Having a BLAST with SLAM
Meeting 4, CSCI 5535, Spring 2010

Announcements

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 exec is called, must have suid != 0

Property 3: IRP Handler

Example SLAM Input

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

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

SLAM in a Nutshell

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  // bebop
| 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

Ideas?
Incorporating Specs

Example (| l |
1: do |
| l |
| l |
| l |
| l |
2: | l |
3: | l |
4: while | l |
5: | l |
6: unlock();
7: return;

Program As Labeled Transition System

State

Transition

Example (| l |
1: do |
| l |
| l |
| l |
| l |
2: | l |
3: unlock();
4: | l |
5: | l |
6: unlock();

The Safety Verification Problem

Error (e.g., states with PC = Err)

Safe States (never reach Error)

Is there a path from an initial to an error state?
Problem? Infinite state graph (old-1, old-2, old-...)
Solution? Set of states = logical formula

Representing [Sets of States] as Formulas

\[ [F] \]

states satisfying \( F \) \( (s \models F) \)

\[ F \]
FO formula over program vars

\[ [F_1] \cap [F_2] \]
\[ F_1 \wedge F_2 \]

\[ [F_1] \cup [F_2] \]
\[ F_1 \lor F_2 \]

\[ \neg F \]

\[ [F_1] \subseteq [F_2] \]
\[ F_1 \Rightarrow F_2 \]
i.e. \( F_1 \wedge \neg F_2 \) 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

\[ F \]
FO formula over program vars
Abstract States and Transitions

State | Transition
---|---

pc => 3
old => 3
new => 5
q => 0x11a

Theorem Prover

lock
old=new
~lock
~old=new

Abstraction

State | Transition
---|---

pc => 3
old => 3
new => 5
q => 0x11a

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

Abstraction

State | Transition
---|---

pc => 3
old => 3
new => 5
q => 0x11a

Theorem Prover

lock
old=new
~lock
~old=new

Analyze 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!

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 until real counterexample or system proved safe

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^#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.

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

Build-and-Search

Example

Reachability Tree

Predicates: lock
Example

1. `q = q->next;`
2. `old = new;`
3. `lock`
4. `unlock();`

Predicates: `LOCK`

Reachability Tree
Repeat Build-and-Search

Example: 1) look(old = new)
3. q = old
2. q = new
4. Build new := old;
5. Unlock();

Reachability Tree

Predicates: LOCK, new = old

Repeat Build-and-Search

Example: 1) look(old = new)
3. q = old
2. q = new
4. Build new := old;
5. Unlock();

Reachability Tree

Predicates: LOCK, new = old

Repeat Build-and-Search

Example: 1) look(old = new)
3. q = old
2. q = new
4. Build new := old;
5. Unlock();

Reachability Tree

Predicates: LOCK, new = old

Repeat Build-and-Search

Example: 1) look(old = new)
3. q = old
2. q = new
4. Build new := old;
5. Unlock();

Reachability Tree

Predicates: LOCK, new = old

Repeat Build-and-Search

Example: 1) look(old = new)
3. q = old
2. q = new
4. Build new := old;
5. Unlock();

Reachability Tree

Predicates: LOCK, new = old

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
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

How to compute successor?

Example:
1. [LOCK, new ++;]
2. 
3. [F]
4. [OP]

For each p
• Check if p is true (or 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 ++; new = old

Predicates: LOCK, new ++; new = old

Weakest Preconditions

WP(P, OP)
Weakest formula P s.t.
if P is true before OP
then P is true after OP

Assign

P(e/x)
new = old

More on this later in the semester!
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 \( \text{WP}(\neg\neg\neg\neg p, \text{OP}) \) is true before \( \text{OP} \), then \( F \) is true before \( \text{OP} \).
- We know \( F \) is true before \( \text{OP} \).

Thm. Pvr. Query:
\[ F \Rightarrow \text{WP}(\neg\neg\neg\neg p, \text{OP}) \]

Predicate: \( \text{new} = \text{old} \)

True? \( \text{LOCK, new=old} \Rightarrow (\text{new + 1} = \text{old}) \) \ NO
False? \( \text{LOCK, new=old} \Rightarrow (\text{new + 1} \neq \text{old}) \) \ 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: \text{f}() \{ \}
2: \text{int} \ x=0; \text{lock}(); \text{x++}; \text{while} (x \neq \text{88}); \text{if} (x < 77) \text{lock}(); \}
3: \text{Preds} = \{\}, \text{Path} = \text{234567}
4: [x=0, \neg x+1=\text{88}, x+\text{177}] \text{Preds} = \{x=0\}, \text{Path} = \text{234567}
5: [x=0, \neg x+1=\text{88}, x+\text{177}] \text{Preds} = \{x=0, x+1=\text{88}\}
6: [x=0, \neg x+2=\text{88}, x+2<\text{77}] \text{Preds} = \{x=0, x+1=\text{88}, x+2=\text{88}\}
7: \text{Path} = \text{23454567}
8: \text{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.