Having a BLAST with SLAM

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

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

Combining Strengths

Theorem Proving
- Need loop invariants (will find automatically)
- Behaviors encoded in logic (used to refine abstraction)
- Theorem provers (used to compute successors, refine abstraction)

Program Analysis
- Imprecise (will be precise)
- Abstraction (will shrink the state space we must explore)

Model Checking
- Finite-state model, state explosion (will find small good model)
- State Space Exploration (used to get a path sensitive analysis)
- Counterexamples (used to find relevant facts, refine abstraction)

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

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

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

![Diagram of IRP Handler](image)

**SLAM in a Nutshell**

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

**Incorporating Specs**

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

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

**Program As Labeled Transition System**

![Diagram of Labeled Transition System](image)

**The Safety Verification Problem**

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

Safe States
(never reach Error)
```

```
Initial
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**

- $F$: FO formula over program variables
- $[F_1] \cap [F_2] = F_1 \land F_2$
- $[F_1] \cup [F_2] = F_1 \lor F_2$
- $[F_1] \subseteq [F_2] = F_1 \Rightarrow F_2$

**i.e.** $F_1 \Rightarrow F_2$ unsatisfiable

**Idea 1: Predicate Abstraction**

- Predicates on program state:
  - lock
  - old = new
- States satisfying same predicates are equivalent
  - Merged into one abstract state
- Number of abstract states is finite
  - Thus model-checking the abstraction will be feasible!

**Abstract States and Transitions**

**Abstraction**

**Existential Lifting**

(i.e., $A_1 \Rightarrow A_2$ iff $\exists c_1 \in A_1. \exists c_2 \in A_2. c_1 \Rightarrow c_2$)

**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!

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 all counterexample or system proved safe

**Problem:** Abstraction is Expensive

**Problem**
#abstract states = 2^#predicates
Exponential Thm. Prover queries

**Observe**
Reachable

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: 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

```
Example

1. 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

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

Reachability Tree

Predicates: LOCK

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

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

Reachability Tree

Predicates: LOCK

Repeat Build-and-Search

Predicates: LOCK, new == old

Inconsistent

Reachability Tree

Predicates: LOCK

Analyze Counterexample

Reachability Tree

Reachability Tree

Predicates: LOCK, new == old

Repeat Build-and-Search

Predicates: LOCK

Analyze Counterexample

Reachability Tree

Reachability Tree
Repeat Build-and-Search

Reachability Tree

Predicates: \textsc{lock, new==old}

Example:
1. Pick tree-node (=abs. state)
2. Add children (=abs. successors)
3. On re-visit abs. state, cut-off

Unroll

Initial

Find min spurious suffix

- Learn new predicates
  - Rebuild subtree with new preds.

Key Idea: Reachability Tree

SAFE

Error Free

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

Q. How to compute “successors”?

Refinement

Predicates: \( \text{LOCK, } \text{new} = \text{old} \)

Two handwaves

Q. How to find predicates?

Weakest Preconditions

\( WP(P, OP) \)

Weakest formula \( P' \) s.t.
if \( P' \) is true before \( OP \)
then \( P \) is true after \( OP \)

\( WP(P, OP) \)

Weakest formula \( P' \) s.t.
if \( P' \) is true before \( OP \)
then \( P \) is true after \( OP \)

\( [P] \)

Assign

\( x = e \)

\( P[x/e] \)

\( new = new+1 \)

\( new \neq old \)
How to compute successor?

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: $F \Rightarrow WP(p, OP)$

Predicate: $\text{new=old}$

SLAM Summary

1) Instrument Program With Safety Policy
2) Predicates = {}   
3) Abstract Program With Predicates
4) Use Weakest Preconditions and Theorem Prover Calls
5) Use Symbolic Model Checking
6) Use Symbolic Execution
7) Check Counterexample Feasibility
8) Find New Predicates (Refine Abstraction)
9) Goto Line 3

Optional: SLAM Weakness

- 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 = 234545467

- Result: the predicates
  "count" the loop iterations

Advanced SLAM/BLAST

- Use Predicates Locally
- Craig Interpolants
- Summaries
- Concurrency

Too Many Predicates
Counter-Examples
Procedures

Predicate:

True? (LOCK, new=old) $\Rightarrow$ (new + 1 = old) NO
False? (LOCK, new=old) $\Rightarrow$ (new + 1 $\neq$ old) YES
Homework

- Project Status Update
- Project Due Tue Apr 25
  - You have ~14 days to complete it.
  - Need help? Stop by my office or send email.