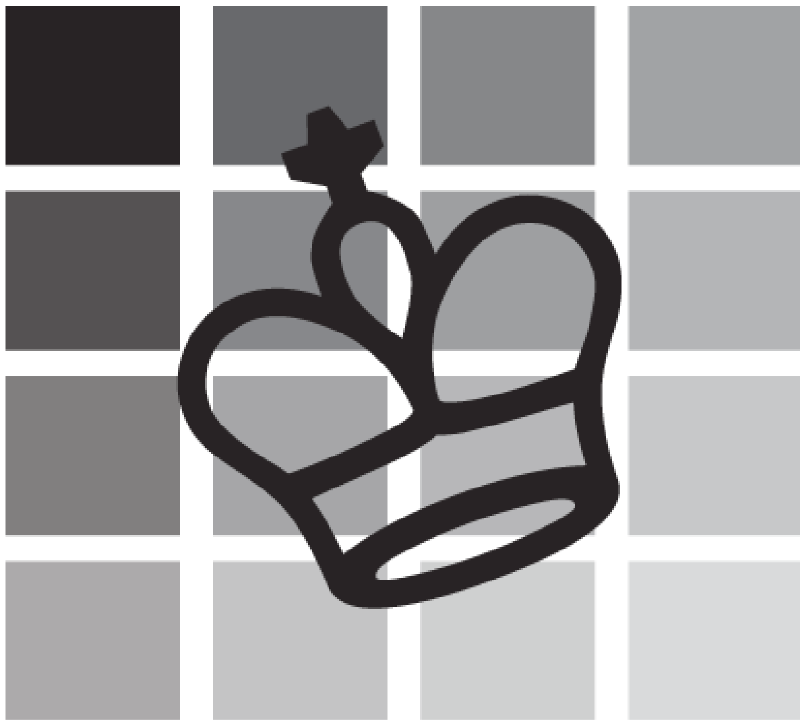
Checkmate: a Generic Static Analyzer of Java Multithreaded Programs



Pietro Ferrara

Chair of Programming Methodology
ETH Zurich
Switzerland

Università Ca' Foscari, Departement of Computer Science, Venice, Italy

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 sufficent 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);
    }
  }
}
```

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

```
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);
  }
}
```

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

NullPointerException – Sign

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

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

+ < 100? T

,null}

domain

> 0

> -

The program may throw a NullPointerException

```
Ba
acc.am
new
synchronize        m)
   System.ou   ntln(acc.am.m);
  }
}
```

NullPointerException – Intervals

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

[1000..1000] < 100?
False

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

acc.am → #a1
#a1.m → [1000..1000]

```
stati
⇨ Ba
⇨ acc.am.m=1000;
⇨ new MyThread(acc).start();
⇨ synchronized(acc.am) {
⇨   System.out.println(acc.am.m);
  }
}
```

The program never throws
a NullPointerException

- 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.m=null;
      else acc.am.m-=1
    }
  }
}
```

0 1000

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

```
class BankAccount {
  ();
```

ows
tion

Everything is in parallel:
The program may
throw a
NullPointerException

Data Race

```
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);
     }
   }
}
```

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

this.acc.am → #a1

1

The program is data race free

1

```
static
   BankAccount acc=new BankAccount();
   acc.am.m=1000;
   new MyThread(acc).st   ();
   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

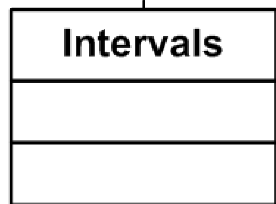
Numerical domains

«interface»

+add(in v1 : NumericalValue, in v…
+divide(in v1 : NumericalValue, i…
+multiply(in v1 : NumericalValue, …
+subtract(in v1 : NumericalValue, …
+evalConstant(i… int) : Numeri…
+lub(in v1 : Numerica… in v2…
+widening(in v1 : NumericalValu…
+lessEqual(in v : NumericalValue…
+testTrue(… v1 : NumericalValue … BooleanDomain
+testFalse(in v1 : Numerica… in v2 : NumericalValue, in c : …arisonOperator) : BooleanDomain
+equal(in v : Object) : bool
+hashCode(in v : NumericalValue) : int
+toString() : String

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

+T < ?



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

Memory Model

[1000..1000] ⊔ [0..0] → [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

[1000..1000]

?

```
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);
  }
}
```

Abstract semantics result

- Abstract semantics result:
  - > Function that relates each thread to a trace of states
  - > One state for each statement of the program
  - > Abstract state:
    - Approximation of the numerical values

- For instance,

Thread1: x=[0..0] —x++→ x=[1..1] —int y=10→ x=[1..1] y=[10..10]

Thread2: z=[1..2] —z=z*x→ z=[0..2] —int w=z*2→ z=[0..2] w=[0..4]

Properties

«interface»
Property

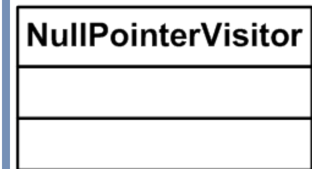
+check(in r : MultiThreadResult) : Alert

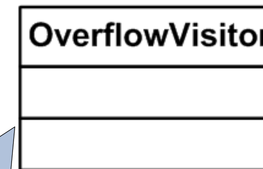
DeadlockProperty

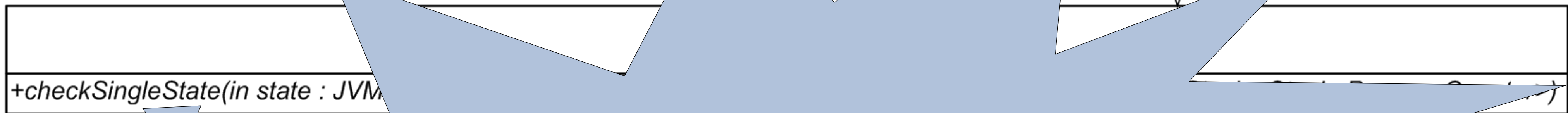
SingleStatementProperty

+SingleStatementProperty(in v : Visitor)

Properties







null ∈ {

The program may throw
a NullPointerException

```
static
  Bank
  a
  new MyThre
  synchroniz   .am) {
    System.o  println(acc.am.m);
  }
}
```

acc.am → {#a1,null}

#a1.m → +

Properties

DeterminismVisitor

NullPointerVisitor

OverflowVisitor

«interface»
Visitor
+checkSingleState(in state : JVMState, inout a : Alert, in tid : Reference, in statement : Statement, in callStack : Stack<ProgramCounter>)

WeakDeterminismVisitor

DivisionByZeroVisitor

null ∉ {#a1}

This statement never throws a NullPointerException

```
static
  Ba
  ac
  new
  synchronized(acc.am) {
    System.out.println(acc.am.m);
  }
}
```

#a1.m → [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 JVM.
Static variables not initizalized

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

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

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
Checkmate
Settings

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

Demo

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
  - Case studies of multithreaded programming
  - Examples of the Java Memory Model
  - Incremental applications
  - External benchmarks taken from [PRA,BENCH]
- Precise
- Fast for small programs
  - But not scalable for large\ industrial programs

[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/

Patterns of multithreading

- Several different case studies
- Discover all the behaviors of interest

- Required different properties
  - Datarace, deadlock, determinism
  - Determinism is the most used

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

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

Java memory model

Thread 1	Thread 2

`r1=x;`
`y=1;`
`r2=x;`
	`x=1;`
	`r3=y;`

- Initially, x=y=0
- r1=r2=r3=0 is a possible behavior

Thread 1	Thread 2

`r3=x;`
`if(r3==0)`
`   x=42;`
`r1=x;`
`y=r1;`
	`r3=y;`
	`x=r3;`

- Initially, x=y=0
- r1=r2=r3=42 is a possible behavior

Java memory model

- We applied Checkmate to several case studies
- Usually composed by few statements and threads
- The analysis converges in some milliseconds
- Analysis of the happens-before memory model
  - More approximated than the Java memory model
  - It allows more behaviors
- Nevertheless, we catch the behavior of interest
- Too much rough when dealing with `volatile` variables
  - Checkmate does not track information on them
  - We may refine the analysis
    - The framework seems to be enough flexible

Incremental application

	Th.	St.	Top	Sign	Int.	Par.	Cong.
1	3	452	814	361	217	404	294
2	5	684	409	391	356	620	545
3	7	807	712	595	925	521	642
4	9	1049	799	823	3806	703	642
5	11	1173	1090	919	5887	779	616
6	13	1405	1382	824	7161	900	986
7	15	1526	1071	1647	9289	1340	863
8	17	1758	1018	1269	10999	1263	1221
9	20	1878	1421	2212	11691	1274	1623
10	24	2294	1466	2432	17016	863	1906

Time of execution per thread

Time of execution per thread

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Time of execution (sec)

1 2 3 4 5 6 7 8 9 10

Program

Top
Sign
Int.
Par.
Cong.

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

Overhead (%)

450
400
350
300
250
200
150
100
50
0

philo forkjoin barrier sync crypt sor elevator lufact montecarlo

Program

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
  - > Patterns of multithreaded programming
  - > Case studies of the Java Memory Model
  - > Incremental application
  - > 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

Question time

Thank you!

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