Having a BLAST with SLAM
Building Up To: 
Software Model Checking via Counter-Example Guided Abstraction Refinement

- There are easily two dozen SLAM/BLAST/MAGIC papers; I will skim.
Where’s the Beef

• To produce the **explicit counter-example**, use the “onion-ring method”
  - A counter-example is a valid **execution path**
  - For each Image Ring (= set of states), find a state and link it with the concrete transition relation R
  - Since each Ring is “reached in one step from previous ring” (e.g., Ring#3 = EX(Ring#4)) this works
  - Each state z comes with L(z) so you know what is true at each point (= what the values of variables are)

![Diagram of labeled transition system]

This is a labeled transition system, not a program!
Key Terms

• CEGAR = Counterexample guided abstraction refinement. A successful software model-checking approach. Sometimes called “Iterative Abstraction Refinement”.

• SLAM = The first CEGAR project/tool. Developed at MSR.

• Lazy Abstraction = A CEGAR optimization used in the BLAST tool from Berkeley.

• Other terms: c2bp, bebop, newton, npackets++, MAGIC, flying boxes, etc.
So ... what is Counterexample Guided Abstraction Refinement?
- Theorem Proving?
- Dataflow Analysis?
- Model Checking?
Verification by Theorem Proving

1. Loop Invariants
2. Logical formula
3. Check Validity

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;
}

Invariant:

\[ \text{lock} \land \text{new} = \text{old} \lor \neg \text{lock} \land \text{new} \neq \text{old} \]
Verification by **Theorem Proving**

1. Loop Invariants
2. Logical formula
3. Check Validity

**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;
}
```

- Loop Invariants
- Multithreaded Programs
  + Behaviors encoded in logic
  + Decision Procedures

Precise [ESC, PCC]
Verification by Program Analysis

1. Dataflow Facts
2. Constraint System
3. Solve constraints

Example

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

- Imprecision due to fixed facts
+ Abstraction
+ Type/Flow Analyses

Scalable [CQUAL, ESP, MC]
Verification by Model Checking

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;
}

1. (Finite State) Program
2. State Transition Graph
3. Reachability

- Pgm → Finite state model
- State explosion
+ State Exploration
+ Counterexamples

Precise [SPIN, SMV, Bandera, JPF ]
One Ring To Rule Them All?
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**
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 `execv` is called, must have `suid ≠ 0`
Property 3: IRP Handler

[Diagram showing the IRP handler process with transitions between states such as start NP, start P, synch, not pending returned, prop completion, no prop completion, Mark Pending, and return Pending.]
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;
}
SLAM in a Nutshell

\[
\text{SLAM}(\text{Program } p, \text{ Spec } s) = \text{Program } q = \text{incorporate_spec}(p,s); \quad \text{// program name}
\]
\[
\text{PredicateSet } abs = \{ \}; \quad \text{// slic}
\]
\[
\text{while } \text{true } \text{do}
\]
\[
\text{BooleanProgram } b = \text{abstract}(q,abs); \quad \text{// c2bp}
\]
\[
\text{match } \text{model\_check}(b) \text{ with}
\]
\[
| \text{No\_Error } \rightarrow \text{printf(“no bug”); exit(0)} \quad \text{// bebop}
\]
\[
| \text{Counterexample}(c) \rightarrow
\]
\[
\text{if } \text{is\_valid\_path}(c, p) \text{ then}
\]
\[
\text{printf(“real bug”); exit(1)} \quad \text{// newton}
\]
\[
\text{else}
\]
\[
\text{abs } \leftarrow \text{abs } \cup \text{new\_preds}(c) \quad \text{// newton}
\]
\[
\text{done}
\]
Incorporating Specs

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

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 ();
6: 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 $\cong$ logical formula
Representing [Sets of States] as *Formulas*

| $[F]$ states satisfying $F$ \{s | s ⊨ F \} | $F$ FO fmla over prog. vars |
|-----------------|----------------------------------|
| $[F_1] \cap [F_2]$ | $F_1 \land F_2$ |
| $[F_1] \cup [F_2]$ | $F_1 \lor F_2$ |
| $\overline{[F]}$ | $\neg F$ |
| $[F_1] \subseteq [F_2]$ | $F_1 \Rightarrow F_2$ |

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 **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++; 
4: } ... 

Theorem Prover

lock
old=new
¬ lock
¬ old=new
Abstraction

Theorem Prover

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

3: unlock();
new++;
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

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**

\[
\#\text{abstract states} = 2 \#\text{predicates}
\]

Exponential Thm. Prover queries

**Solution**

Build abstraction *during* search
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
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
Build-and-Search

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

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

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();
}

Predicates: LOCK

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

Reachability Tree

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

Inconsistent
[new == old]

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:  \textit{LOCK}, \textit{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();
}

Predicates: LOCK, new==old

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
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 ();
}

Reachability Tree

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

Predicates:  LOCK, new==old

Reachability Tree
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();
}

Reachability Tree

Predicates:  \textit{LOCK, new==old}
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;
        unlock();
        new ++;
    }
4:  }while(new != old);
5:  unlock();
}

Predicates:  LOCK, new==old

Reachability Tree
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”? 

Reachability Tree

Predicates:  LOCK, new==old
Two handwaves

Q. How to compute “successors”?

Q. How to find predicates?

Refinement

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

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

$x = e$

$P[e/x]$

$P$

$new + 1 = old$

$new = new + 1$

$new = old$
How to compute successor?

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 ();
}

LOCK, new==old  3  F

OP

¬ LOCK, ¬ new = old  4  ?

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 ⇒ WP(p, OP)

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

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

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

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

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

LOCK, new==old  F

OP

¬ 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)

Predicate: new==old

True ?  (LOCK, new==old) ⇒ (new + 1 = old)  NO

False ?  (LOCK, new==old) ⇒ (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
Optional: SLAM Weakness

```c
F() {
    int x=0;
    lock();
    do x++;
    while (x != 88);
    if (x < 77)
        lock();
}
```

- Preds = {}, Path = 234567
- \([x=0, \neg x+1\neq 88, x+1<77]\)
- Preds = \(\{x=0\}\), Path = 234567
- \([x=0, \neg x+1\neq 88, x+1<77]\)
- Preds = \(\{x=0, x+1=88\}\)
- Path = 23454567
- \([x=0, \neg x+2\neq 88, x+2<77]\)
- 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 Optional papers