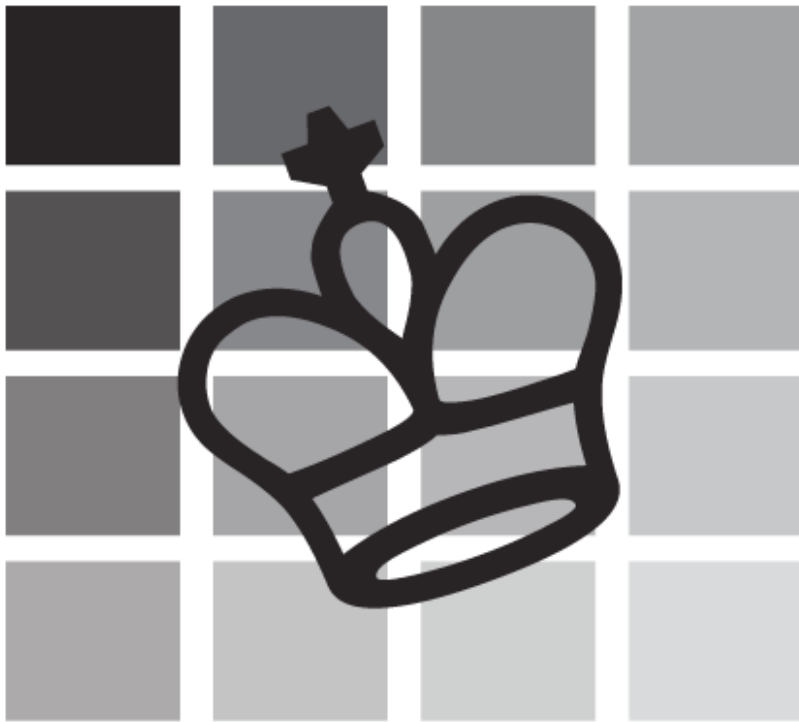


Checkmate: a Generic Static Analyzer of Java Multithreaded Programs



Pietro Ferrara

Chair of Programming Methodology
ETH Zurich
Switzerland

7th IEEE International Conference on Software Engineering and Formal Methods
Hanoi, Vietnam

Multicore revolution

- Multicore:
 - > one CPU contains many cores
 - > the only way to extend Moore's law
- Current trend: **manycore**
 - > Quad and eight cores already on the market
- Sequential programs do not exploit multicores
- Parallelism supported through **multithreading**
 - > Java
 - > C#
- Implicit communications via shared memory
- Synchronization on monitors
- **Subtle** and problematic



Testing and static analysis

- Testing can expose only **few multithreaded executions**
 - > Some executions exposed only by specific VM
 - > Difficult to reproduce an execution
- **Not sufficient** to effectively debug multithreading
- **Static analysis**:
 - > Infer and prove properties at **compile time** respected by **all the possible executions**
- **Tradeoff** between precision and efficiency
- **Successfully applied** to sequential programs
- **Abstract interpretation**
 - > Mathematical theory developed by P. & R. Cousot
 - > **Define** semantics of programs
 - > **Soundly approximate** it



Generic analyzers

- **Main components:**
 - > Numerical domain
 - > Heap abstraction
 - > Property of interest
- **Generic analyzers**
 - > New trend
 - > Already applied to industrial contexts (e.g. Microsoft)
- **Plugged with different**
 - > Numerical domains
 - > Properties
- **Reuse of**
 - > Heap abstraction
 - > Semantics



An example

```
class MyThread extends Thread{
    BankAccount acc;
    public void run() {
        synchronized(acc.am) {
            if (acc.am.m<100)
                acc.am=null;
            else acc.am.m-=100);
        }
    }
}
```

```
static void main (String[] ar) {
    BankAccount acc=new BankAccount();
    acc.am.m=1000;
    new MyThread(acc).start();
    synchronized(acc.am) {
        System.out.println(acc.am.m);
    }
}
```

```
class BankAccount {
    Amount am=new Amount();
}
class Amount {int m=0;}
```

- Several properties
 - > Data race
 - > Null pointer access
- Numerical precision
 - > Intervals
- Memory model
 - > What may be executed in parallel?



NullPointerException – Sign

```
class MyThread extends Thread{  
    {  
        acc.am) {  
            if (acc.am.m<100)  
                acc.am=null;  
            else acc.am.m-=100  
        }  
    }  
}
```

```
class BankAccount {  
    Amount am=new Amount();  
}  
class Amount {int m=0;}  
    ,null}
```

+ < 100? T

The program may throw
a NullPointerException

```
BankAccount acc=new BankAccount();  
acc.am=new Amount();  
new MyThread().start();  
synchronized (acc){  
    System.out.println(acc.am.m);  
}
```

> 0
> -

domain



NullPointerException – Intervals

```
class MyThread extends Thread{  
    {  
        [1000..1000] < 100?  
        False  
        {  
            acc.am) {  
                if (acc.am.m < 100)  
                    acc.am = null;  
                else (acc.am.m -= 100);  
            }  
        }  
    }  
}
```

```
class BankAccount {  
    Amount am = new Amount();  
}  
class Amount {int m = 0;}
```

`acc.am` \Rightarrow `#a1`
`#a1.m` \rightarrow `[0000..1000]`

The program never throws
a NullPointerException

```
static {  
    BankAccount acc = new BankAccount();  
    acc.am.m = 1000;  
    new MyThread(acc).start();  
    synchronized (acc.am) {  
        System.out.println(acc.am.m);  
    }  
}
```

Intervals domain
 $> [a..b]$



NullPointerException – Memory Model

```
class MyThread extends Thread{
    BankAccount acc;
    public void run() {
        synchronized(acc.am) {
            → if (acc.am.m < 100)
                acc.am = null;
            else acc.am.m -= 100;
        }
    }
}
```

0 1000

Everything is in parallel:
The program may
throw a
NullPointerException

```
static void main (String[] args) {
    BankAccount acc = new BankAccount(1000);
    acc.am.m = 1000;
    → new MyThread(acc).start();
    synchronized(acc.am) {
        System.out.println(acc.am.m);
    }
}
```



Data Race

```
class MyThread extends Thread{
    BankAccount acc;
    public void run() {
        → synchronized(acc.am) {
            if(acc.am.m<1000)
                acc.am=null;
            else acc.am.m-=100);
        }
    }
}
```

```
class BankAccount {
    Amount am=new Amount();
}
class Amount {int m=0;}
```

this.acc.am → #a1

The program is data race free

static

```
→ BankAccount acc=new BankAccount();
acc.am.m=1000;
new MyThread(acc).start();
→ synchronized(acc.am) {
    System.out.println(acc.am.m);
}
```

LOCK on #a1

- Other property
> Data races



Outline

1. Introduction

- ~~Multithreading~~
- ~~Abstract interpretation and generic analyzers~~

2. Checkmate

- Overall structure
- Implementation

3. Experimental results

4. Open problems and conclusion



Checkmate

- 1st generic analyzer of multithreaded program
- **Generic with respect to**
 - > Numerical domain
 - Interval, sign, parity, congruence
 - > Memory model
 - Happens-before one
 - > Property of interest
 - Multithreading: data race, deadlock, determinism
 - Well-known: division by zero, access to null, etc..

<http://www.pietro.ferrara.name/checkmate>

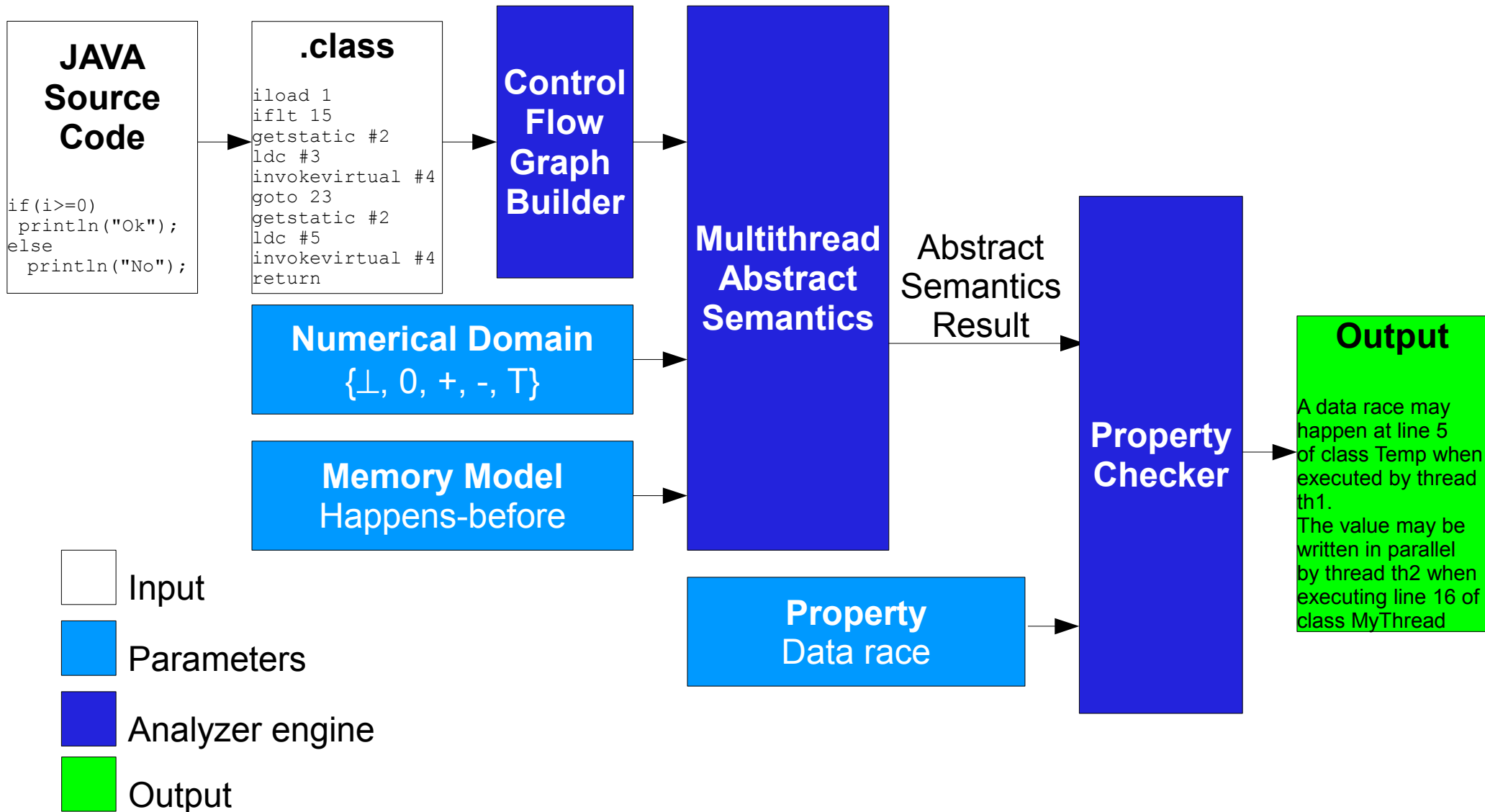


Supported features

- **Whole-program analysis**
 - > Start from a **main** method
 - > Inter-procedural analysis
- **Checkmate supports:**
 - > All the Java bytecode language
 - > Dynamic unbounded creation of **threads**
 - Threads are objects
 - > Dynamic creation and management of **monitors**
 - Monitors are defined on objects
 - **Heap analysis** abstracts them
 - > Method **overloading and overriding**
 - > **Recursive** methods



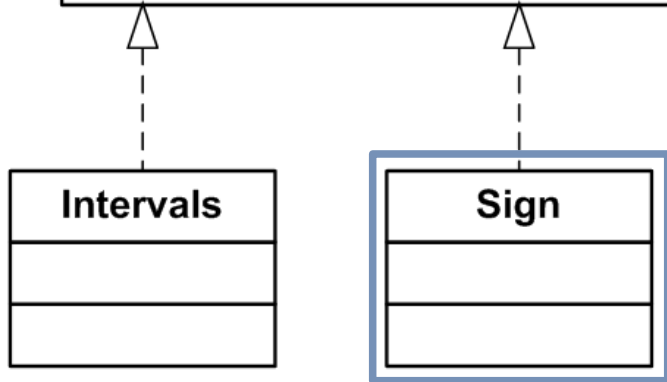
Architecture



Numerical domains

	<	0	+	-	T
0	false	true	false	T	
+	false	T	false	T	
-	true	true	T	T	
T	T	T	T	T	

`+add(in v1 : NumericalValue, in v2 : NumericalValue) : NumericalValue`
`+divide(in v1 : NumericalValue, in v2 : NumericalValue) : NumericalValue`
`+multiply(in v1 : NumericalValue, in v2 : NumericalValue) : NumericalValue`
`+subtract(in v1 : NumericalValue, in v2 : NumericalValue) : NumericalValue`
`+evalConstant(in c : int) : NumericalValue`
`+lub(in v1 : NumericalValue, in v2 : NumericalValue) : NumericalValue`
`+widening(in v1 : NumericalValue) : NumericalValue`
`+lessEqual(in v : NumericalValue, in c : int) : BooleanDomain`
`+testTrue(in v1 : NumericalValue, in v2 : NumericalValue, in c : ComparisonOperator) : BooleanDomain`
`+testFalse(in v1 : NumericalValue, in v2 : NumericalValue, in c : ComparisonOperator) : BooleanDomain`
`+equal(in v : Object) : bool`
`+hashCode(in v : NumericalValue) : int`
`+toString() : String`



```

public void run () {
  synchronized (acc.am) {
    if (acc.am.m < 100)
      acc.am = null;
    else acc.am.m -= 100;
  }
}
  
```



Memory Model

[1000..1000] \sqcup [0..0] \longrightarrow [0..1000]

```
«interface»
MemoryModel
+get(in ref : Reference, in s : String, in current_state : JVMState, in current_statement : Statement) : Value
+factory(in prev : MultiThreadResult, in iteration_number : int) : MemoryModel
```

```
public void run() {
  synchronized(acc.am) {
    if (acc.am.m < 100)
      acc.am = null;
    else acc.am.m -= 100;
  }
}
```

```
static void main(String[] ar) {
  BankAccount acc = new BankAccount();
  acc.am.m = 1000;
  new MyThread(acc).start();
  synchronized(acc.am) {
    System.out.println(acc.am.m);
  }
}
```

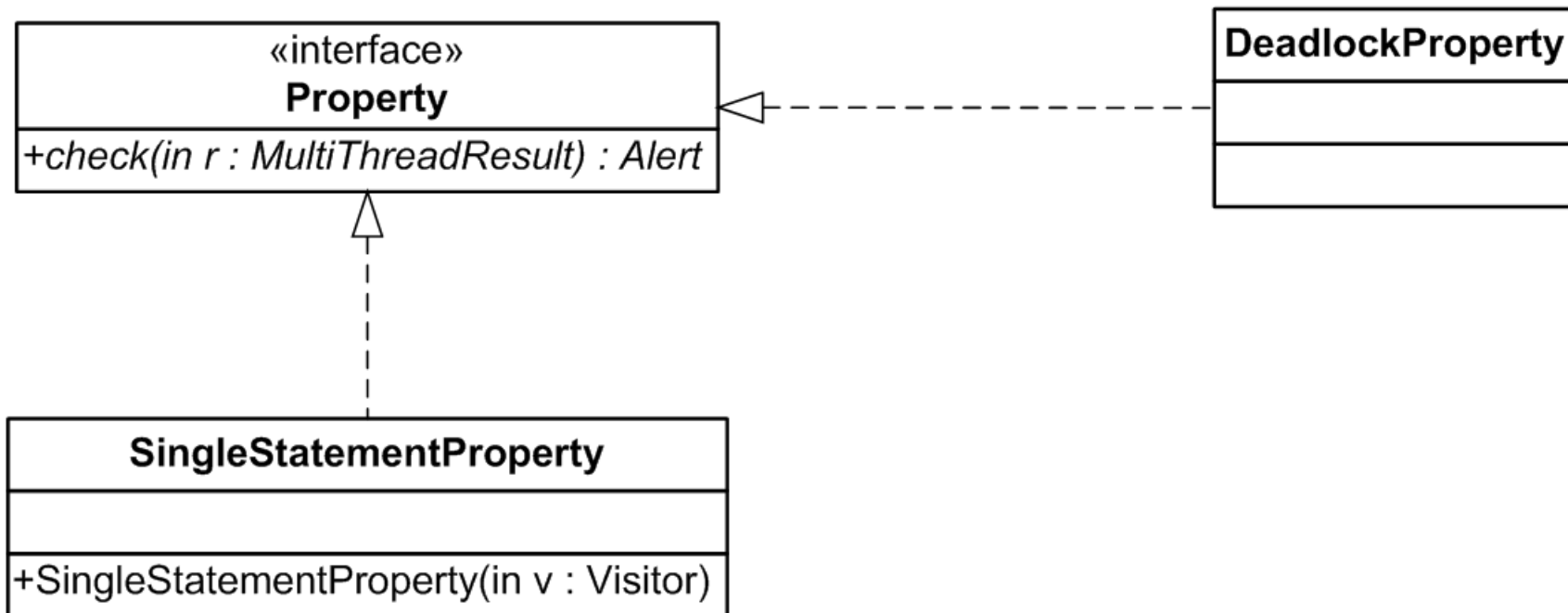


Properties

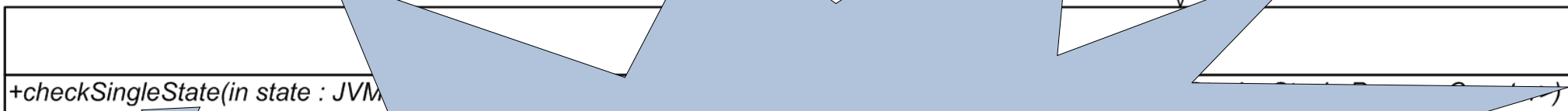
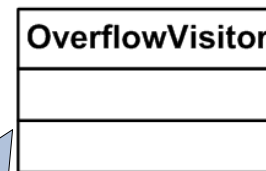
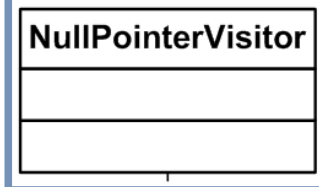
- **MultiThreadResult:**

Thread1: $x=[0..0] \xrightarrow{x++} x=[1..1] \xrightarrow{\text{int } y=10} x=[1..1], y=[10..10]$

Thread2: $z=[1..2] \xrightarrow{z=z*x} z=[0..2] \xrightarrow{\text{int } w=z*2} z=[0..2], w=[0..4]$



Properties



`null ∈ t`

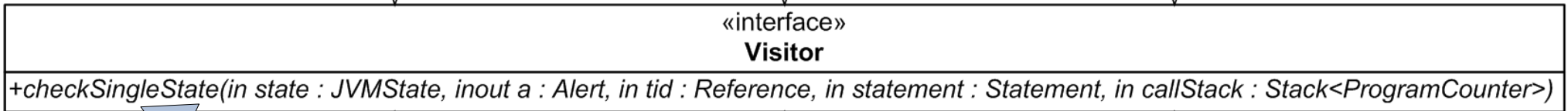
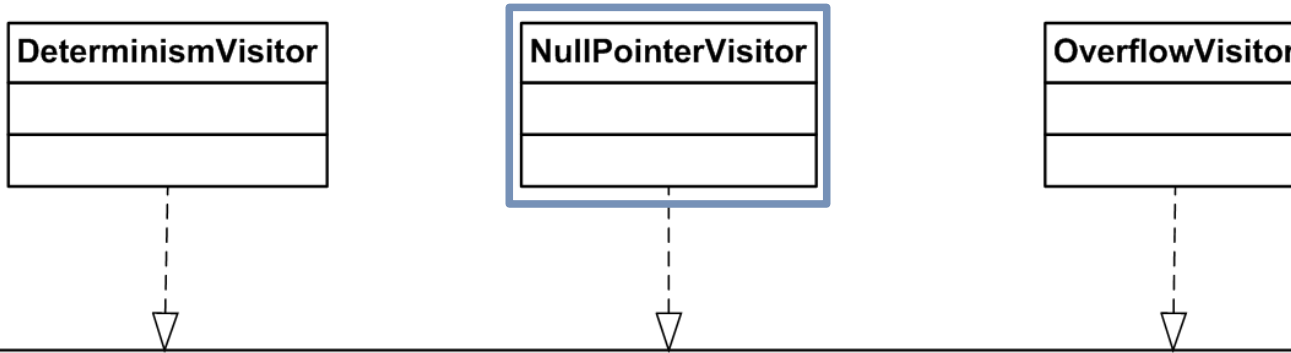
The program may throw a `NullPointerException`

```
static
    Bank
    a
    new MyIntr
    synchroniz
    System.out.println(acc.am.m);
}
```

`acc.am` → `{#a1,null}`
`#a1.m` → `+`



Properties



$null \notin \{ \#a1 \}$

This statement never throws a NullPointerException

```
state
Bar
acc
new
synchronized (acc.am) {
    System.out.println(acc.am.m);
}
```

$\#a1.m \rightarrow [900..1000]$



Command line interface

```
C:\Users\Pietro\Checkmate>java -jar checkmate.jar DataRace1 -p:d -n:i -m:h
Class java.lang.Object not found in the provided directory.
Read from the local repository of JUM.
Static variables not initialized

Class java.lang.String not found in the provided directory.
Read from the local repository of JUM.
Static variables not initialized

Iteration 1
Class java/lang/Thread not found in the provided directory.
Read from the local repository of JUM.
Static variables not initialized

Iteration 2
Iteration 3
*****

Warning: Possible data race at line 12 of method run of class DataRace1$MyThread

*****

Warning: Possible data race at line 32 of method main of class DataRace1

*****
```



Eclipse plugin interface

- New
- Open F3
- Open With
- Open Type Hierarchy F4
- Open Call Hierarchy Ctrl+Alt+H
- Show In Alt+Shift+W
- Copy Ctrl+C
- Copy Qualified Name
- Paste Ctrl+V
- Delete Delete
- Remove from Context Ctrl+Alt+Shift+Down
- Build Path
- Source Alt+Shift+S
- Refactor Alt+Shift+T
- Import...
- Export...
- References
- Declarations
- Refresh F5
- Assign Working Sets...
- Toggle Class Load Breakpoint
- Run As
- Debug As

Property to be analyzed

- Weak determinism
- Full determinism
- Data race
- Null pointer access
- Overflow
- Division by zero
- Deadlock

OK Cancel

Memory model

All values written in parallel
Excluding values written before a thread is launched
Happens-before memory model

OK Cancel

Numerical domain

- Signs
- Intervals
- Parity
- Congruence
- Top (i.e. ignore numerical information)

OK Cancel

Problems Tasks Console Error Log Search Checkmate results

- Warning: Possible NullPointerException at line 38 of method main of class MultipleProperties
- Warning: Possible NullPointerException at line 41 of method main of class MultipleProperties

Checkmate Settings



Outline

1. ~~Introduction~~

- ~~Multithreading~~
- ~~Abstract interpretation and generic analyzers~~

2. ~~Checkmate~~

- ~~Overall structure~~
- ~~Implementation~~

3. Experimental results

4. Open problems and conclusion



Experimental results

- Applied to
 - > Some case studies taken from [LEA]
 - > Some benchmarks taken from [PRA, BENCH]
- Precise
- Fast for small programs
 - > But **not scalable** for large\ industrial programs

[LEA] D. Lea. *Concurrent Programming in Java*. Addison-Wesley, 1996.

[PRA] C. Von Praun and T. R. Gross. *Object race detection*. In ACM Press, editor, *Proceedings of OOPSLA 01*, 2001.

[BENCH] *Java Grande Forum Benchmark Suite*. At <http://www.epcc.ed.ac.uk/research/activities/java-grande/>



Precision

- Patterns of multithreaded programming
- Several different case studies
- Discover **all** the behaviors of interest

- Required **different properties**
 - > Data race, deadlock, determinism
 - > Determinism is the most used

- Different numerical domains
 - > Intervals domain is the most precise
 - > Often we do not need its precision!



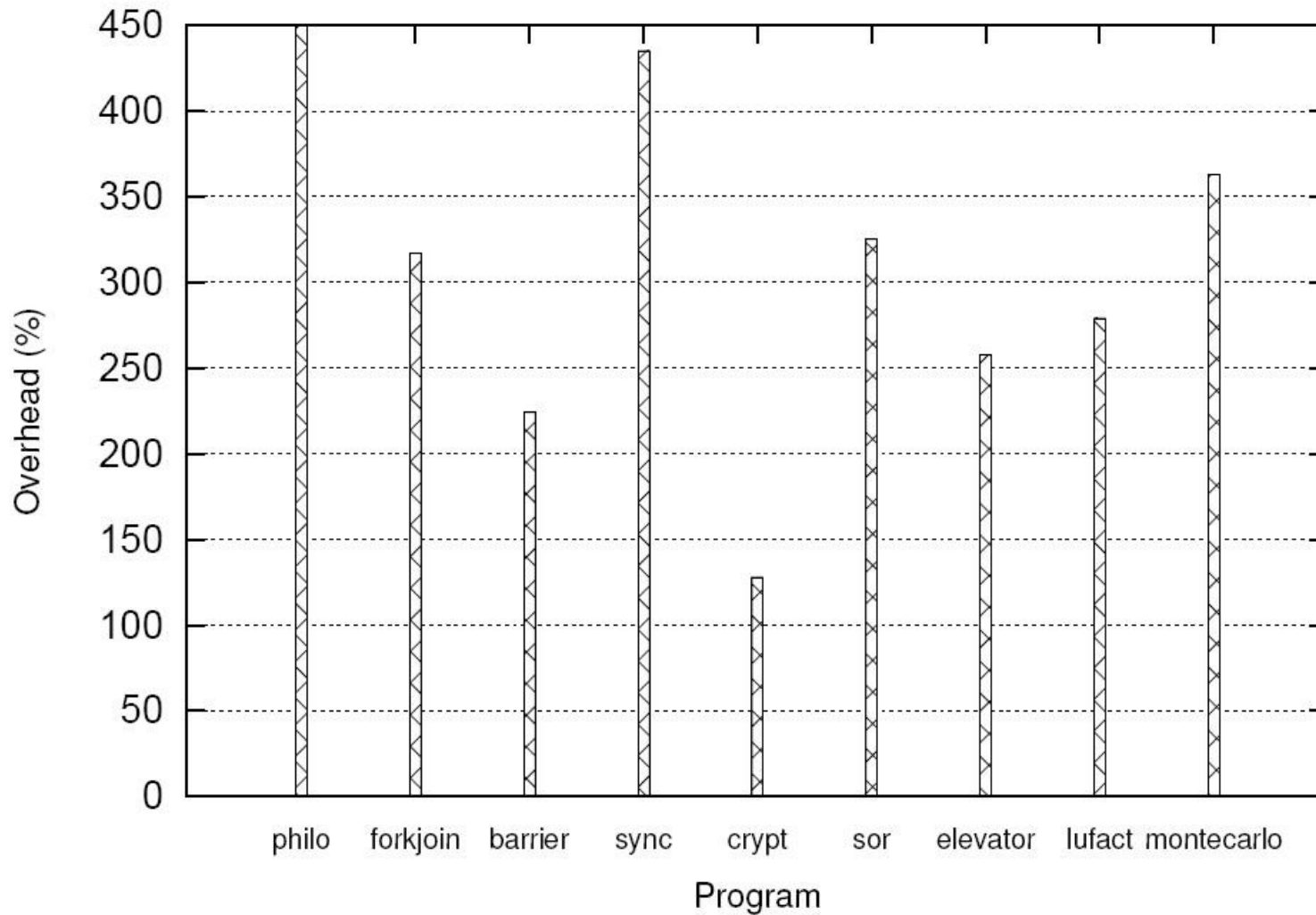
External benchmarks

Program	St.	Th.	Top	Sign	Int.	Par.	Cong.
philo	213	2	<1''	<1''	1''	<1''	<1''
forkjoin	170	2	<1''	<1''	<1''	<1''	<1''
barrier	363	3	<1''	1''	2''	1''	1''
sync	320	3	1''	1''	3''	1''	2''
crypt	2636	3	5''	6''	17''	6''	5''
sor	1121	2	4''	7''	17''	6''	5''
elevator	1829	2	31''	11''	19''	30''	29''
lufact	3732	2	27''	53''	5'59''	29''	29''
montecarlo	3864	2	1'02''	2'35''	1h00'56''	1'43''	1'04''



Overhead of multithread semantics

Overhead in % of multithread fixpoint computation using Intervals and HB memory model



Outline

1. ~~Introduction~~

- ~~Multithreading~~
- ~~Abstract interpretation and generic analyzers~~

2. ~~Checkmate~~

- ~~Overall structure~~
- ~~Implementation~~

3. ~~Experimental results~~

4. Open problems and conclusion



Related work

- **Generic analyzers**
 - > Do **not support** multithreading
- **Specific analyses (e.g. on data race and deadlock)**
 - > More precise (usually)
 - > **Not generic**
- **Context bound model checking**
 - > **Generic**
 - > **More precise**
 - > **Not sound** for all the possible executions



Conclusion

- 1st generic analyzer of multithreaded programs
 - > Pluggable with different
 - Properties
 - Numerical domains
 - Memory models
- Applied to
 - > Some patterns of multithreaded programming
 - > Benchmarks
- Experimental results are encouraging
 - > Precise
 - > Fast



Future work

- **Optimize** the computation of single-thread semantics
- **Improve** the precision of the analysis
 - > Support numerical relational domains
 - e.g. Octagons, Polyhedra
 - > Refine the memory model
 - More synchronizations primitives
- Apply the analysis to **other properties**



Future work

- Whole program analysis
 - > Limit: **do not scale!**
- **Modular reasoning**
 - > **Impossible** on multithreaded programs
 - > **Lack** of programming languages and contracts
- Object-oriented programs
 - > Restrict the **visibility** of fields and methods
 - public, private, protected
 - > **Contracts** on classes and methods
- Intuition
 - > **Apply and tune these ideas to multithreading**



Related Publications

[Ferr08] P. Ferrara, "*Static analysis via abstract interpretation of the happens-before memory model*", in Springer editor, Proceedings of TAP 2008, volume 4966 of LNCS, Prato, Italy, April 9-11, 2008

[Ferr08a] P. Ferrara, "*Static analysis of the determinism of multithreaded programs*", in IEEE Computer Society, editor, Proceedings of SEFM 2008, Cape Town, South Africa, November 10-14, 2008

[Ferr08b] P. Ferrara, "*A fast and precise analysis for data race detection*", in Elsevier editor, Proceedings of Bytecode'08, ENTCS, Budapest, Hungary, April 6, 2008



Question time

Thank you!

<http://www.pietro.ferrara.name/checkmate>

