CCC Seminar, ETH Zürich
March 11, 2015
Fabian Stutz

CONLOCK

A Constraint-Based approach to dynamic checking on Deadlocks in multithreaded programs

Yan Cai, Shangru Wu, W.K. Chan
ICSE 14

Problem outline

Find (real) deadlocks

Static techniques
- Analyze code
- Many false positives

Dynamic techniques
- „Educated“ scheduling
- Still many false positives

Goals

(a) Find potential deadlocks

(b) Automatically confirm potential deadlocks

- Eliminate false positives
- Do ***not*** eliminate true positives

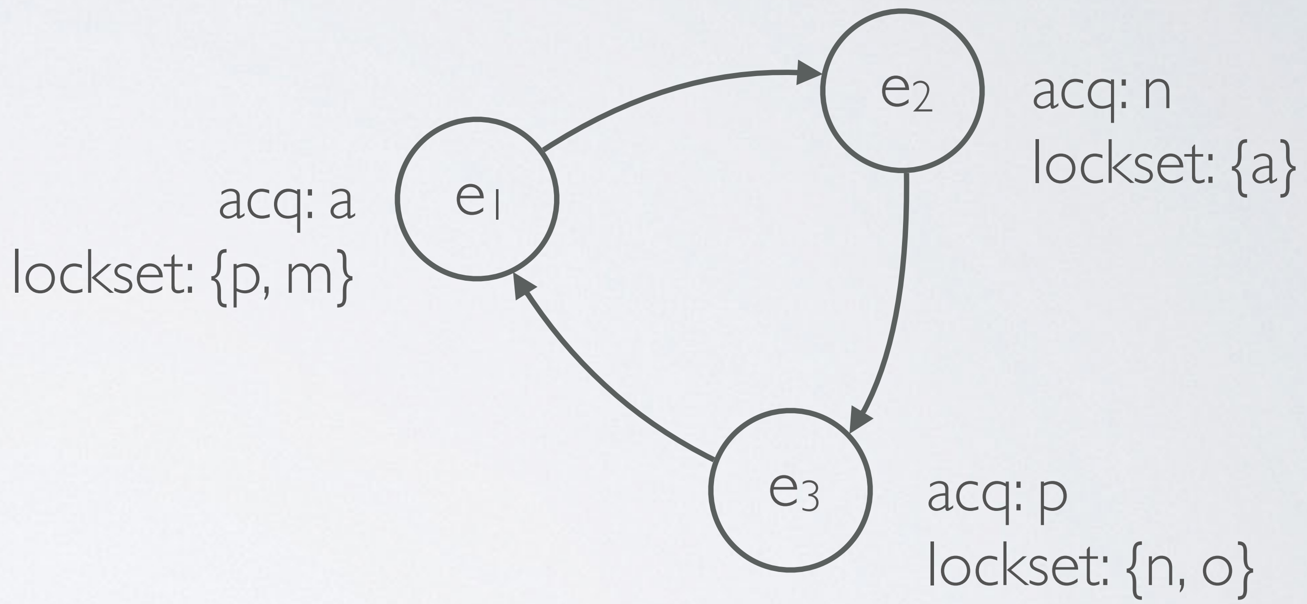
Concepts

Events

- Lock acquisition or release

Lockset

- Set of locks hold by one thread



Cycles

- Chain of events ε, that build a circular dependency
- potential deadlock

Example

thread 1 Events	thread 1 Locksets	thread 2 Events	thread 2 Locksets
01 acq(n)	{}		
02 rel(n)	{n}		
03 acq(a)	{}		
04 acq(n)	{a}		
05 rel(n)	{a,n}	10 acq(a)	{}
06 acq(p)	{a}	11 rel(a)	{a}
07 acq(m)	{a,p}	12 acq(n)	{}
08 acq(n)	{a,m,p}	13 acq(p)	{n}

ε_1 Cycle c ε_2

Approach

`Phase 0:` Identify cycles

`Phase 1:` Generate constraints

- Analyze order of operations
- Provoke deadlock

`Phase 2:` Educated scheduling of execution

- No violation of any constraint
- Trigger deadlock (if any)

Phase 1: Constraints

Should happen before relation: $e_1 \rightsquigarrow e_2$

Rule 1:

Deadlock event ε_α on t_α is an acq. of lock o.
⇒ All operations of any thread $t_{\beta\neq\alpha}$ on o must happen before ε_α.

Rule 2:

Thread t_α holds lock on object o from e_1 until its deadlock event ε_α.
⇒ All operations (except ε_β) of any thread $t_{\beta\neq\alpha}$ on o must happen before e_1.

Phase 1: Example

thread 1

Events / Locksets

```
01 acq(n) { , , , } 01
02 rel(n) { , ,n, } 02
03 acq(a) { , , , } 03
04 acq(n) {a, , , } 04
05 rel(n) {a, ,n, } 05
06 acq(p) {a, , , } 06
07 acq(m) {a, , ,p} 07
08 acq(n) {a,m, ,p} 08
```

thread 2

Events / Locksets

```
10 acq(a) { , , , }
11 rel(a) {a, , , }
12 acq(n) { , , , }
13 acq(p) { , ,n, }
```

Rule 1:

Deadlock event ε_α on t_α is an acq. of lock o.
⇒ All operations of any thread $t_{\beta\neq\alpha}$ on o must happen before ε_α.

Phase 1: Example

thread 1			thread 2	
Events	**Locksets**		**Events**	**Locksets**

```
thread 1                          thread 2
Events        Locksets            Events        Locksets
01 acq(n)     { , , , }
02 rel(n)     { , ,n, }
03 acq(a)     { , , , }
04 acq(n)     {a, , , }
05 rel(n)     {a, ,n, }           10 acq(a)     { , , , }
06 acq(p)     {a, , , }           11 rel(a)     {a, , , }
07 acq(m)     {a, , ,p}           12 acq(n)     { , , , }
08 acq(n)     {a,m, ,p}           13 acq(p)     { , ,n, }
```

Rule 2:

Thread t_α holds lock on object o from e_1 until its deadlock event ε_α.
⇒ All operations (except ε_β) of any thread $t_{\beta\neq\alpha}$ on o must happen before e_1.

Phase 1: Optimization

$e_{01} \rightsquigarrow e_{15}$

$e_{02} \rightsquigarrow e_{15}$

$e_{04} \rightsquigarrow e_{15}$

$e_{05} \rightsquigarrow e_{15}$

$e_{06} \rightsquigarrow e_{16}$

$e_{13} \rightsquigarrow e_{03}$

$e_{14} \rightsquigarrow e_{03}$

$e_{15} \rightsquigarrow e_{08}$

Reduce Constraint set

- Transitivity
- Program Locking Order

Nearest Scheduling points

- Nearest operation where lockset is empty
- Only consider operations from NSPs

Phase 2: Scheduling

Schedule randomly (by OS)

- Keep track of constraints
- Only „non-violating“ operations get performed

No progress possible?

- Deadlock? Output trace and halt. Success!
- No deadlock? Report scheduling violation, deadlock not possible anymore. Start over.

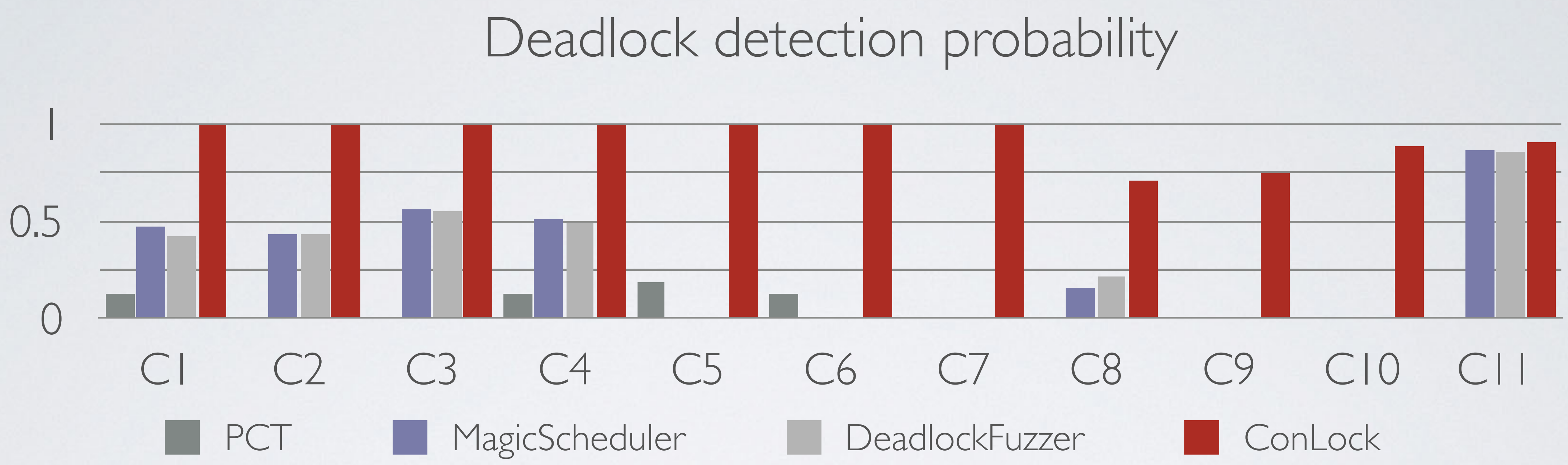
Analysis

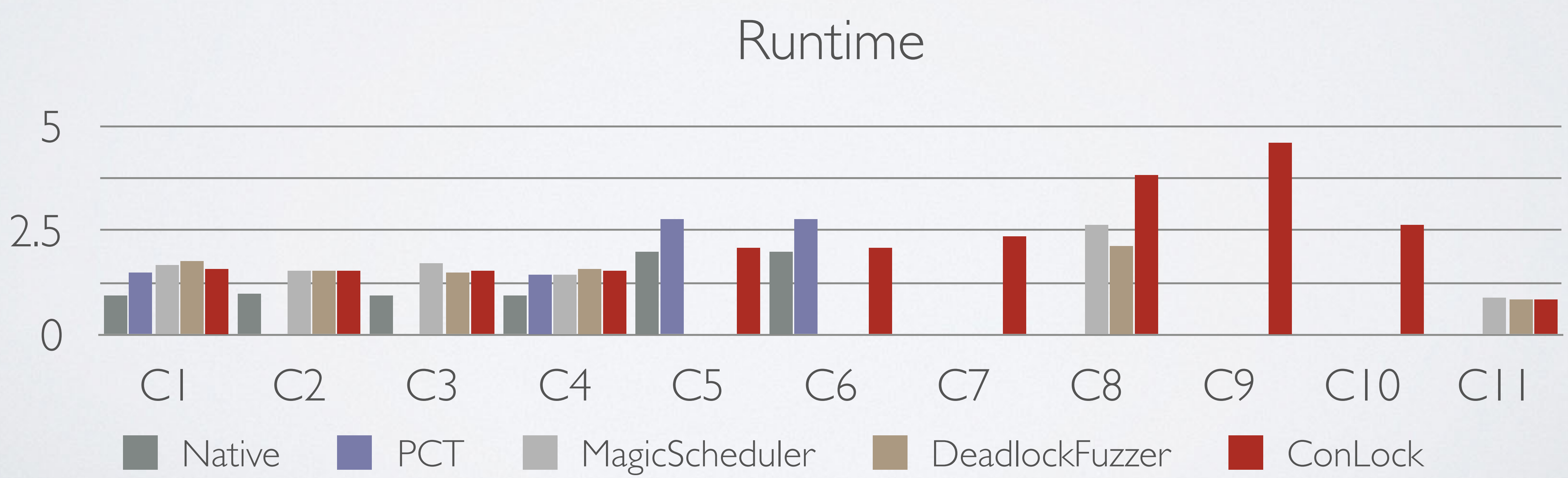
Experiment

- Code from JDBC Connector, SQLite and MySQL Server
- Compare with other deadlock detectors (PCT, MagicScheduler, DeadlockFuzzer)
- 100 runs each

- Test precision and efficiency for known deadlock
- Test efficiency for false positives

Analysis


Deadlock detection probability
1
0.5
0
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
PCT
MagicScheduler
DeadlockFuzzer
ConLock


Runtime
5
2.5
0
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
Native
PCT
MagicScheduler
DeadlockFuzzer
ConLock

Analysis

False positives

- Analysed 87 false positives
- All other deadlock detection algorithms timed out
- All but one run of Conlock showed scheduling violations

→ Probabilistic method to discard cycles

Conclusion

Limitations

- Sufficient test set?
- False positives? → Manual inspection!
- Can it find unknown bugs?

Contributions

- Successful new approach
- Significantly improved precision

Thank you.