

Static analysis of the determinism of multithreaded programs

Pietro Ferrara

Università Ca' Foscari
Venice, Italy

École Normale Supérieure
Paris, France

Lunch Seminars, Venice, Italy


Motivations

- Multicore revolution
- Applications with explicit parallelism
 - Multithreading
- Difficult to reason about them
 - Random interleaving
 - Memory model
- Debug
 - Testing
 - Not all the executions may be exposed
 - Hard to reproduce an execution
 - Static analysis

An example

Thread Deposit1

Variable	Value
tempt1	
...	




```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	



```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

An example

Thread Deposit1

Variable	Value
tempt1	10.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

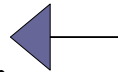
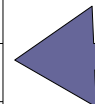
Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

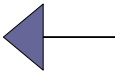
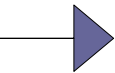
Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

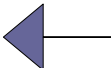
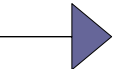
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

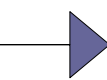
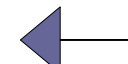
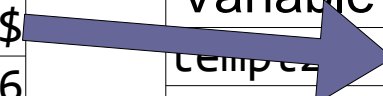
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	11.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

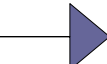


An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```



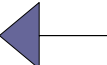
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	12.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

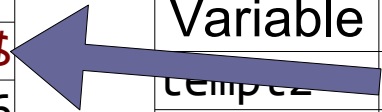
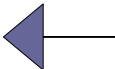
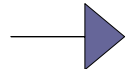
Shared Memory

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	12.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

Shared Memory

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

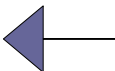
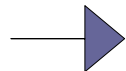
Thread Deposit2

Variable	Value
tempt2	12.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

At the end:

`a.amount=12.000$`



An example

Thread Deposit1

Variable	Value
tempt1	
...	

→
`int tempt1=a.amount;`
`tempt1=tempt1+1000$;`
`a.amount=tempt1;`

Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	

←
`int tempt2=a.amount;`
`tempt2=tempt2+1000$;`
`a.amount=tempt2;`

An example

Thread Deposit1

Variable	Value
tempt1	10.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

An example

Thread Deposit1

Variable	Value
tempt1	10.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	10.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

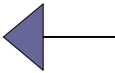
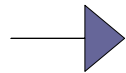
Shared Memory

Object	Field	Value
a	amount	10.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	10.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

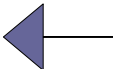
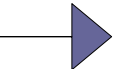
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	10.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

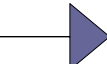


An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```



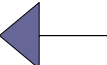
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	11.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

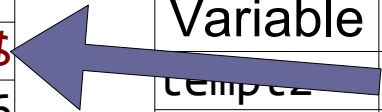
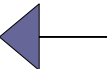
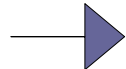
Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	11.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```



An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;  
tempt1=tempt1+1000$;  
a.amount=tempt1;
```

Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	11.000\$
...	

```
int tempt2=a.amount;  
tempt2=tempt2+1000$;  
a.amount=tempt2;
```

At the end:

`a.amount=11.000$`

An example

Thread Deposit1

Variable	Value
tempt1	11.000\$
...	

```
int tempt1=a.amount;
```

```
tempt1=tempt1+1000$;
```

```
a.amount
```

Shared Memory

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Thread Deposit2

Variable	Value
tempt2	11.000\$
...	

```
int tempt2=a.amount;
```

```
tempt2=tempt2+1000$;
```

```
a.amount
```

1st execution: a.amount=12.000\$

2nd execution: a.amount=11.000\$

Our solution

- **Statically** analyze the determinism
 - Focused on communications on shared memory
 - Via abstract interpretation
 - **Generic** w.r.t.
 - Programming language
 - Numerical domain
 - Memory model
- Advantages
 - Deal **directly** with the effects of random interleaving
 - **Flexible**

Concrete domain

$$S : [\text{Var} \rightarrow (V \times T)]$$

- S: shared memories
- Var: variables
- V: values
- T: thread identifiers
- Thread-partitioned trace domain
 - Relates each thread to its trace of execution

$$\Psi : [T \rightarrow S^{\vec{+}}]$$

- Concrete semantics: set of functions
 - All the possible executions

An example – Concrete semantics

Thread Deposit1:

Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1

Thread Deposit2:

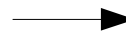
Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1



Obj.	Field	Value	Thread
a	amount	12.000\$	Deposit2

Thread Deposit1:

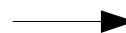
Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1

Thread Deposit2:

Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit2

First abstraction

$$\widehat{S} : [\text{Var} \rightarrow [T \rightarrow \widehat{V}]]$$

- Parameterized on an abstract domain $V^\#$
 - One value for each thread

$$\widehat{\Psi} : [T \rightarrow \widehat{S}^\ddagger]$$

- Abstract semantics: one function
 - Approximates all the possible executions
- **Sound** w.r.t. concrete domain

$$\langle \wp(\Psi), \sqsubseteq \rangle \begin{array}{c} \xleftarrow{\gamma_\Psi} \\ \xrightarrow{\alpha_\Psi} \end{array} \langle \widehat{\Psi}, \sqsubseteq_{\widehat{\Psi}} \rangle$$

An example – 1st abstraction

Thread Deposit1:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit1	[11.000..11.000]\$

Thread Deposit2:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$
		Deposit1	[11.000..11.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit2	[11.000..12.000]\$

Second abstraction

$$\bar{S} : [\text{Var} \rightarrow (\widehat{V} \times \wp(T))]$$

- Trace
 - One abstract value
 - The set of threads that may have written it

$$\bar{\Psi} : [T \rightarrow \bar{S}^{\vec{\tau}}]$$

- Sound

$$\langle \wp(\Psi), \sqsubseteq \rangle \xrightleftharpoons[\alpha_{\Psi}]{\gamma_{\Psi}} \langle \widehat{\Psi}, \sqsubseteq_{\widehat{\Psi}} \rangle \xrightleftharpoons[\alpha_{\widehat{\Psi}}]{\gamma_{\widehat{\Psi}}} \langle \bar{\Psi}, \sqsubseteq_{\bar{\Psi}} \rangle$$

An example – 2nd abstraction

Thread Deposit1:

Obj.	Field	Value	Threads
a	amount	[10.000..10.000]\$	{System}

 →

Obj.	Field	Value	Threads
a	amount	[11.000..11.000]\$	{Deposit1}

Thread Deposit2:

Obj.	Field	Value	Threads
a	amount	[1.....11.....]\$	{System, Deposit1}

 →

Obj.	Field	Value	Threads
a	amount	[11.000..12.000]\$	{Deposit2}

Determinism on concrete states

$$ds(s_1, s_2) = \text{false}$$



$$\exists \text{var} \in \text{dom}(s_1) \cap \text{dom}(s_2) : s_1(\text{var}) = (\text{val}_1, t_1), \\ s_2(\text{var}) = (\text{val}_2, t_2), t_1 \neq t_2$$

- A program is not deterministic iff
 - two executions
 - of the **same thread**
 - in the **same position** of the traces of execution
 - contains two shared memories that relate
 - the **same variable**
 - to **values written by different threads**

An example – Concrete semantics

Thread Deposit1:

Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1

Thread Deposit2:

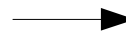
Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1



Obj.	Field	Value	Thread
a	amount	12.000\$	Deposit2

Thread Deposit1:

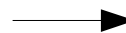
Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit1

Thread Deposit2:

Obj.	Field	Value	Thread
a	amount	10.000\$	System



Obj.	Field	Value	Thread
a	amount	11.000\$	Deposit2

Determinism on abstract states

- First abstraction

$$\widehat{ds}(\widehat{s}) = \text{false}$$



$$\exists \text{var} \in \text{dom}(\widehat{s}) : |\text{dom}(\widehat{s}(\text{var}))| > 1$$

- Second abstraction

$$\overline{ds}(\overline{s}) = \text{false} \Leftrightarrow \exists \text{var} \in \text{dom}(\overline{s}) : |\pi_2(\overline{s}(\text{var}))| > 1$$

- Soundness

$$\forall \theta \in \wp(\Psi) : d(\theta) = \text{false} \Rightarrow \widehat{d}(\alpha_{\Psi}(\theta)) = \text{false}$$

$$\forall f \in \widehat{\Psi} : \widehat{d}(f) = \text{false} \Rightarrow \overline{d}(\alpha_{\widehat{\Psi}}(f)) = \text{false}$$

An example – 1st abstraction

Thread Deposit1:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit1	[11.000..11.000]\$

Thread Deposit2:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$
		Deposit1	[11.000..11.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit2	[11.000..12.000]\$

An example – 2nd abstraction

Thread Deposit1:

Obj.	Field	Value	Threads
a	amount	[10.000..10.000]\$	{System}

 →

Obj.	Field	Value	Threads
a	amount	[11.000..11.000]\$	{Deposit1}

Thread Deposit2:

Obj.	Field	Value	Threads
a	amount	[10.000..11.000]\$	{System, Deposit1}

 →

Obj.	Field	Value	Threads
a	amount	[11.000..12.000]\$	{Deposit2}

Weak determinism

$$\widehat{wds} : [\widehat{S} \rightarrow \{\text{true}, \text{false}\}]$$

$$\widehat{wds}(\widehat{s}) = \text{false}$$



$$\exists \text{var} \in \text{dom}(\widehat{s}) : |\text{dom}(\widehat{s}(\text{var}))| > 1$$

$$\wedge \exists t_1, t_2 \in \text{dom}(\widehat{s}(\text{var})) : \widehat{s}(\text{var})(t_1) \neq \widehat{s}(\text{var})(t_2)$$

- **Relax** the full determinism
- On the first level of abstraction
- Rely on a **numerical abstract domain**

An example – 1st abstraction

Thread Deposit1:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit1	[11.000..11.000]\$

Thread Deposit2:

Obj.	Field	Thread	Value
a	amount	System	[10.000..10.000]\$
		Deposit1	[11.000..11.000]\$



Obj.	Field	Thread	Value
a	amount	Deposit2	[11.000..12.000]\$

An example – 1st abstraction

Thread Deposit1:

Obj.	Field	Thread	Value
a	amount	System	+



Obj.	Field	Thread	Value
a	amount	Deposit1	+

Thread Deposit2:

Obj.	Field	Thread	Value
a	amount	System	+
		Deposit1	+



Obj.	Field	Thread	Value
a	amount	Deposit2	+

Projecting states

- Check the determinism only
 - On a **subset of the shared variables**, e.g.
 - Only the amount of the bank account

$$\alpha_S^{stPrj} : [S \rightarrow S]$$

$$\alpha_S^{stPrj}(s) = \{s' : dom(s') \subseteq dom(s), \forall var \in dom(s') : s'(var) = s(var) \wedge stPrj(var) = true\}$$

$$dS_{stPrj} : [S \times S \rightarrow \{true, false\}]$$

$$dS_{stPrj}(s_1, s_2) = false$$



$$\exists var \in dom(\alpha_S^{stPrj}(s_1)) \cap dom(\alpha_S^{stPrj}(s_2)) : s_1(var) = (val_1, t_1), s_2(var) = (val_2, t_2), t_1 \neq t_2$$

Projecting traces

- Check the determinism only
 - On a **subset of the trace**
 - Only the actions that deposit and withdraw money

$$d_{trPrj} : [\wp(\Psi) \rightarrow \{\text{true}, \text{false}\}]$$

$$d_{trPrj}(\theta) = \text{false}$$



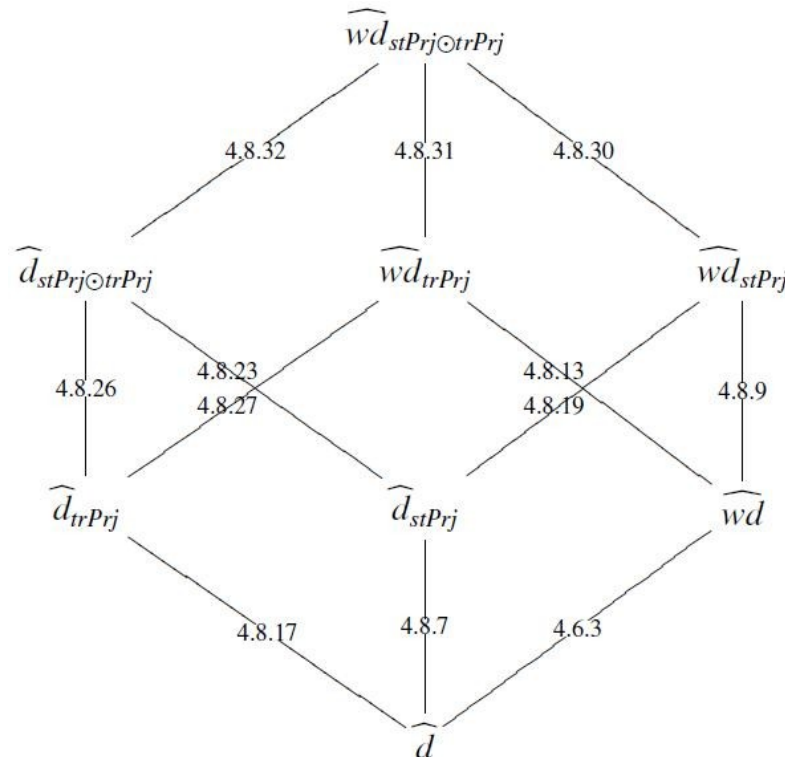
$$\exists f_1, f_2 \in \theta : \exists t \in \text{dom}(f_1) \cap \text{dom}(f_2) : \tau_1 = f_1(t), \tau_2 = f_2(t),$$

$$\exists i \in [0.. \min(\text{len}(\tau_1), \text{len}(\tau_2))] :$$

$$\text{trPrj}(i) = \text{true} \wedge ds(\tau_1(i), \tau_2(i)) = \text{false}$$

Global hierarchy

- Can be **composed**, e.g. check the determinism
 - On a . amount
 - When withdrawing
 - If the values written are always positive



Semi-automatic parallelization

- Given a sequential program
 - **Divide** it in two partitions - Input of the developer
 - **Analyze** them as executed in parallel
 - **Check** if the determinism is respected
 - Or one of its relaxations

Semi-automatic parallelization

- Given a sequential program
 - **Divide** it in two partitions - Input of the developer
 - **Analyze** them as executed in parallel
 - **Check** if the determinism is respected
 - Or one of its relaxations

Thread 1 → `acc.deposit(1.000$);`
`acc.deposit(1.000$);` ← Thread 2

Non-determinism on acc.amount

Semi-automatic parallelization

- Given a sequential program
 - **Divide** it in two partitions - Input of the developer
 - **Analyze** them as executed in parallel
 - **Check** if the determinism is respected
 - Or one of its relaxations

Thread 1 → `acc.deposit(1.000$);`
`acc.printAmount();` ← Thread 2



Non-determinism on stdout

Experimental results

Program	# St.	# Th.	An.	Det.	Weak Det.
philo	213	2	< 1''	< 1''	< 1''
tsp	1936	2	24''	< 1''	< 1''
elevator	1829	2	15''	< 1''	< 1''
barrier	363	3	< 1''	< 1''	< 1''
forkjoin	170	2	< 1''	< 1''	< 1''
sync	320	3	< 1''	< 1''	< 1''
sor	1121	2	4''	< 1''	< 1''
crypt	2636	3	4''	< 1''	< 1''
lufact	3732	2	26''	< 1''	< 1''
montecarlo	3864	2	46''	4''	4''
molodyn	9029	2	13'51''	50''	1'01''

Conclusion

- We present a novel deterministic property
 - Generic
 - Flexible
- Concrete and two levels of abstract semantics
 - Sound
- Many ways of **relaxing** it
 - On the information → Weak determinism
 - On the states
 - On the traces
- First step to semi-automatic **parallelization**
- **Applied** to a set of well-known benchmarks

PhD thesis – An overview

Pietro Ferrara

Università Ca' Foscari
Venice, Italy

École Normale Supérieure
Paris, France

Lunch Seminars, Venice, Italy

Summary

- Ch. 1/2: Introduction and Preliminaries
- Ch. 3: Static Analysis of the Happens-Before Memory Model
- Ch. 4: Determinism of Multithreaded Programs
- Ch. 5: Concrete and Abstract Domain and Semantics of Java Bytecode
- Ch. 6: Checkmate: a Generic Static Analyzer of Java Multithreaded Programs
- Ch. 7: Static Analysis of Unsafe Code
- Ch. 8: Conclusion

Ch. 3: Happens-Before MM

- Lunch seminar of 28/05/2008
- MM: which behaviours are allowed
- My contribution:
 - **Defining** the HBMM in a **fixpoint** form
 - **Abstracting** it with a computable semantics
 - 1st generic analysis sound w.r.t. HBMM

[Ferr08] P. Ferrara, "*Static analysis via abstract interpretation of the happens-before memory model*", in Springer editor, Proceedings of the 2nd International Conference on Tests and Proofs (TAP 2008), volume 4966 of Lecture Notes in Computer Science, Prato, Italy, April 9-11, 2008

Ch. 4: Determinism

- Lunch seminar... today!

[Ferr08b] P. Ferrara, "*Static analysis of the determinism of multithreaded programs*", in IEEE Computer Society, editor, Proceedings of the Sixth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2008), Cape Town, South Africa, November 10-14, 2008

Ch. 5: Java Bytecode

- **Low-level** domain and semantics
 - Java bytecode
 - Core: **alias analysis**
 - Threads are objects → identified by reference
 - Shared memory is the heap → accessed by reference
 - Monitors are object → identified by reference
- Initially applied to data race detection
- Then extended to build up a **generic analyzer**

[Ferr08a] P. Ferrara, "*A fast and precise analysis for data race detection*", in Elsevier editor, Proceedings of the Third Workshop on Bytecode Semantics, Verification, Analysis and Transformation (Bytecode'08), Electronic Notes in Theoretical Computer Science, Budapest, Hungary, April 6, 2008

Ch. 6: Checkmate

- 1st **generic** analyzer of **multithreaded** programs:
 - Property (determinism and weak determinism)
 - Numerical domain
 - Memory model (HBMM)
- Applied to
 - **Case studies** taken from the Java MM
 - Incremental application
 - **Benchmarks**
- Experimental results encouraging

P. Ferrara, "*Checkmate: a generic static analyzer of Java multithreaded programs*", to be submitted

Ch. 7: Unsafe Code

- Lunch seminar of 19/12/2007
- Direct access to the memory in .Net
- Static analysis of **buffer overrun**
- Implemented in Clousot
 - Generic industrial static analyzer of MSIL
- **Precise** and **scalable**
- Application of a generic static analyzer to industrial software

[FLF08] P. Ferrara, F. Logozzo and M. Fähndrich, "*Safer unsafe code for .NET*", in ACM Press, editor, Proceedings of the 23rd ACM Conference on Object-oriented Programming (OOPSLA 2008), Nashville, USA, October 19-23, 2008

Question time

Thank you!

“Wrong” program with data race

Thread Deposit1

```
int tempt1=a.amount;  
  
tempt1=tempt1+1000$;  
  
a.amount=tempt1;
```

Shared Memory

Thread Deposit2

```
int tempt2=a.amount;  
  
tempt2=tempt2+1000$;  
  
a.amount=tempt2;
```

Variable	Value
tempt1	11.000\$
...	

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	12.000\$
...	

Variable	Value
tempt1	11.000\$
...	

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	11.000\$
...	

“Correct” program without data race

Thread Deposit1

```
lock(a);  
int tempt1=a.amount;  
  
tempt1=tempt1+1000$;  
  
a.amount=tempt1;  
unlock(a);
```

Shared Memory

Thread Deposit2

```
lock(a);  
int tempt2=a.amount;  
  
tempt2=tempt2+1000$;  
  
a.amount=tempt2;  
unlock(a);
```

Variable	Value
tempt1	11.000\$
...	

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	12.000\$
...	

Variable	Value
tempt1	12.000\$
...	

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	11.000\$
...	

“Wrong” program without data race

Thread Deposit1

```
lock(a);  
int tempt1=a.amount;  
unlock(a);  
tempt1=tempt1+1000$;  
lock(a);  
a.amount=tempt1;  
unlock(a);
```

Shared Memory

Thread Deposit2

```
lock(a);  
int tempt2=a.amount;  
unlock(a);  
tempt2=tempt2+1000$;  
lock(a);  
a.amount=tempt2;  
unlock(a);
```

Variable	Value
tempt1	11.000\$
...	

Object	Field	Value
a	amount	12.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	12.000\$
...	

Variable	Value
tempt1	11.000\$
...	

Object	Field	Value
a	amount	11.000\$
	ID	58656
	...	
...		

Variable	Value
tempt2	11.000\$
...	