Active Testing for Concurrent Programs

Pallavi Joshi★
Mayur Naik‡
Chang-Seo Park★
Koushik Sen★
1/8/2009
ParLab Retreat

★ ParLab, UC Berkeley  ‡ Intel Research
Overview

- Checking correctness of concurrent programs
- Active testing directed by
  - Data races
  - Atomicity violations
  - Deadlocks
- Programmer directed active testing
  - “Breakpoints” for parallel programs
- Conclusion
Checking Correctness of Concurrent Programs

- Multiple threads of execution with shared memory
- Want correctness for
  - All executions
  - All inputs need to be checked
- Interference among threads introduces non deterministic bugs
  - All schedules need to be checked
Checking All Schedules

- Model checking
  - Check all schedules systematically
  - Does not scale to large programs

- Focus on potentially buggy schedules
  - Schedules that may expose bugs
  - “Boundary case” schedules

- What are those “potentially buggy schedules”?
Checking All Schedules

- Model checking
  - Check all schedules systematically
  - Does not scale to large programs
- Focus on potentially buggy schedules
  - Schedules that may expose bugs
  - “Boundary case” schedules
- What are those “potentially buggy schedules”? 
  - Schedules exhibiting data races, atomicity violations, deadlocks
Active Testing

- Check the “potentially buggy schedules”
  - Practical and efficient
  - Find many bugs quickly
  - Create an actual execution showing a bug

- How?
Active Testing

- Check the “potentially buggy schedules”
  - Practical and efficient
  - Find many bugs quickly
  - Create an actual execution showing a bug

- Actively control scheduler
  - Explore schedules systematically or randomly
Active Testing

- Check the “potentially buggy schedules”
  - Practical and efficient
  - Find many bugs quickly
  - Create an actual execution showing a bug

- Actively control scheduler
  - Explore schedules systematically or randomly
  - But, preempt threads at “appropriate” states
    - so that buggy schedules get created
  - Observe execution for any abnormal behavior (e.g. uncaught exceptions, assertion failures)
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;

Thread 2

f();

if(X == 0) ERROR;

[Sen; PLDI’08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;

Thread 2

f();

In race

if(X == 0) ERROR;

[Sen; PLDI’08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;

State to preempt

Thread 2

f();

if(X == 0) ERROR;

State to preempt

[Sen; PLDI’08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;

Thread 2

f();

if(X == 0) ERROR;

[Sen; PLDI'08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;
Paused

Thread 2

f();
if(X == 0) ERROR;

[Sen; PLDI’08]
Preempting at "Appropriate" States: Race Directed

Thread 1: X = 1; Paused

Thread 2: f(); if(X == 0) ERROR;

[Sen; PLDI'08]
Preempting at “Appropriate” States: Race Directed

Thread 1

\[ X = 1; \]

Paused

Thread 2

\[ f(); \]

\[ \text{if}(X == 0) \text{ERROR}; \]

Paused

[Sen; PLDI’08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;
Paused

Thread 2

f();

if(X == 0) ERROR;
Paused

[Sen; PLDI’08]
Preempting at “Appropriate” States: Race Directed

Thread 1

X = 1;

Paused

Thread 2

f();

if(X == 0) ERROR;

[Sen; PLDI’08]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1

- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Paused

Thread 2

- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

[Paused]

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
- lock(L1)
- lock(L2)
- unlock(L2)
- unlock(L1)

Thread 2
- f();
- lock(L2)
- lock(L1)
- unlock(L1)
- unlock(L2)

Paused

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1

lock(L1)
lock(L2)
unlock(L2)
unlock(L1)

Thread 2

f();
lock(L2)
lock(L1)
unlock(L1)
unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1

lock(L1)

Paused

lock(L2)
unlock(L2)
unlock(L1)

Thread 2

f();

lock(L2)

lock(L1)

Paused

unlock(L1)
unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1

lock(L1)

lock(L2)

unlock(L2)

unlock(L1)

Thread 2

f();

lock(L2)

lock(L1)

unlock(L1)

unlock(L2)

[Paused]

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1

lock(L1)

lock(L2)

unlock(L2)

unlock(L1)

Thread 2

f();

lock(L2)

lock(L1)

unlock(L1)

unlock(L2)

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Deadlock Directed

Thread 1
lock(L1)
lock(L2)
unlock(L2)
unlock(L1)

Thread 2
f();
lock(L2)
lock(L1)
unlock(L1)
unlock(L2)

Deadlock!

[Joshi, Park, Sen, Naik; Under submission]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1
- t = x;
- x = t + 1;

Thread 2
- f();
- t = 0;

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

- $t = x$;
- $x = t + 1$;

Thread 2

- $f()$;
- $t = 0$;

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

\[ t = x; \]

Thread 2

\[ f(); \]

\[ x = t + 1; \]

Atomicity Violation!

\[ t = 0; \]

atomic

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1
- \( t = x; \)
- \( x = t + 1; \)

Thread 2
- \( f(); \)
- \( t = 0; \)

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

Thread 2

\[
t = x;
\]

\[
x = t + 1;
\]

\[
f();
\]

\[
t = 0;
\]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

\[ t = x; \]

\[ x = t + 1; \]

Thread 2

\[ f(); \]

\[ t = 0; \]

atomic

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

\[ t = x; \]

\[ x = t + 1; \]

atomic

Thread 2

\[ f(); \]

\[ t = 0; \]

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

\[ t = x; \]

\[ x = t + 1; \]

Paused

atomic

Thread 2

\[ f(); \]

\[ t = 0; \]

[Park, Sen; FSE '08]
Preempting at “Appropriate” States:
Atomicity Violation Directed

Thread 1

\[ t = x; \]

\[ x = t + 1; \]

Paused

Thread 2

\[ f(); \]

\[ t = 0; \]

Paused

atomic

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1
\[
t = x; \\
x = t + 1;
\]
Paused

Thread 2
\[
\text{atomic} \\
f(); \\
t = 0;
\]
Paused

[Park, Sen; FSE '08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

\[ t = x; \]

\[ x = t + 1; \]

Paused

atomic

Thread 2

\[ f(); \]

\[ t = 0; \]

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1

1. $t = x$;
2. $x = t + 1$;

Thread 2

1. $f()$;
2. $t = 0$;

atomic

[Park, Sen; FSE ’08]
Preempting at “Appropriate” States: Atomicity Violation Directed

Thread 1:

\[ t = x; \]

\[ x = t + 1; \]

Thread 2:

\[ f(); \]

\[ t = 0; \]

[Park, Sen; FSE ’08]
Bugs Found

- Tool implemented for Java
- Races, deadlocks, atomicity violations in
  - Java Collections Framework
- Deadlocks found and reproduced in
  - Jigsaw web server
  - Java Swing GUI framework
  - Java Database Connectivity (JDBC)
- Atomicity violations in
  - Apache Commons Collections
Lessons Learned

- Controlling scheduler makes testing effective
  - Focus on “potentially buggy schedules”
  - Reproduce bugs with high probability

- Need the right preemption points
  - Directed preemption uncovered many bugs

- Why not let the programmer choose the preemption points?
  - “Breakpoints” for concurrent programs
  - Language for describing schedules
Schedule Exploration Language

- Facility to specify thread schedules
  - User specifies “breakpoints”
    - Preempt a thread at a user specified state
  - User specifies ordering constraints on events
    - Control schedule to meet constraints

- Testing for logical (semantic) bugs

- Schedules are given as input to program
  - Either generated automatically (directed)
  - or programmer specified
  - Inputs to unit tests but on schedules
Example Usage

foo(L) {
1: lock(L)
2: f()
3: unlock(L)
4: ...
5: lock(L)
6: g()
7: unlock(L)
}
foo(L) {
1: lock(L)
2: f()
3: unlock(L)
4: ...
5: lock(L)
6: g()
7: unlock(L)
}

Example Usage
```c
foo(L) {  
1: lock(L)  
2: f()  
3: unlock(L)  
4: ...  
5: lock(L)  
6: g()  
7: unlock(L)  
}
```

Break

```
lock(L2)
```
foo(L) {
1: lock(L)
2: f()
3: unlock(L)
4: ...
5: lock(L)
6: g()
7: unlock(L)
}

break lock(L2), 5
Example Usage

foo(L) {
  1: lock(L)
  2: f()
  3: unlock(L)
  4: ...
  5: lock(L)
  6: g()
  7: unlock(L)
}

break lock(L2), 5, T1
Abstraction: Naming Runtime Objects

- Threads and locks are runtime objects
  - Address changes across runs
  - Need to identify consistently with only static information
- An abstraction based on object creation site
  - Line number (PC) of where object allocated
main {  
1: Lock L1 = new Lock();  
2: Lock L2 = new Lock();  
3: Thread T1 =  
    new MyThread(L1);  
4: Thread T2 =  
    new MyThread(L2);  
5:  ...  
}  
MyThread(L) {  
6: foo(L);  
}
main {  
1:  Lock L1 = new Lock();  
2:  Lock L2 = new Lock();  
3:  Thread T1 =  
      new MyThread(L1);  
4:  Thread T2 =  
      new MyThread(L2);  
5:  ...  
}  
MyThread(L) { 
6:  foo(L);  
}
Abstraction: Naming Runtime Objects

- Threads and locks are runtime objects
  - Address changes across runs
  - Need to name consistently and statically
- Abstraction based on object creation site
  - Line number (PC) of where object allocated
- Use and combine richer abstractions for increased separation
  - Objects created in loops
  - Objects created in different contexts
Another Example Usage

before (notify(abs(o)), abs(T3) ->
wait(abs(o)), abs(T1))
Conclusion

- Practical and efficient testing of concurrent programs
- Find and reproduce bugs to lessen burden of manual inspection
- “Input language” for user specified schedules

parkcs@cs.berkeley.edu