Having a BLAST with SLAM
Meetings 3-4, CSCI 7135, Fall 2010

Announcements
• Moodle problems?
• Blog problems?
• Looked at the syllabus on the website?

This Week: Building up to SDV
• Verify properties or find bugs in software
• Take an important program (e.g., a device driver)
• Merge it with a property (e.g., no deadlocks)
• Transform the result into a boolean program
• Use a model checker to exhaustively explore the resulting state space
  - Result 1: program provably satisfies property
  - Result 2: program violates property "right here on line 92,376!"

Model Checking
There are complete courses in model checking (see ECEN 5139, Prof. Somenzi).
Model Checking by Edmund M. Clarke, Orna Grumberg, and Doron A. Peled.
Symbolic Model Checking by Ken McMillan.
We will skim.

Hope in program analysis
Microsoft uses and distributes the Static Driver Verifier
Airbus applies the Astrée Static Analyzer

Take-Home Message
• Model checking is the exhaustive exploration of the state space of a system, typically to see if an error state is reachable. It produces concrete counterexamples.
• The state explosion problem refers to the large number of states in the model.
Keywords
Model checking is an automated technique.
Model checking verifies transition systems.
Model checking verifies temporal properties.
Model checking falsifies by generating counterexamples.

A model checker is a program that checks if a (transition) system satisfies a (temporal) property.

Verification vs. Falsification

- What is verification?
- What is falsification?

Verification vs. Falsification

- An automated verification tool
  - can report that the system is verified (with a proof);
  - or that the system was not verified.
- When the system was not verified, it would be helpful to explain why.
  - Model checkers can output an error counterexample: a concrete execution scenario that demonstrates the error.
- Can view a model checker as a falsification tool
  - The main goal is to find bugs.

**So what can we verify or falsify?**

Software Model Checking via Counterexample Guided Abstraction Refinement

There are easily dozens of papers.
We will skim.

Key Terms

- **Counterexample guided abstraction refinement (CEGAR):** A successful software model-checking approach. Sometimes called "Iterative Abstraction Refinement".
- **SLAM:** The first CEGAR project/tool.
  - Developed at MSR.
- **Lazy Abstraction:** CEGAR optimization.
  - Used in the BLAST tool from Berkeley.

**What is Counterexample Guided Abstraction Refinement (CEGAR)?**

Verification by ...
- Model Checking?
- Theorem Proving?
- Dataflow Analysis or Program Analysis?
### Verification

**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; 
)  

Is this program correct?  
What does correct mean?  
How do we determine if a program is correct?  

---

### Verification by Model Checking

**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; 
)  

1. (Finite State) Program  
2. State Transition Graph  
3. Reachability  
- Program—Finite state model  
- State explosion  
- State exploration  
- Counterexamples  

Precise [SPIN,SMV,Bandera,IFP]

---

### Verification by Theorem Proving

**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; 
)  

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

**Invariant:**  
lock \land \text{new} = \text{old}  
\lor  
\neg lock \land \text{new} \neq \text{old}

---

### Verification by Theorem Proving

**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; 
)  

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

- Loop invariants  
- Multi-threaded programs  
- Behaviors encoded in logic  
- Decision procedures  

Precise [ESC/PCC]

---

### Verification by Program Analysis

**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; 
)  

1. Dataflow Facts  
2. Constraint System  
3. Solve Constraints  
- Imprecision: fixed facts  
- Abstraction  
- Type/Flow analyses  

Scalable [Qud,ESP]

---

### Combining Strengths

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

**Program Analysis**
- 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)  

---

Program Checking
- Imprecise  
- Abstraction (will shrink the state space we must explore)  

Precise [SMART]
Revisiting SLAM

**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

First Question?

So what can we verify or falsify?

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

[Chen-Wagner-Dean '02]
**Example SLAM Input**

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

**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 → print("no bug"); exit(0)  
  | Counterexample(c) →  
      if is_valid_path(c, p) then  
          print("real bug"); exit(1)  
      else  
          abs ← abs ∪ new_preds(c)  
  done
```

**Incorporating Specs**

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

**Ideas?**

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

**Infinite Set of states (never reach ERR)**

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

**Program As Labeled Transition System**

```
State    Transition
pc ← 3  lock ← 1  done ← 5  q ← old+3a  
lock ← 1  done ← 5  q ← old+3a
```

**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
```

---

- Image 11: Example SLAM Input
- Image 12: SLAM in a Nutshell
- Image 13: Incorporating Specs
- Image 14: Incorporating Specs
- Image 15: Program As Labeled Transition System
- Image 16: The Safety Verification Problem
### Representing [Sets of States] as Formulas

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</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>$[F_1]$</td>
<td>$F_1$</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

---

### Announcements

- Please label blog entries with the relevant class date (e.g., "9-1")
  - This will help me keep track of the "minimum commenting requirement"
- I read all posts but may not respond to all of them.

---

### Having a BLAST with SLAM

Meeting 4, CSCI 7135, Fall 2010

---

### Soundness and Completeness

---

### Review
Example SLAM Input

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

Program As Labeled Transition System

State Transition
pc = 3
old = old = new
new = 5
q = NULL
pc = 4
old = old = new
new = 6
q = NULL

Representing [Sets of States] as Formulas

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([F] )</td>
<td>states satisfying ( F )</td>
</tr>
<tr>
<td>([F_1] \land [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] \implies [F_2] )</td>
<td>( F_1 \implies F_2 )</td>
</tr>
</tbody>
</table>

On to Software Model Checking

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 as logical formula

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**

- pc = 3
- lock = 0
- old = 5
- new = 5
- q = 0x112a

**Transition**

- 3: unlock
- new = 5
- q = 0x112a

Theorem Prover

- lock
- old = new
- ¬lock
- ¬old = new

**Abstraction**

State

- pc = 3
- lock = 0
- old = 5
- new = 5
- q = 0x112a

Transition

- 3: unlock
- new = 5
- q = 0x112a

Theorem Prover

- lock
- old = new
- ¬lock
- ¬old = new

Existential Lifting
(i.e., A₁ → A₂ iff ∃c₁ ∈ A₁. ∃c₂ ∈ A₂. c₁ → c₂)

Abstraction

State

- pc = 3
- lock = 0
- old = 5
- new = 5
- q = 0x112a

Transition

- 3: unlock
- new = 5
- q = 0x112a

Theorem Prover

- lock
- old = new
- ¬lock
- ¬old = new

Analyze Abstraction

- Analyze finite graph
- Over Approximate
- Safe ⇒ 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

[Kurshan et al 93][Clarke et al 00][Bull-Bajajani 01]

Problem: Abstraction is Expensive

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

Solution1: Only Abstract Reachable States

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

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.

Example:
- Rebuild subtree with new preds.

Locked
- Learn new predicates
- Rebuild subtree with new preds.

Reachability Tree

Example:
- Learn new predicates
- Rebuild subtree with new preds.
Repeat Build-and-Search

Example 1:1
1. look();
   old = new;
2. if (old != NULL) {
   new = old;
3. unlock();
4. return;
5. } else {
   new = old;
2. unlock();
3. return;
}

Reachability Tree

Predicates: LOCK, new == old

Repeat Build-and-Search

Example 1:1
1. look();
   old = new;
2. if (old != NULL) {
   new = old;
3. unlock();
4. return;
5. } else {
   new = old;
2. unlock();
3. return;
}

Reachability Tree

Predicates: LOCK, new == old

Repeat Build-and-Search

Example 1:1
1. look();
   old = new;
2. if (old != NULL) {
   new = old;
3. unlock();
4. return;
5. } else {
   new = old;
2. unlock();
3. return;
}

Reachability Tree

Predicates: LOCK, new == old

Repeat Build-and-Search

Example 1:1
1. look();
   old = new;
2. if (old != NULL) {
   new = old;
3. unlock();
4. return;
5. } else {
   new = old;
2. unlock();
3. return;
}

Reachability Tree

Predicates: LOCK, new == old

Repeat Build-and-Search

Example 1:1
1. look();
   old = new;
2. if (old != NULL) {
   new = old;
3. unlock();
4. return;
5. } else {
   new = old;
2. unlock();
3. return;
}

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
Two Handwaves

Q. How to compute "successors"?

Two Handwaves

Q. How to compute "successors"?

Weakest Preconditions

Weakest Preconditions

How to compute successor?

How to compute successor?
How to compute successor?

Example

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

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

Bonus: SLAM/BLAST Weakness

```c
f() {

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

- 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 = 234567
- `x=0, ¬x=1=88, x=1=77`
- Preds = `(x=0, x=1=88)` Path = 234567
- `x=0, ¬x=2=88, x=2=77`
- Preds = `(x=0, x=1=88, x=2=88)` Path = 234567
- ...

Result: the predicates "count" the loop iterations.