Having a BLAST with SLAM
Meeting 3, CSCI 5535, Spring 2009

Announcements
• Homework 0 is out, due Jan 26
  - Use BLAST
  - Don't use the Cygwin version
  - Linux binaries work "out of the box"
• Submit on the course moodle as a PDF
  • Can be typeset or handwritten
• Forums

Software Model Checking
via Counterexample Guided
Abstraction Refinement

There are easily dozens of papers.
We will skim.

SLAM Overview/Review
• 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 `execv` is called, must have `suid ≠ 0`

Example SLAM Input

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

SLAM in a Nutshell

```c
SLAM(Program p, Spec s) =  
    Program q = incorporate_spec(p, s);  // slic  
    PredicateSet abs = {};  
    while true do  
        BooleanProgram b = abstract(q.abs);  // c2bp  
        match model_check(b) with  // beboop  
            | No_Error --> print("no bug"); exit(0)  
        | Counterexample(c) -->  
            if is_valid_path(c, p) then  // newton  
                print("real bug"); exit(1)  
            else  
                abs ← abs ∪ new_preds(c)  // newton  
        done
```

Incorporating Specs

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

Ideas?
The Safety Verification Problem

Incorporating Specs

Example {() { }
1: if lock();
2: q = q->next;
3: return();
Example {() { }
1: do{ if L=1 goto ERR;
2: else L=1;
3: q = q->next;
4: goto 1;
5: return();
Original program reveals spec iff new program reaches ERR

Program As Labeled Transition System

State
pc -> 3
lock -> 0
old -> 0
new -> 0
q = q->next;

Transition
pc = 3
lock = 0
old = 0
new = 0
q = q->next;

Example () { }
1: unlock();
2: return();
3: unlock();
4: unlock();
5: unlock();

Representing [Sets of States] as Formulas

<table>
<thead>
<tr>
<th>[F]</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>states satisfying F (s</td>
<td>s \models F)</td>
</tr>
<tr>
<td>FO formula over program vars</td>
<td></td>
</tr>
<tr>
<td>[F_1] \cap [F_2]</td>
<td>F_1 \land F_2</td>
</tr>
<tr>
<td>[F_1] \lor [F_2]</td>
<td>F_1 \lor F_2</td>
</tr>
<tr>
<td>\neg 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., lock=true)
  old = new
- States satisfying some predicates are equivalent
  - Merged into one abstract state
- Num of abstract states is finite
  - Thus model checking the abstraction will be feasible
Abstract States and Transitions

State Transition
- pc = 3
- old = new
- new = 5
- q = 0x133a

Theorem Prover
- lock old=new
- ~lock ~old=new

Abstraction

State Transition
- pc = 3
- lock = new
- old = new
- new = 5
- q = 0x133a

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

Abstraction

State Transition
- pc = 3
- lock = new
- old = new
- new = 5
- q = 0x133a

Analyse Abstraction

Analyze finite graph
- Over Approximate
- Safe \Rightarrow System Safe
- No false negatives

Problem
- Spurious counterexamples
- false positives

Idea 2: Counterexample-Guided Refinement

Solution
- Use spurious counterexamples to refine abstraction!

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
3. Repeat search until real counterexample or system proved safe

(Sources: Kurshan et al 93, Clarke et al 00, Bull-Rajman 01)

Problem: Abstraction is Expensive

Problem
#abstract states $\geq 2^#$predicates
Exponential Thm. Prover queries

Observe
Fraction of state space reachable
#Preds ~ 100’s, #States ~ 2^{100}, #Reach ~ 1000’s

Solution

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 $= 2^#$predicates
Exponential Thm. Prover queries

Solution
Don’t refine error-free regions

Key Idea for Solutions?
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.

Build-and-Search

Example: 1 2
Example: 1 2

Reachability Tree

Predicates: LOCK

Reachability Tree

Predicates: LOCK
while old = new;
lock new ++;
(q != NULL) {
Build-and-Search

Reachability Tree

Reachability Tree

Reachability Tree

Reachability Tree

Reachability Tree

Reachability Tree
Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

Repeat Build-and-Search

Example: 
1. lock();
2. old = new;
3. q = lock();
4. if (q == NULL) unlock();
5. new = new;
6. unlock();
7. if (new != old) new = old;
8. unlock();

Predicates: LOCK, new == old

Reachability Tree

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

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

Initial
Q. How to compute “successors”?  

Example: How to compute successor?

Predicates: LOCK, new \equiv old

Reachability Tree

Two Handwaves

Weakest Preconditions

WP(\textbf{P}), \textbf{OP}

Weakest formula \textbf{P} s.t.

- if \textbf{P} is true before \textbf{OP}
  - then \textbf{P} is true after \textbf{OP}

Assign $\textbf{P}(e/x)$

new+1 = old

new = old

```plaintext
Example: 1 1
look();
old = new;
if (q != NULL) {
  q = q->next;
}
unlock();
unlock();

Predicates: LOCK, new \equiv old
```

How to compute successor?

For each \textbf{P}

- Check if \textbf{P} is true (or false) after \textbf{OP}
  - If WP(\textbf{P}, \textbf{OP}) is true before \textbf{OP}
  - We know \textbf{P} is true before \textbf{OP}
  - Then, Pvr. Query: \textbf{F} = WP(\textbf{P}, \textbf{OP})
How to compute successor?

For each \( p \)
- Check if \( p \) is true (or false) after \( \text{OP} \)
- Q: When is \( p \) false after \( \text{OP} \)?
  - If \( \text{WP}(\lnot \lnot \lnot \lnot p, \text{OP}) \) is true before \( \text{OP} \)
  - We know \( F \) is true before \( \text{OP} \)
  - Thm. Par. Query: \( F \Rightarrow \text{WP}(\lnot \lnot \lnot \lnot p, \text{OP}) \)

Predicate: new == old

True? (LOCK, new==old) \(\Rightarrow\) (new + 1 == old) NO
False? (LOCK, new==old) \(\Rightarrow\) (new + 1 == old) 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

Bonus: SLAM/BLAST Weakness

1: \( f() \) {
2:  \text{int x=0;}
3:  \text{lock();}
4:  x++;
5:  while (x \neq 88);
6:  if (x < 77)
7:    lock();
8: }

\( \text{Preds} = \{\}, \text{Path} = 234567 \)
\( \{x=0, \neg x+1=88, x+1=77\} \)
\( \text{Preds} = (x=0), \text{Path} = 234567 \)
\( \{x=0, \neg x+1=88, x+1=77\} \)
\( \text{Preds} = (x=0, x+1=88) \)
\( \text{Path} = 23454567 \)
\( \{x=0, \neg x+2=88, x+2=77\} \)
\( \text{Preds} = (x=0,x+1=88,x+2=88) \)
\( \text{Path} = 23454567 \)
\( \text{Preds} = (x=0,x+1=88,x+2=88) \)
\( \text{Path} = 23454567 \)
\( \text{Preds} = (x=0,x+1=88,x+2=88) \)
\( \text{Path} = 23454567 \)

Result: the predicates "count" the loop iterations.

For Next Time

- Post about today's class and reading
- Read Winskel, Chapter 2 up to 2.6
- Read "Hints on Programming Language Design"
- Forum for Jan 26 will be on Winskel chapter and class only.