rightarrow$ System Safe
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

Solution
Use spurious counterexamples to refine abstraction

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

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

![Diagram](image1)

**Problem**

\#abstract states = 2\#predicates

Exponential Thm. Prover queries

**Observe**

Fraction of state space reachable

\#Preds \sim 100’s, \#States \sim 2^{100}, \#Reach \sim 1000’s
**Solution 1:** 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

Exponential Thm. Prover queries

Solution

Don’t refine error-free regions
In T.S. Eliot's 1939 Old Possum's Book Of Pratical Cats, this "mystery cat is called the hidden paw / for he's a master criminal who can defy the law."
Q: Computer Science

• This American Turing award winner is sometimes called the “father” of analysis of algorithms, and is known for popularizing asymptotic notation, creating TeX, and co-developing a popular string search algorithm. His most famous work is *The Art of Computer Programming*.
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.

Error Free
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

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();
}

Reachability Tree

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();
}
Build-and-Search

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

Reachability Tree

Predicates: LOCK
Build-and-Search

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

Predicates: LOCK

Reachability Tree
**Build-and-Search**

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

**Reachability Tree**

Predicates:  \( \text{LOCK} \)
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 ();
}

Reachability Tree

Predicates: $\text{LOCK}$
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();
}

Reachability Tree

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();
}

Reachability Tree

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();
}

Predicates: LOCK, new==old
Repeat Build-and-Search

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

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

Reachability Tree
Repeat Build-and-Search

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

Reachability Tree

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 ++;
      }
    }while(new != old);
  4:}unlock ();
}
Repeat Build-and-Search

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

Reachability Tree

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();
}
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
Two handwaves

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

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

Q. How to compute “successors”? 

Reachability Tree

Predicates: LOCK, new==old
Two handwaves

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. How to compute “successors”?

Q. How to find predicates?

Refinement

Predicates:  \textit{LOCK, new==old}
Two handwaves

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

Q. How to compute “successors”?
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 \( OP \)

Assign \( x = e \)

\[ P[e/x] \]

\[ new = new + 1 \]

\[ new = old \]
How to compute successor?

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

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. Pvr. Query: $F \Rightarrow WP(p, OP)$

Predicates: $LOCK, new==old$
How to compute successor?

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

LOCK, new==old

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

Q: When is p false after OP?
  - If WP(¬p, OP) is true before OP!
  - We know F is true before OP
  - Thm. Pvr. Query: F ⇒ WP(¬p, OP)

Predicates: LOCK, new==old
How to compute successor?

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();
}

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

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

Predicate: $new==old$

True ? $(LOCK, new==old) \Rightarrow (new + 1 = old) \quad \text{NO}$
False ? $(LOCK, new==old) \Rightarrow (new + 1 \neq old) \quad \text{YES}$
Advanced SLAM/BLAST

Too Many Predicates
  - Use Predicates Locally

Counter-Examples
  - Craig Interpolants

Procedures
  - Summaries

Concurrency
  - Thread-Context Reasoning
SLAM Summary

1) Instrument Program With Safety Policy
2) Predicates = { } 
3) Abstract Program With Predicates
   - Use Weakest Preconditions and Theorem Prover Calls
4) Model-Check Resulting Boolean Program
   - Use Symbolic Model Checking
5) Error State Not Reachable?
   - Original Program Has No Errors: Done!
6) Check Counterexample Feasibility
   - Use Symbolic Execution
7) Counterexample Is Feasible?
   - Real Bug: Done!
8) Counterexample Is Not Feasible?
   1) Find New Predicates (Refine Abstraction)
   2) Goto Line 3
Optional: SLAM Weakness

1: F() {
2:   int x=0;
3:   lock();
4:   do x++;
5:   while (x ≠ 88);
6:   if (x < 77)
7:     lock();
8: }

- Preds = {}, Path = 234567
- [x=0, ¬x+1≠88, x+1<77]
- Preds = {x=0}, Path = 234567
- [x=0, ¬x+1≠88, x+1<77]
- Preds = {x=0, x+1=88}
- Path = 23454567
- [x=0, ¬x+2≠88, x+2<77]
- Preds = {x=0,x+1=88, x+2=88}
- Path = 2345454567
- ...
- Result: the predicates “count” the loop iterations
Homework

- Read Winskel Chapter 2
- Read Hoare paper
- Read Spolsky article