SMT for Cryptographic & Concurrent Software Verification (Part I)

Chao Wang
ECE Dept., Virginia Tech

SAT/SMT Summer School, Semmering, Austria, July 10-12, 2014
My Background

- Performance
- Reliability
- Security

VLSI Circuits
- Finite state machine
- Boolean analysis
- Model checking

Concurrent Software
- Multi-core, multi-threaded
- Concurrency
- Symbolic predictive analysis

Embedded Software
- C/C++ programs
- Numerical analysis
- Abstract interpretation

Cryptographic Software
- Security and Privacy
- Power side channel
- SMT based analysis

Univ. Colorado, Boulder 2000-2004
NEC Labs, Princeton 2004-2011
Virginia Tech, Blacksburg 2011-present
Today’s Talk

- Performance
- **Reliability**
- **Security**

**Embedded Software**
- C/C++ program
- Numerical analysis
- Abstract interpretation

**Concurrent Software**
- Multi-core, multi-threaded
- Concurrency
- **Symbolic predictive analysis**

**Cryptographic Software**
- Security and Privacy
- Power side channel
- SMT based analysis

**VLSI Circuits**
- Finite state machine
- Boolean analysis
- Model checking

Univ. Colorado, Boulder 2000-2004

NEC Labs, Princeton 2004-2011

Virginia Tech, Blacksburg 2011-present
Today’s Talk

- Performance
- **Reliability**
- Security

**Embedded Software**
- C/C++ program
- Numerical analysis
- Abstract interpretation

**Concurrent Software**
- Multi-core, multi-threaded
- Concurrency
- **Symbolic predictive analysis**

**Cryptographic Software**
- Security and Privacy
- Power side channel
- SMT based analysis

**VLSI Circuits**
- Finite state machine
- Boolean analysis
- Model checking

Univ. Colorado, Boulder 2000-2004

NEC Labs, Princeton 2004-2011

Virginia Tech, Blacksburg 2011-present
Concurrency: Why?

• “Moore says Moore’s Law to hit wall.” CNET News 1997
  – “If current trends in microprocessors were to continue, by 2010 they’d burn as hot as the surface of the sun.” Intel engineers 1999

• Can’t make it go faster? Stuff more CPU cores!
Everyone’s Happy

Tilera (2007)

**TILE64 Processor**

The TILE64™ family of multicore processors delivers immense compute performance to drive the latest generation of embedded applications.

<table>
<thead>
<tr>
<th>Multicore Development Environment Options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• ANSI standard C/C++ compiler</td>
<td></td>
</tr>
<tr>
<td>• Advanced profiling and debugging designed for multicore programming</td>
<td></td>
</tr>
<tr>
<td>• Supports SMP Linux with 2.6 kernel</td>
<td></td>
</tr>
<tr>
<td>• TMC libraries for efficient inter-tile communication</td>
<td></td>
</tr>
</tbody>
</table>
Everyone’s Happy… except programmers

Apps

Programming Models & Languages
Compilers/Auto-Tuners/Tools
Runtime Systems

Processors

Program Analysis
Outline

• Embedded Systems: Why?
• Concurrent Software: Why?

• Symbolic Predictive Analysis
  – Part I: Monitoring
  – Part II: Predicting

• Future Research Vision
Challenging Problem (in industry today)

Your expectation:
If the program fails the given test, you want to see the bug

Reality:
Even if the program may fail, you perhaps won’t see it

Why?
Thread scheduling is controlled by the OS and the Pthreads library
The Monitoring Approach

We control scheduling!

We repeatedly run the program till all possible scenarios are covered

Tools like this:
VeriSoft, CHESS, Inspect, ...
Thread #1

... 
s1: tmpx = data1;
    ...
 s2: data2 = tmpx+1;

Thread #2

... 
s3: tmpy = data2;
    ...
 s4: data1 = tmpy-1;

Instrumentation
(source code injection)
### Serialize an execution (interleaving)

**Scheduling (Master)**

<table>
<thead>
<tr>
<th>Interleaving 1:</th>
<th>Interleaving 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thr#1: s1</td>
<td>Thr#2: s3</td>
</tr>
<tr>
<td>Thr#1: s2</td>
<td>Thr#2: s4</td>
</tr>
<tr>
<td>Thr#2: s3</td>
<td>Thr#1: s1</td>
</tr>
<tr>
<td>Thr#2: s4</td>
<td>Thr#1: s2</td>
</tr>
<tr>
<td>data1 = 0</td>
<td>data1 = -1</td>
</tr>
<tr>
<td>data2 = +1</td>
<td>data2 = 0</td>
</tr>
</tbody>
</table>

**Thread #1**

```
...  
s1: ask_master("I'm #1, can I go?");
tmpx = data1;
...

s2: ask_master("I'm #1, can I go?");
data2 = tmpx+1;
```

**Thread #2**

```
...  

s3: ask_master("I'm #2, can I go?");
tmpy = data2;
...

s4: ask_master("I'm #2, can I go?");
data1 = tmpy-1;
```
Concurrency Bugs

- Incorrect uses of shared resources
Deadlock

(Bridge Crossing Example)
Deadlock

Both threads get stuck waiting for locks held by others
Data Race

Un-protected accesses to shared variable (data2)
Data Race

(Bridge Crossing Example)
Atomicity Violation

The red cars must stay together…
Atomicity Violation

The “check” and “use” should not be interrupted

```
If (p != NULL) {
    Write to p->x;
}
p := NULL;
```
For detecting assertion failures

Interrupt Service Routine (ISR) of a 16-bit microcontroller

```c
212 main(void) {
222    for (;;) {
224       test_flag++;
225
226       if (test_flag == 0x0a) {
227          test_flag = 0x00;
230
231       } }
232    assert(0x0a > test_flag)
233 }
```

Therac 25 radiation therapy machine 1985-1987, six accidents of massive overdose (100 times) due to a concurrency bug
For detecting assertion failures

Interrupt Service Routine (ISR) of a 16-bit microcontroller

```c
212 main(void) {
    ...
    222    for (;;) {
        223        test_flag++;
        225
        226        if(test_flag == OxOa) {
            228            test_flag = Ox00;
            230
            231            assert(OxOa > test_flag)
        }  
    232}
}
```

Found a data race between
(tid=0, obj_write , obj_id=7, obj_val=0) and
(tid=1, obj_write , obj_id=7, obj_val=0)

......

@@@: Caught Assert Exception

******************************************************************************
Inspect: Total number of runs: 9, killed-in-the-middle runs: 0
Inspect: Transitions explored: 1837
Inspect: Used time (seconds): 2.506451

******************************************************************************

code-transformation to build & run the SW on 32-bit Linux
(original code runs on bare-metal w/o OS)

INSPECT: Developed by
Yu Yang et al., Univ. Utah
For detecting concurrency bugs

• Incorrect uses of shared resources
  – Deadlocks,
  – Data races,
  – Atomicity violations,
  – Order violations,
  – Type-state violations,
  – Linearizability violations,
  – etc.
Problem: too many interleavings

• Exponential in the worst case …

• Partial Order Reduction (POR)
  – Classify interleavings into several equivalence classes, and from each class, test only one “representative”…
More scalable

More general but less scalable

SPIN [Holzmann 1980, 1991]

Verisoft [Godefroid 1997, 2005]

CHESS [Musuvathi et al, 2007]

Inspect [Yang et al, 2008]

Java Pathfinder [Visser et al, 2004]

SPIN [Holzmann 1980, 1991]

 POR

Dynamic POR & context bounding

Symbolic Pruning (beyond POR)

More scalable

More general but less scalable
POR not good enough: Why?

- Equivalence is based on “data conflicts”

They are not equivalent!
Symbolic Pruning: main idea

- Tracking “flow of data” to assess impact

A[x]=5 initially

Interleaving #1

<table>
<thead>
<tr>
<th>a = A[x] if (a&gt;0) {…}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[x] = 10</td>
</tr>
</tbody>
</table>

(a>0) is true

Interleaving #2

<table>
<thead>
<tr>
<th>a = A[x] if (a&gt;0) {…}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[x] = 10</td>
</tr>
</tbody>
</table>

(a>0) is true

They Are Still Equivalent!
Experiments (Symbolic Pruning vs. POR)

Example from DPOR paper [Flanagan & Godefroid, POPL 2005]
Concurrent threads writing to a shared “hash table”

[Wang et al. FSE 2009]
Monitoring: ...useful but ...

• Still too expensive for larger applications
Outline

• Embedded Systems Software: Why?
• Concurrent Programs: Why?

• Symbolic Predictive Analysis
  – Part I: Monitoring
  – Part II: Predicting
• Conclusions & Future Work
Predictive Analysis

• Detecting bugs by statically analyzing a logged execution trace without re-running the program…
Predictive Analysis

Search Space

Given Trace

Predictive Model

Inferred Trace

(cf. K. Sen, G. Rosu)
Predicting Errors: How?

- Given run is good, but …

Thread1

```c
e1: if (buf_index + len < BUFFSIZE) {
    ... 
}

e2: memcpy(buf[buf_index], log, len);
```

Thread2

```
e3: buf_index + =len;
```
Predicting Errors: How?

• Given run is good, but an alternative run is buggy

Thread1

e1: if ( buf_index + len < BUFFSIZE) {
    ...

  e2: memcpy(buf[buf_index], log, len);
}  

Thread2

  e3: buf_index += len;

Buffer Overflow (caused by a TOCTTOU error) is a major security vulnerability – responsible for numerous software exploits in the past decade.
Two Existing Approaches
(1) Happens-Before Methods

- Start with totally ordered events (the given trace)
  - Try to relax some of these ordering constraints
More real traces

Fewer real traces

Trace (total order)

Happens-before [Lamport 1978]

MCM (Maximum Causal Model)
[Serbanuta, Chen, Rosu 2008]

[Chen, Rosu 2007]

[Sen, Rosu 2003]

Higher Coverage
(2) Lock-Set Based Methods

- Start with **un-ordered events** (of the given trace)
  - Use lock/unlock to impose ordering constraints
More bogus traces

Lower Precision

Fewer bogus traces

Higher Precision

UCG (Universal Causality Graph) [Kahlon & Wang, CAV 2010]

Lock Acquisition History [Kahlon 2005]

Lock-set [Savage 1997]

Trace (not ordered)
Universal Causality Graph (UCG)

- For deciding “control-state reachability”
  - Sound and complete for 2 threads
  - Sound for >2 threads

[Kahlon & Wang, CAV 2010]
Experiments (UCG is more precise)

Detecting Atomicity Violations [Kahlon & Wang, CAV 2010]

Number of warnings

![Bar chart showing number of warnings for different test cases. The chart compares lockset, must-hb, and UCG, with UCG being the most precise.]
public class Data{
    public static int x;
    public static boolean s= false;

    public static void main(String[] args){
        Class o = Data.class;
        Thread t2 = new Thread (new Task());
t2.start();
Data.x = 1;
synchronized (o){
    Data.x = 0;
if(!Data.s){
    try{
        o.wait();
    }catch(Exception ex){}
}
}
}    }
}

class Task implements Runnable {
    Class o = Data.class;
public void run() {
    int a,b;
synchronized (o){
        a = Data.x;
        Data.s = true;
o.notifyAll();
    }
    b = Data.x;
}
}
public class Data{
    public static int x;
    public static boolean s= false;
    public static void main(String[] args){
        Class o = Data.class;
        Thread t2 = new Thread (new Task());
        t2.start();
        Data.x = 1;
        synchronized (o){
            Data.x = 0;
            if(!Data.s){
                try{
                    o.wait();
                }catch(Exception ex){}
            }
        }
    }
}

class Task implements Runnable {
    Class o = Data.class;
    public void run() {
        int a,b;
        synchronized (o){
            a = Data.x;
            Data.s = true;
            o.notifyAll();
        }
        b = Data.x;
    }
}

Running Example

[Said, Wang, Sakallah,&Yang, NFM 2011]
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
</tr>
</tbody>
</table>
Thread 1

```java
t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}
```

Thread 2

```java
synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
```

Execution Trace

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>Value</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>
t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>Value</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td></td>
<td>Thread 2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>
Thread 1

```java
t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}
```

Thread 2

```java
synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
```

Execution Trace

<table>
<thead>
<tr>
<th></th>
<th>Operation</th>
<th>Type</th>
<th>Value</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o) {
    a = x;
    s = true;
    o.notifyAll();
}

b = x;

Execution Trace

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td></td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
  x = 1;
  synchronized (o) {
    x = 0;
    if(!s)
      o.wait();
  }

Thread 2

synchronized (o) {
  a = x;
  s = true;
  o.notifyAll();
}

b = x;

Execution Trace

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
x = 1;
synchronized (o) {
  x = 0;
  if(!s)
    o.wait();
}

Thread 2

synchronized (o){
  a = x;
  s = true;
  o.notifyAll();
}
b = x;

Execution Trace

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
</tbody>
</table>
t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}
synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>notifyall</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

Execution Trace

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>notifyall</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>release</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>
Thread 1

t2.start();
x = 1;
synchronized (o) {
    x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
    a = x;
    s = true;
    o.notifyAll();
}
b = x;

<table>
<thead>
<tr>
<th></th>
<th>fork</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>notifyAll</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>release</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>
```java
Thread 1

t2.start();
x = 1;
synchronized (o) {
x = 0;
    if(!s)
        o.wait();
}

Thread 2

synchronized (o){
a = x;
s = true;
o.notifyAll();
}
b = x;
```

**Execution Trace**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fork</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>notifyall</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>release</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>release</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>

This trace may not be buggy
Execution Analysis

Symbolic Predictive Analysis
Partial Order

\[
E_0 < E_1 < E_2 < E_3 < E_4 < E_5 < E_{12}
\]
&
\[
E_6 < E_7 < E_8 < E_9 < E_{10} < E_{11}
\]
&
\[
E_0 < E_6
\]
Write-Read Consistency

\[ E_7 < E_1 \quad \text{OR} \quad E_3 < E_7 \]
### Synchronization Consistency

[Said, Wang, Sakallah, & Yang, NFM 2011]

<table>
<thead>
<tr>
<th>Event</th>
<th>Encoding</th>
<th>Encoding with Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>e3</td>
<td>$(a_3 &lt; a_4 \land a_2 &lt; a_5) \lor ((a_8 &lt; a_7) \land (a_5 &lt; a_3 \lor a_2 &lt; a_5))$</td>
<td>$(a_7 &lt; a_3) \lor (a_8 &lt; a_7)$</td>
</tr>
<tr>
<td>e4</td>
<td>$(a_5 &lt; a_4) \land (a_4 &lt; a_5 \lor a_2 &lt; a_5) \land (a_5 &lt; a_2 \lor a_4 &lt; a_5)$</td>
<td>$(a_4 &lt; a_4) \land (a_5 &lt; a_4 \lor a_2 &lt; a_5)$</td>
</tr>
<tr>
<td></td>
<td>$(a_5 &lt; a_2 \lor a_4 &lt; a_5) \land (a_5 &lt; a_4 \lor a_2 &lt; a_5)$</td>
<td>$(a_5 &lt; a_4) \land (a_5 &lt; a_4)$</td>
</tr>
<tr>
<td></td>
<td>$(a_5 &lt; a_4 \lor a_2 &lt; a_5) \land (a_5 &lt; a_4 \lor a_2 &lt; a_5)$</td>
<td>$(a_5 &lt; a_4)$</td>
</tr>
<tr>
<td>e7</td>
<td>$(a_7 &lt; a_4) \land (a_4 &lt; a_7 \lor a_4 &lt; a_4)$</td>
<td>$(a_7 &lt; a_4)$</td>
</tr>
<tr>
<td>e5</td>
<td>$(a_5 &lt; a_2 \land a_2 &lt; a_5 \land a_5 &lt; a_4 \land a_5 &lt; a_10) \lor (a_4 &lt; a_5) \land (a_2 &lt; a_4 \lor a_5 &lt; a_2)$</td>
<td>$(a_5 &lt; a_2) \lor (a_10 &lt; a_5) \lor (a_4 &lt; a_5)$</td>
</tr>
<tr>
<td></td>
<td>$(a_5 &lt; a_2) \land (a_2 &lt; a_4 \lor a_5 &lt; a_2) \land (a_5 &lt; a_2) \land (a_2 &lt; a_4 \lor a_5 &lt; a_2)$</td>
<td>$(a_4 &lt; a_5)$</td>
</tr>
<tr>
<td></td>
<td>$(a_5 &lt; a_2) \land (a_2 &lt; a_4 \lor a_5 &lt; a_2) \land (a_5 &lt; a_2) \land (a_2 &lt; a_4 \lor a_5 &lt; a_2)$</td>
<td>$(a_4 &lt; a_5)$</td>
</tr>
<tr>
<td>e7</td>
<td>$(a_9 &lt; a_7) \land (a_2 &lt; a_5 \lor a_7 &lt; a_2) \land (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5)$</td>
<td>$(a_2 &lt; a_5 \lor a_7 &lt; a_2) \land (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5)$</td>
</tr>
<tr>
<td>e8</td>
<td>$(a_7 &lt; a_7) \land (a_2 &lt; a_5 \lor a_7 &lt; a_2) \land (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5)$</td>
<td>$(a_2 &lt; a_5 \lor a_7 &lt; a_2) \land (a_4 &lt; a_5 \lor a_7 &lt; a_4) \lor (a_5 &lt; a_5 \lor a_7 &lt; a_5)$</td>
</tr>
<tr>
<td>e10</td>
<td>$(a_5 &lt; a_10) \land (a_5 &lt; a_4 \lor a_10 &lt; a_5) \lor (a_8 &lt; a_4 \lor a_10 &lt; a_8)$</td>
<td>$(a_5 &lt; a_4 \lor a_10 &lt; a_5)$</td>
</tr>
</tbody>
</table>
Existence of Bugs

- E1 (Write X) and E11 (Read X) has a potential data race

- Adding two happens-before constraints

\[ E_{11} < E_0 \quad \text{AND} \quad E_1 < E_{10} \]
Constraints

\[\alpha_\pi \equiv \left( \bigwedge_{t=1}^{T} (o_{e_1^t}.idx < \cdots < o_{e_n^t}.idx) \bigwedge_{e \in \text{FORK}} (o.e.idx < o_{(t.e.val).first}.idx) \right) \wedge (o_{(t.e.val).last}.idx < o.e.idx) \]

\[\beta_\pi \equiv \bigwedge_{e \in \pi \land e \text{ type - read}} \left( (e.tiwp = \text{null}) \wedge (e.val = e.var.init) \bigwedge_{e_1 \in e.pws} (o.e.idx < o_{e_1}.idx) \right) \vee \left( (o_{e_1}.idx < o.e.idx) \bigwedge_{e_2 \in e.pws \land e_2 \neq e_1} (o.e.idx < o_{e_2}.idx) \vee o_{e_2}.idx < o.e.idx \right) \]

\[\rho_\pi \equiv \bigvee_{(e_1, e_2) \in \text{PDR}} ((o_{e_1}.idx < o_{e_2}.idx < o_{e_2}.idx) \wedge (o_{e_2}.idx < o.e.idx < o_{e_2}.idx)) \]
## Data-Race / Atomicity Violation

<table>
<thead>
<tr>
<th>1</th>
<th>fork</th>
<th>-</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>acquire</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>write</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>read</td>
<td>s</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>wait</td>
<td>o</td>
<td>-</td>
</tr>
</tbody>
</table>

**Normal execution without data race as there are no conflict access enabled at the same time**

<table>
<thead>
<tr>
<th>2</th>
<th>acquire</th>
<th>o</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>write</td>
<td>s</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>notifyall</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>release</td>
<td>o</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>

**Witness is a valid execution with conflicting accesses enabled simultaneously**

| 1 | release | o | - |
More bogus traces

More real traces

Fewer real traces

Fewer bogus traces

Higher Precision

Happens-before [Lamport 1978]

MCM (Maximum Causal Model) [Serbanuta, Chen, Rosu 2008]

[Chen, Rosu 2007]

[Sen, Rosu 2003]

MCM

(Happens-before [Lamport 1978])

UCG (Universal Causality Graph) [Kahlon & Wang, CAV 2010]

Lock Acquisition History [Kahlon 2005]

Lock-set [Savage 1997]

CTP (Concurrent Trace Program) [Wang et al., FM 2009]

Trace (not ordered)

Higher Coverage

Trace (total order)
Outline

• Embedded Systems: Why?
• Concurrent Programs: Why?

• **Symbolic Predictive Analysis**
  – Part I: Monitoring
  – Part II: Predicting (CTP)

• Fusion Research Vision
C/C++ program: multi-threaded, using Pthreads

Execution trace

“assume( c )” means if (c)-branch is taken
CTP (concurrent trace program)

Think of them as bounded, straight-line, sequential programs

Execution trace

Thread $T_0$:
- $t_0: x=0, y=0$
- $t_1: fork(1)$
- $t_2: fork(2)$
- $t_{11}: a=y$
- $t_{12}: (a = 0)$
- $t_{13}: x=1$
- $t_{14}: a=x+1$
- $t_{15}: x=a$
- $t_3: assert(x \neq y)$
- $t_4$ join(1)
- $t_{27}: y=0$
- $t_{28}$

Thread $T_1$:
- $t_{12}: (a = 0)$
- $t_{13}: x=1$
- $t_{14}: a=x+1$
- $t_{15}: x=a$
- $t_{18}$
- $t_{11}: a=y$
- $t_{12}: assert(a=0)$
- $t_{27}: y=0$
- $t_{28}$

Thread $T_2$:
- $t_{21}: b=x$
- $t_{26}: (b \neq 0)$
- $t_{27}: y=0$
- $t_{28}$
Precise for Detecting Bugs

- Bugs found in CTP are always real bugs
  - Concurrency bugs
Why is CTP a Good Model?

CTP is like a ...
Symbolic Analysis

- Capturing valid executions in SAT/SMT formulas
  - Relations over variables/events
  - Not interleaved traces

- Decide formulas
  - Satisfiable \(\rightarrow\) Found a bug
  - Unsatisfiable…

[Wang et al., FM09]
[Wang et al., TACAS10]
Constructing Formulas

\[ F_{po} \]:
\[
HB(t_0,t_1) & \& HB(t_1,t_2) & \& HB(t_2,t_3) & \& HB(t_3,t_4) & \& HB(t_4,t_5) \\
HB(t_0,t_11) & \& HB(t_11,t_12) & \& HB(t_12,t_13) & \& HB(t_13,t_14) & \& HB(t_14,t_15) & \& HB(t_15,t_18) & \& HB(t_18,t_5) \\
HB(t_1,t_21) & \& HB(t_21,t_26) & \& HB(t_26,t_27) & \& HB(t_27,t_28) & \& HB(t_28,t_4)
\]

\[ F_{vd} \]:
\[
(x_0=0) & \& (y_0=0) & \\
(a_1=Y_1) & \& (b_1 = X_1) \\
(a_1=0) & \& (b_1 \neq 0) \\
(x_1=1) & \& (y_1 = 0) \\
(a_2=X_2+1) & \\
(x_2=a_2)
\]

\[ F_{property} \]:
\[
(X_3 == Y_2)
\]

\[ F_{pi} \]:
\[
(& (Y_1=y_0) & \& HB(t_{11},t_{27}) \ OR \\
(Y_1=y_1) & \& HB(t_{27},t_{11}) ) \\
& (X_1=x_0) & \& HB(t_{21},t_{13}) \ OR \\
&(X_1=x_1) & \& HB(t_{13},t_{21}) & \& HB(t_{21},t_{15}) \ OR \\
&(X_1=x_2) & \& HB(t_{15},t_{21}) ) \\
& (X_2=x_1) \\
& (X_3=x_2) \\
& (Y_2=y_1)
\]

[Wang et al., FM09]
[Wang et al., TACAS10]
Our Approach (constraint-based)

• Flexible
  – Everything boils down to a set of “constraints”
Our Approach (constraint-based)

- Flexible

- Efficient
  - SMT solver based symbolic search
Our Approach (constraint-based)

• Flexible

• Efficient

• Precise
  – It’s a precise analysis (data flow + control flow)
Experiments (tracking data flow is better)

Detecting Atomicity Violations

Number of warnings

Different Test Cases

[w/o Data]

(Symbolic)
Constraint-Based Approach

- Flexible
- Efficient
- Precise
- Compositional
  - Encode each thread, and then compose threads with “interference constraints”
Composing Incrementally

\( F_{po} \):  
\[ HB(t_0,t_1) & HB(t_1,t_2) & HB(t_2,t_3) & HB(t_3,t_4) & HB(t_4,t_5) \]
\[ HB(t_0,t_{11}) & HB(t_{11},t_{12}) & HB(t_{12},t_{13}) & HB(t_{13},t_{14}) & HB(t_{14},t_{15}) & HB(t_{15},t_{18}) & HB(t_{18},t_5) \]
\[ HB(t_1,t_{21}) & HB(t_{21},t_{26}) & HB(t_{26},t_{27}) & HB(t_{27},t_{28}) & HB(t_{28},t_4) \]

\( F_{vd} \):  
\[ (x_0=0) & (y_0=0) \]
\[ (a_1=Y_1) & (b_1 = X_1) \]
\[ (a_1=0) & (b_1 != 0) \]
\[ (x_1=1) & (y_1 = 0) \]
\[ (a_2=X_2+1) \]
\[ (x_2=a_2) \]

\( F_{property} \):  
\[ (X_3 == Y_2) \]

\( F_{pi} \):  
\& (Y_1=y_0) \& HB(t_{11},t_{27}) \OR
\[ (Y_1=y_1) \& HB(t_{27},t_{11}) \]
\& (X_1=x_0) \& HB(t_{21},t_{13}) \OR
\[ (X_1=x_1) \& HB(t_{13},t_{21}) \& HB(t_{21},t_{15}) \]
\[ (X_1=x_2) \& HB(t_{15},t_{21}) \]
\& (X_2=x_1)
\& (X_3=x_2)
\& (Y_2=y_1)

Interference constraints
Compositionality is useful

- Example
  - Add “interference constraints” on a need-to basis
More bogus traces

More real traces

Fewer real traces

Fewer bogus traces

Trace (total order)

Interference Abstraction [Sinha & Wang, POPL 2011]

Higher Precision

Good Enough

Trace (not ordered)

CTP

More bogus traces

More real traces

Higher Coverage

Fewer real traces

Trace (not ordered)
Experiments (iterative refinement is much faster)

[Sinha & Wang, POPL 2011]
Conclusions

- Concurrent Trace Program (CTP)

- Symbolic Predictive Analysis

Static Analysis (UCG)  Symbolic Analysis (SMT-based)  Interference Abs (Ref)
Related Work

• Some other symbolic analysis methods
  – [Sinha & Wang, FSE 2010],
  – [Sinha & Wang, POPL 2011]
  – [Alglave, Kroening & Tautschnig, CAV 2013]
  – …

• Applications of “symbolic predictive analysis”
  – HW deterministic replay under TSO [Lee et al., HPCA11]
  – SW deterministic replay [Huang & Zhang, PLDI’13]
  – Null pointer detection [Farzan et al., FSE’12]
  – Deadlock detection [Eslamimehr & Palsberg, FSE’14]
  – …
Our Recent Work *(brief overview)*

- Detecting Concurrency Failures
- **Mitigating** Concurrency Failures
- Optimizing Synchronization Code
Lock Removal (Ex.1)

<table>
<thead>
<tr>
<th></th>
<th>a0:  ------;</th>
<th>b0:  ------;</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1:</td>
<td>lock(l₁);</td>
<td>b1:  a[10]++;</td>
</tr>
<tr>
<td>a3:</td>
<td>unlock(l₁);</td>
<td>b3:  lock(l₁);</td>
</tr>
<tr>
<td>a4:</td>
<td>a[2]++;</td>
<td>b4:  lock(l₂);</td>
</tr>
<tr>
<td>a5:</td>
<td>lock(l₁);</td>
<td>b5:  a[12]++;</td>
</tr>
<tr>
<td>a6:</td>
<td>lock(l₂);</td>
<td>b6:  unlock(l₂);</td>
</tr>
<tr>
<td>a7:</td>
<td>a[3]++;</td>
<td>b7:  unlock(l₂);</td>
</tr>
<tr>
<td>a8:</td>
<td>unlock(l₂);</td>
<td>b8:  sh++;</td>
</tr>
<tr>
<td>a9:</td>
<td>unlock(l₁);</td>
<td>b9:  ------;</td>
</tr>
<tr>
<td>a10:</td>
<td>sh++;</td>
<td></td>
</tr>
</tbody>
</table>

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Ex.1)

a0:  ------;
a1:  lock(l_1);
a2:  a[1]++;
a3:  unlock(l_1);
a4:  a[2]++;
a5:  lock(l_1);
a6:  lock(l_2);
a7:  a[3]++;
a8:  unlock(l_2);
a9:  unlock(l_1);
a10: sh++;

b0:  ------;
b1:  a[10]++;
b3:  lock(l_1);
b4:  lock(l_2);
b5:  a[12]++;
b6:  unlock(l_2);
b7:  unlock(l_2);
b8:  sh++;
b9:  ------;

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Ex. 1)

a0:  ------;
a1:  lock(l

a2:  a[1]++;
a3:  unlock(l

a4:  ------;
a5:  lock(l

a6:  lock(l

a7:  a[3]++;
a8:  unlock(l

a9:  unlock(l

a10: sh++;

b0:  ------;
b1:  a[10]++;
b3:  lock(l

b4:  lock(l

b5:  a[12]++;
b6:  unlock(l

b7:  unlock(l

b8:  sh++;
b9:  ------;

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Ex. 1)

a0: -------;
a1: lock(l₁);
a2: -------;
a3: unlock(l₁);
a4: -------;
a5: lock(l₁);
a6: lock(l₂);
a7: -------;
a8: unlock(l₂);
a9: unlock(l₁);
a10: sh++;  

b0: -------;
b1: -------;
b2: -------;
b3: lock(l₁);
b4: lock(l₂);
b5: -------;
b6: unlock(l₂);
b7: unlock(l₂);
b8: sh++;  

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Ex. 1)

a0: 

a1: lock(l₁);
a2: 

a3: unlock(l₁);
a4: 

a5: lock(l₁);
a6: lock(l₂);
a7: 

a8: unlock(l₂);
a9: unlock(l₁);
a10: sh++;

b0: 

b1: 

b2: 

b3: lock(l₁);
b4: lock(l₂);
b5: 

b6: unlock(l₂);
b7: unlock(l₂);
b8: sh++;
b9: 

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
<table>
<thead>
<tr>
<th></th>
<th>a0</th>
<th>b0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a2</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a3</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a4</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a5</td>
<td>lock(l₁);</td>
<td></td>
</tr>
<tr>
<td>a6</td>
<td>lock(l₂);</td>
<td></td>
</tr>
<tr>
<td>a7</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>a8</td>
<td>unlock(l₂);</td>
<td></td>
</tr>
<tr>
<td>a9</td>
<td>unlock(l₁);</td>
<td></td>
</tr>
<tr>
<td>a10</td>
<td>sh++;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>b1</th>
<th>b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b3</td>
<td>lock(l₁);</td>
<td></td>
</tr>
<tr>
<td>b4</td>
<td>lock(l₂);</td>
<td></td>
</tr>
<tr>
<td>b5</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>b6</td>
<td>unlock(l₂);</td>
<td></td>
</tr>
<tr>
<td>b7</td>
<td>unlock(l₂);</td>
<td></td>
</tr>
<tr>
<td>b8</td>
<td>sh++;</td>
<td></td>
</tr>
<tr>
<td>b9</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Lock Removal (Ex.1)

a0: --------;
a1: --------;
a2: --------;
a3: --------;
a4: --------;
a5: --------;
a6: --------;
a7: --------;
a8: --------;
a9: --------;
a10: sh++; b0: --------;
b1: --------;
b2: --------;
b3: --------;
b4: --------;
b5: --------;
b6: --------;
b7: --------;
b8: sh++; b9: --------;

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Example 2)

a0:  \texttt{sh++;}

a1:  -------;

a2:  \texttt{lock(l_1);}  

a3:  -------;

a4:  \texttt{unlock(l_1);} 

a5:  -------;

a6:  -------;

a7:  \texttt{sh++;}

a8:  -------;

a9:  -------;

b0:  -------;

b1:  \texttt{lock(l_1);}  

b2:  \texttt{sh = 0;}

b3:  -------;

b4:  \texttt{x = sh;}

b5:  -------;

b6:  \texttt{unlock(l_1);}  

b7:  -------;

b8:  \texttt{assert(x != 2);} 

b9:  -------;

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Example 2)

a0: sh++;
a1: -------;
a2: lock(l_1);
a3: -------;
a4: unlock(l_1);
a5: -------;
a6: -------;
a7: sh++;
a8: -------;
a9: -------;

b0: -------;
b1: lock(l_1);
b2: sh = 0;
b3: -------;
b4: x = sh;
b5: -------;
b6: unlock(l_1);
b7: -------;
b8: assert(x != 2);
b9: -------;

Cannot be removed although it seems to be doing nothing

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Example 2)

Cannot be removed although it seems to be doing nothing

a0: \texttt{sh++;}
a1: -------;
a2: \texttt{lock(l_1);}  \quad \texttt{assert(x \neq 2);}
a3: -------;
a4: \texttt{unlock(l_1);}  \quad \texttt{b0: -------;}
a5: -------;
a6: -------;
a7: \texttt{sh++;}
a8: -------;
a9: -------;

b0: -------;
b1: \texttt{lock(l_1);}  \quad \texttt{b1: lock(l_1);}  \quad \texttt{b0: -------;}
b2: \texttt{sh = 0;}
b3: -------;
b4: \texttt{x = sh;}
b5: -------;
b6: \texttt{unlock(l_1);}  \quad \texttt{unlock(l_1);}  \quad \texttt{b6: unlock(l_1);}  \quad \texttt{b5: -------;}
b7: -------;
b8: \texttt{assert(x \neq 2);}
b9: -------;

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]
Lock Removal (Example 2)

a0: \text{sh}++;  
a1: \text{------};  
a2: \text{\underline{lock}(l_1)};  
a3: \text{------};  
a4: \text{\underline{unlock}(l_1)};  
a5: \text{------};  
a6: \text{------};  
a7: \text{\underline{sh}++};  
a8: \text{------};  
a9: \text{------};  

b0: \text{------};  
b1: \text{\underline{lock}(l_1)};  
b2: \text{\underline{sh} = 0};  
b3: \text{------};  
b4: \text{x = sh};  
b5: \text{------};  
b6: \text{\underline{unlock}(l_1)};  
b7: \text{------};  
b8: \text{\underline{assert}(x != 2)};  
b9: \text{------};

Cannot be removed although it seem to be doing nothing.
Results: for improving predictive bug detection

Lock Removal in Concurrent Trace Programs [Kahlon & Wang, CAV 2012]

<table>
<thead>
<tr>
<th>Concurrent Trace Program</th>
<th>Lock Removal Computation</th>
<th>Symbolic Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>p-avs r-avs pre(s) post(s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mem</td>
</tr>
<tr>
<td>name</td>
<td>thrs</td>
<td>events</td>
</tr>
<tr>
<td>ra.Main</td>
<td>3</td>
<td>55</td>
</tr>
<tr>
<td>connect</td>
<td>4</td>
<td>97</td>
</tr>
<tr>
<td>hedcex</td>
<td>1</td>
<td>122</td>
</tr>
<tr>
<td>liveness</td>
<td>7</td>
<td>283</td>
</tr>
<tr>
<td>BarrierB1</td>
<td>10</td>
<td>653</td>
</tr>
<tr>
<td>BarrierB2</td>
<td>13</td>
<td>805</td>
</tr>
<tr>
<td>account1</td>
<td>11</td>
<td>902</td>
</tr>
<tr>
<td>phito</td>
<td>6</td>
<td>1141</td>
</tr>
<tr>
<td>account2</td>
<td>21</td>
<td>1747</td>
</tr>
<tr>
<td>Daisy1</td>
<td>3</td>
<td>2998</td>
</tr>
<tr>
<td>Elevator1</td>
<td>4</td>
<td>3004</td>
</tr>
<tr>
<td>Elevator2</td>
<td>4</td>
<td>5001</td>
</tr>
<tr>
<td>Elevator3</td>
<td>4</td>
<td>8004</td>
</tr>
<tr>
<td>Tsp</td>
<td>4</td>
<td>4563</td>
</tr>
<tr>
<td>At1</td>
<td>3</td>
<td>88</td>
</tr>
<tr>
<td>At1a</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>At2a</td>
<td>3</td>
<td>462</td>
</tr>
<tr>
<td>Bank-av</td>
<td>3</td>
<td>748</td>
</tr>
<tr>
<td>Bank-sav</td>
<td>3</td>
<td>852</td>
</tr>
<tr>
<td>Bank-fix</td>
<td>3</td>
<td>856</td>
</tr>
</tbody>
</table>
Our Recent Work (*brief overview*)

- Detecting Concurrency Failures
- **Mitigating** Concurrency Failures
- Optimizing Synchronization Code
Type-State Violations

Concurrency related **API usage rule** violations

**Type-State Automaton**

- Showing correct usage rules of a shared object library

**Multi-threaded Client Program**

- $a_1: o$.init();
- $a_2: o$.start();
- $a_3: o$.stop();
- $a_4:...$

- $--- [T1] ---$

- $b_1: o$.destroy();
- $b_2:...$

- $--- [T2] ---$
Runtime Failure Mitigation

- Resilient system on potentially unreliable components
  - Hardening client program with API usage rules
    - Only correct method call sequences are produced
    - Correctness is defined by "type-state automaton"
Runtime Failure Mitigation

- To perturb thread interleavings by a lookahead analysis
- Delay a method call if it leads to the “doomed” state
Results

Runtime Prevention of Concurrency Related Type-State Violations
[Zhang & Wang ISSTA 2014]

<table>
<thead>
<tr>
<th>Appl. Name</th>
<th>LoC</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRIFT-488, revision:772478, bug id:488</td>
<td>23k</td>
<td>pthread_mutex_t</td>
</tr>
<tr>
<td>pbzip2-0.9.4</td>
<td>2k</td>
<td>pthread_mutex_t</td>
</tr>
<tr>
<td>Transmission-1.42</td>
<td>90k</td>
<td>t-&gt;peerMgr</td>
</tr>
<tr>
<td>Memcached-1.4.4, bug id:127</td>
<td>62k</td>
<td>key</td>
</tr>
<tr>
<td>Ilvm 2.8, r118458, bug id:8441</td>
<td>178</td>
<td>AttributesList</td>
</tr>
<tr>
<td>MySQL-4.0.26, bug id:12848</td>
<td>137</td>
<td>Query_cache</td>
</tr>
<tr>
<td>Boost-1.54.0:file reader</td>
<td>48k</td>
<td>basic_file</td>
</tr>
<tr>
<td>Boost-1.54.0:file writer</td>
<td>48k</td>
<td>basic_file</td>
</tr>
<tr>
<td>Boost-1.54.0:timer</td>
<td>27k</td>
<td>cpu_timer</td>
</tr>
<tr>
<td>Boost-1.54.0:object_pool</td>
<td>54k</td>
<td>pool</td>
</tr>
<tr>
<td>Boost-1.54.0:move</td>
<td>45k</td>
<td>file_descriptor</td>
</tr>
<tr>
<td>Boost-1.54.0:unordered_map</td>
<td>53k</td>
<td>unordered_map</td>
</tr>
<tr>
<td>Boost-1.54.0:unordered_set</td>
<td>53k</td>
<td>unordered_set</td>
</tr>
<tr>
<td>Boost-1.54.0:any</td>
<td>27k</td>
<td>any</td>
</tr>
<tr>
<td>Boost-1.54.0:priority_queue</td>
<td>54k</td>
<td>fibonacci_heap</td>
</tr>
<tr>
<td>Boost-1.54.0:mutex</td>
<td>47k</td>
<td>mutex</td>
</tr>
</tbody>
</table>

Test Program | Original | Prevention
---|----------|------------------|
| name | threads | error | time | error | time |
| THRIFT-488 | 3 | crash | (33.0 s) | fixed | 48.3 s |
| Pbzip2 | 5 | crash | (16.7 s) | fixed | 58.0 s |
| Transmission | 2 | crash | (5.5 s) | fixed | 12.6 s |
| Memcached-127 | 4 | crash | (10.1 s) | fixed | 20.1 s |
| Ilvm-8441 | 3 | crash | (0.4 s) | fixed | 0.5 s |
| MySQL-12848 | 2 | crash | (4.0 s) | fixed | 5.3 s |
| Boost:file_read | 2 | fail | (15.1 s) | fixed | 15.5 s |
| Boost:file_write | 2 | fail | (11.4 s) | fixed | 12.7 s |
| Boost:timer | 2 | fail | (9.0 s) | fixed | 9.3 s |
| Boost:object_pool | 2 | fail | (15.2 s) | fixed | 15.6 s |
| Boost:move | 2 | fail | (10.0 s) | fixed | 10.4 s |
| Boost:unordered_map | 2 | fail | (10.2 s) | fixed | 10.8 s |
| Boost:unordered_set | 2 | fail | (11.4 s) | fixed | 12.6 s |
| Boost:any | 2 | crash | (<0.1 s) | fixed | 8.0 s |
| Boost:priority_queue | 2 | crash | (<0.1 s) | fixed | 9.4 s |
| Boost:mutex | 2 | crash | (<0.1 s) | fixed | 7.7 s |

Average runtime overhead: 2.36%

Open source C/C++ applications in the field
Results

Runtime Prevention of Concurrency Related Type-State Violations
[Zhang & Wang ISSTA 2014]

<table>
<thead>
<tr>
<th>Appl. Name</th>
<th>LoC</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>THRIFT, revision:772478, bug id:483</td>
<td>23k</td>
<td>pthread_mutex_t</td>
</tr>
<tr>
<td>pbzip2-0.9.4</td>
<td>2k</td>
<td>pthread_mutex_t</td>
</tr>
<tr>
<td>Transmission-1.42</td>
<td>90k</td>
<td>t-&gt;peerMgr</td>
</tr>
<tr>
<td>Memcached-1.4.4, bug id:127</td>
<td>62k</td>
<td>key</td>
</tr>
<tr>
<td>LLVM 2.8, r118458, bug id:8441</td>
<td>178</td>
<td>AttributesList</td>
</tr>
<tr>
<td>MySQL-4.0.26, bug id:12848</td>
<td>137</td>
<td>Query_cache</td>
</tr>
<tr>
<td>Boost-1.54.0:file reader</td>
<td>48k</td>
<td>basic_file</td>
</tr>
<tr>
<td>Boost-1.54.0:file writer</td>
<td>48k</td>
<td>basic_file</td>
</tr>
<tr>
<td>Boost-1.54.0:timer</td>
<td>27k</td>
<td>cpu_timer</td>
</tr>
<tr>
<td>Boost-1.54.0:object_pool</td>
<td>54k</td>
<td>pool</td>
</tr>
<tr>
<td>Boost-1.54.0:move</td>
<td>45k</td>
<td>file_descriptor</td>
</tr>
<tr>
<td>Boost-1.54.0:unordred_map</td>
<td>53k</td>
<td>unordred_map</td>
</tr>
<tr>
<td>Boost-1.54.0:unordred_set</td>
<td>53k</td>
<td>unordred_set</td>
</tr>
<tr>
<td>Boost-1.54.0:unordred_set</td>
<td>27k</td>
<td>any</td>
</tr>
<tr>
<td>Boost-1.54.0:priority_queue</td>
<td>54k</td>
<td>fibonacci_heap</td>
</tr>
<tr>
<td>Boost-1.54.0:mutex</td>
<td>47k</td>
<td>mutex</td>
</tr>
</tbody>
</table>

Average runtime overhead: 2.36%

Open source C/C++ applications in the field
Our Recent Work

Detecting
Concurrency Failures

Mitigating
Concurrency Failures

Optimizing
Embedded Software
Today’s Talk

- Performance
- **Reliability**
- Security

**Embedded Software**
- C/C++ program
- Numerical analysis
- Abstract interpretation

**Concurrent Software**
- Multi-core, multi-threaded
- Concurrency
- **Symbolic predictive analysis**

**Cryptographic Software**
- Security and Privacy
- Power side channel
- SMT based analysis

**VLSI Circuits**
- Finite state machine
- Boolean analysis
- Model checking

Univ. Colorado, Boulder 2000-2004

NEC Labs, Princeton 2004-2011

Virginia Tech, Blacksburg 2011-present
Let’s take a break...