A proof system for exception handling in multithreaded *Java*

Erika Ábrahám Frank S. de Boer Willem-Paul de Roever Martin Steffen

Christian-Albrechts University Kiel

FSEN'05, Tehran

Oct. 2, 2005

Structure

introduction

The programming language

The assertional proof system

- assertion language
- verification conditions

conclusion

Motivation

- Safety-critical *Java* application areas
  $\longrightarrow$ need for verification
- model checking: mostly for finite state systems
- existing deductive methods: mostly for sequential *Java*

introduction

The programming language

The assertional proof system

assertion language

verification conditions

conclusion

Multithreaded core of *Java* with exceptions

- class-based object-oriented language
- dynamically heap-allocated objects, aliasing
- method invocation, recursion, self-calls

- multithreading
- wait & notify monitor synchronization

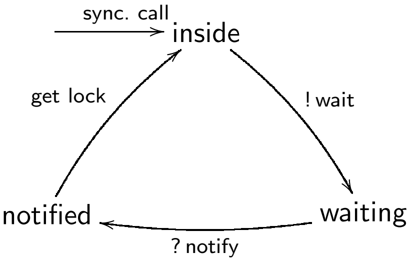
- here in particular: exception handling
- not covered (yet): inheritance, polymorphism, inner classes . . .

Multithreading

- threads = sequential sequence of actions
- method calls/returns: stack of method bodies, each with local variables
- running in parallel
- sharing instance states
- dynamically created as instances of thread classes (+ explicitely started)

Monitors

- each object can act as monitor:
  - mutual exclusion between *synchronized* methods of a single instance
  - monitor coordination via methods: wait, notify, notifyAll



Exceptions

```
class Inc extends Thread{
   int x;
   public void m(){
      E v;
      try{
         /*statements possibly throwing exceptions*/
         ... throw v; ...
      }
      catch (E1 u){/*handle exceptions of type E1*/}
      ...
      catch (En u){/*handle exceptions of type En*/}
      finally{/*clean up*/}
   }
}
class E extends Exception{...} ...
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;
        stmt;
    }
    catch (E u){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;     // v IS THROWN
        stmt;
    }
    catch (E u){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;      // v IS THROWN
        stmt;
    }
    catch (E u){   // v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;    // v IS THROWN
        stmt;
    }
    catch (E u){   // v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;    // v IS THROWN
        stmt;
    }
    catch (E u){   // v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
try{
    try{
        v = new E();
        throw v;     // v IS THROWN
        stmt;
    }
    catch (E u){   // v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
catch (E w){
    stmt;
}
finally{
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();
    }
    catch (E w){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();
    }
    catch (E w){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();
    }
    catch (E w){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;   //v IS THROWN
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();   //v IS RETHROWN
    }
    catch (E w){
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;   //v IS THROWN
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();   //v IS RETHROWN
    }
    catch (E w){    //v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;   //v IS THROWN
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();   //v IS RETHROWN
    }
    catch (E w){    //v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;   //v IS THROWN
    stmt;
}
```

Semantics: Exception handling

```
void m1(){
    try{
        this.m2();   //v IS RETHROWN
    }
    catch (E w){    //v IS CAUGHT
        stmt;
    }
    finally{
        stmt;
    }
    stmt;
}
void m2(){
    v = new E();
    throw v;   //v IS THROWN
    stmt;
}
```

Semantics: Exception handling

```
void m(){
    try{
        v = new E();
        throw v;
        stmt;
    }
    catch (E' u){
        stmt;
    }
    finally{
        u = new E''();
        throw u;
    }
    stmt;
}
```

Semantics: Exception handling

```
void m(){
    try{
        v = new E();
        throw v; //v IS THROWN
        stmt;
    }
    catch (E' u){
        stmt;
    }
    finally{
        u = new E''();
        throw u;
    }
    stmt;
}
```

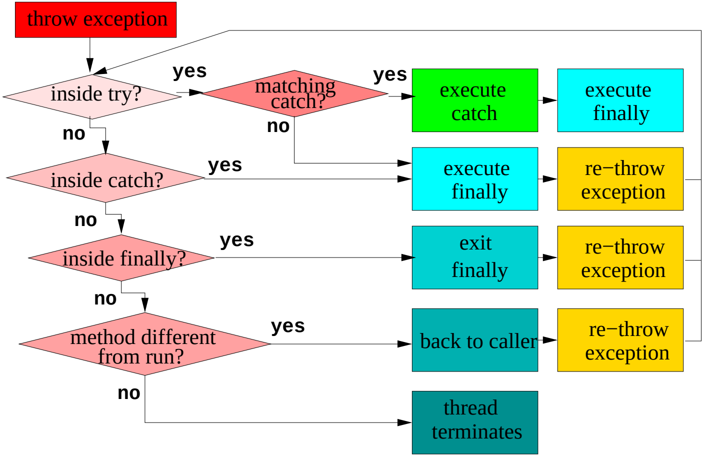
Semantics: Exception handling

```
void m(){
    try{
        v = new E();
        throw v; //v IS THROWN
        stmt;
    }
    catch (E' u){
        stmt;
    }
    finally{
        u = new E''();
        throw u; //u IS THROWN
    }
    stmt;
}
```

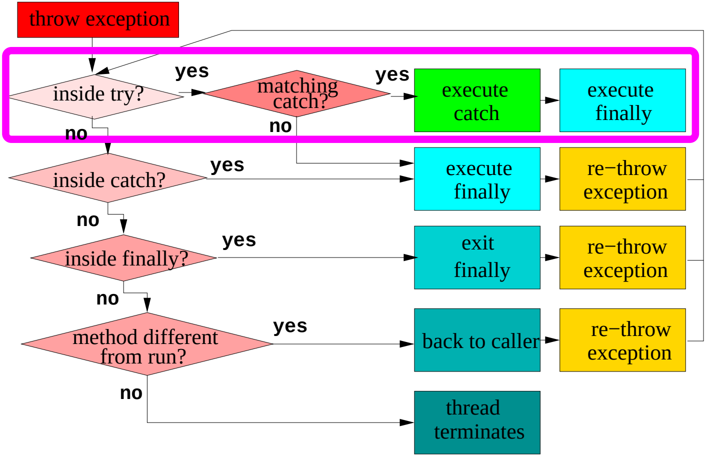
Semantics: Exception handling

```
void m(){
    try{
        v = new E();
        throw v;  //v IS THROWN
        stmt;
    }
    catch (E' u){
        stmt;
    }
    finally{
        u = new E''();
        throw u;  //u IS THROWN
    } //u IS RETHROWN
    stmt;
}
```

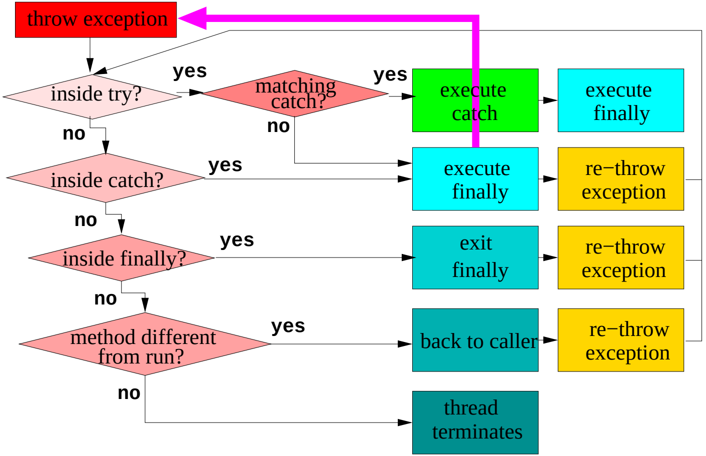
Semantics: Exception handling



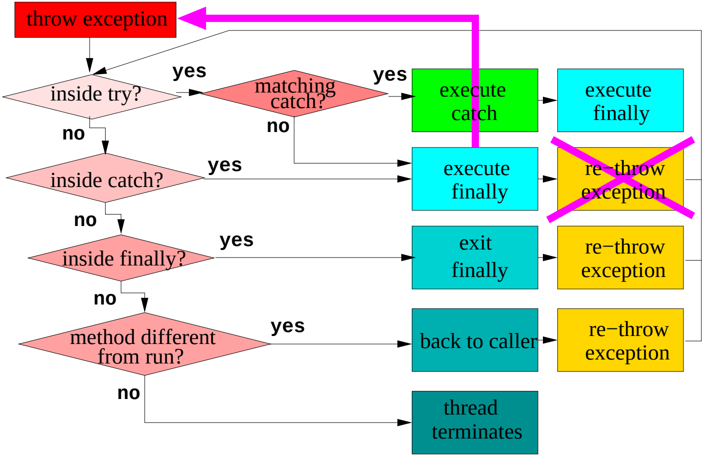
Semantics: Exception handling



Semantics: Exception handling



Semantics: Exception handling



Abstract syntax

$$
\begin{array}{rcl}
e & ::= & x \mid u \mid \mathsf{this} \mid \mathsf{null} \mid \mathsf{f}(e,\ldots,e) \\
e_{ret} & ::= & \epsilon \mid e \\
stm & ::= & x := e \\
 & \mid & u := e \mid u := \mathsf{new}^{c} \mid u := e.m(e,\ldots,e) \mid e.m(e,\ldots,e) \\
 & \mid & \mathsf{throw}\ e \\
 & \mid & \mathsf{try}\ stm\ \mathsf{catch}\ (c\ u)\ stm \ldots \mathsf{catch}\ (c\ u)\ stm\ \mathsf{finally}\ stm\ \mathsf{yrt} \\
 & \mid & \epsilon \mid stm; stm \mid \mathsf{if}\ e\ \mathsf{then}\ stm\ \mathsf{else}\ stm\ \mathsf{fi} \mid \mathsf{while}\ e\ \mathsf{do}\ stm\ \mathsf{od} \ldots \\
modif & ::= & \mathsf{nsync} \mid \mathsf{sync} \\
meth & ::= & modif\ m(u,\ldots,u)\{\ stm; \mathsf{return}\ e_{ret}\} \\
meth_{\mathsf{run}} & ::= & \mathsf{nsync\ run}()\{\ stm; \mathsf{return}\ \} \\
meth_{predef} & ::= & meth_{\mathsf{start}}\ meth_{\mathsf{wait}}\ meth_{\mathsf{notify}}\ meth_{\mathsf{notifyAll}} \\
class & ::= & \mathsf{class}\ c\{meth \ldots meth\ meth_{\mathsf{run}}\ meth_{predef}\} \\
class_{\mathsf{main}} & ::= & class \\
prog & ::= & \langle class \ldots class\ class_{\mathsf{main}} \rangle
\end{array}
$$

Semantics

states

local	τ	values of local vars
global	σ	values of instance vars for each *existing* object

configurations

local	(α, τ, stm)	local state + point of exec.
thread	$(\alpha_0, \tau_0, stm_0) \ldots (\alpha_n, \tau_n, stm_n)$	stack of local confs
global	$\langle T, \sigma \rangle$	set of thread confs + global state

Impressionistic view on SOS

$$\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{try } stm)\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,stm)\},\sigma\rangle \quad \text{TRY}$$

$$\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{yrt}; stm)\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,stm)\},\sigma\rangle \quad \text{FINALLY}_{out}$$

$$\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{yrt}_\beta; stm)\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{throw } \beta; stm)\},\sigma\rangle \quad \text{FINALLY}^{exc}_{out}$$

$$\frac{n \geq 0}{\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{catch}\,(c_1\ u_1)\ stm_1 \ldots \text{catch}\,(c_n\ u_n)\ stm_n \text{ finally } stm \text{ yrt}; stm')\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,stm \text{ yrt}; stm')\},\sigma\rangle} \ \text{FINALLY}_{in}$$

$$\frac{\begin{array}{c} stm \text{ is try-closed} \quad stm' = \text{catch}\,(c_1\ u_1)\ stm_1 \ldots \text{catch}\,(c_n\ u_n)\ stm_n \text{ finally } stm_{n+1} \text{ yrt} \\ 1 \leq i \leq n \quad [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} \in Val^{c_i} \quad \forall 1 \leq j < i.\, [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} \notin Val^{c_j} \\ \tau' = \tau[u_i \mapsto [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau}] \end{array}}{\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{throw } e; stmstm'; stm'')\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau',stm_i \text{ finally } stm_{n+1} \text{ yrt}; stm'')\},\sigma\rangle} \ \text{CATCH}$$

$$\frac{\begin{array}{c} stm \text{ is try-closed} \quad stm' = \text{catch}\,(c_1\ u_1)\ stm_1 \ldots \text{catch}\,(c_n\ u_n)\ stm_n \text{ finally } stm_{n+1} \text{ yrt} \\ [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} = \beta \neq null \quad 0 \leq n \quad \forall 1 \leq i \leq n.\, [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} \notin Val^{c_i} \end{array}}{\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{throw } e; stmstm'; stm'')\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,stm_{n+1}\text{yrt}_\beta; stm'')\},\sigma\rangle} \ \text{FINALLY}^{exc}_{in}$$

$$\frac{stm \text{ is try-closed} \quad [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} = \beta \neq null}{\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{throw } e; stm \text{ yrt}_{\beta'}; stm')\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{yrt}_\beta; stm')\},\sigma\rangle} \ \text{THROWFINALLY}$$

$$\frac{stm' \text{ is try-closed} \quad [\![e]\!]_{\mathcal{E}}^{\sigma(\beta),\tau'} = \gamma \neq null}{\langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{receive } u_{ret}; stm) \circ (\beta,\tau',\text{throw } e; stm')\},\sigma\rangle \longrightarrow \langle T \dot\cup \{\xi \circ (\alpha,\tau,\text{throw } \gamma; stm)\},\sigma\rangle} \ \text{RETURN}^{exc}$$

$$\frac{stm \text{ is try-closed} \quad [\![e]\!]_{\mathcal{E}}^{\sigma(\alpha),\tau} = \beta \neq null}{\langle T \dot\cup \{(\alpha,\tau,\text{throw } e; stm; \text{return})\},\sigma\rangle \longrightarrow \langle T \dot\cup \{(\alpha,\tau,\text{return}_\beta)\},\sigma\rangle} \ \text{TERMINATE}^{exc}$$

introduction

The programming language

The assertional proof system

- assertion language
- verification conditions

conclusion

The assertional proof system

Verification process:

1. define a proof outline through
   - augmentation by auxiliary variables and
   - annotation, which specifies invariant properties.
2. generate verification conditions for
   - initial correctness and
   - for the inductive step for
     - local correctness,
     - interference freedom test, and
     - cooperation test.
3. prove the verification conditions

Augmentation

Built-in auxiliary variables:

- local configuration id: aux. local variable
- thread id: aux. formal parameter
- identification of caller: aux. formal parameter
- capture monitor discipline: aux. instance variables
- exception handling: aux. local variables store the thrown but not yet caught exceptions

Augmentation

User-defined augmentation: (for exceptions)

- $\text{throw}\ e\ \langle \vec{y} := \vec{e} \rangle^{throw}$
- $\text{try} \ldots \text{yrt}\ \langle \vec{y} := \vec{e} \rangle^{rethrow}$
- $u := e_0.m(\vec{e});\ \langle \vec{y}_1 := \vec{e}_1 \rangle^{!call}\ \langle \vec{y}_4 := \vec{e}_4 \rangle^{?ret}\ \langle \vec{y}' := \vec{e}' \rangle^{rethrow}$
- $\langle \vec{y}_2 := \vec{e}_2 \rangle^{?call}\ stm; \text{return}\ e_{ret}\ \langle \vec{y}_3 := \vec{e}_3 \rangle^{!ret}$
- ...

The assertion language

Local sublanguage: properties of method execution

$$
\begin{array}{rcl}
exp_l & ::= & z \mid x \mid u \mid \mathsf{this} \mid \mathsf{null} \mid \mathsf{f}(exp_l, \ldots, exp_l) \\
ass_l & ::= & exp_l \mid \neg ass_l \mid ass_l \wedge ass_l \\
 & \mid & \exists z{:}\mathsf{Int}.\ ass_l \ldots \\
 & \mid & \exists (z{:}\mathsf{Object}) \in exp_l.\ ass_l \mid \exists (z{:}\mathsf{Object}) \sqsubseteq exp_l.\ ass_l
\end{array}
$$

Global sublanguage: properties of communication/object structure

$$
\begin{array}{rcl}
exp_g & ::= & z \mid exp_g.x \mid \mathsf{null} \mid \mathsf{f}(exp_g, \ldots, exp_g) \\
ass_g & ::= & exp_g \mid \neg ass_g \mid ass_g \wedge ass_g \mid \exists z.\ ass_g
\end{array}
$$

Annotation

An annotation assigns

- local assertions to all control points;
- a class invariant to each class;
- a global invariant to the program.

For example:

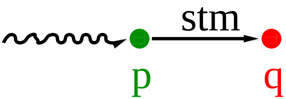
- $\{p_0\}$ throw e $\{p_1\}^{throw}$ $\langle \vec{y} := \vec{e}\rangle^{throw}$ $\{p_2\}$
- $\{p_0\}$ try ... yrt $\{p_1\}^{exc}$ $\{p_2\}^{rethrow}$ $\langle \vec{y} := \vec{e}\rangle^{rethrow}$ $\{p_3\}$
- $\{p_0\}$ $u := e_0.m(\vec{e}); \ldots$ $\{p_1\}^{exc}$ $\{p_2\}^{rethrow}$ $\langle \vec{y}' := \vec{e}'\rangle^{rethrow}$ $\{p_3\}$

Example: Proof outline

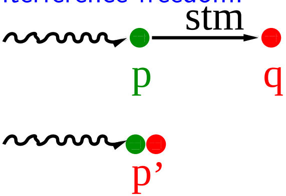
```
class Inc{
    int x;
    public void m(){
        try {
            while (true){
                inc(); {x == 100 ∧ hastype(exc, E) }^exc
            } {false}
        }
        catch (E u){{x == 100} }
        finally { {x == 100} } {x == 100}^exc {x == 100}
        return; }
    public synchronized void inc(){
        E v;
        if (x==100) { {x == 100}
          v = new E(); {x == 100 ∧ hastype(v, E) }
          throw v; {false}
        } {x != 100}
        x = x+1;
        return; }
}
```

Verification conditions

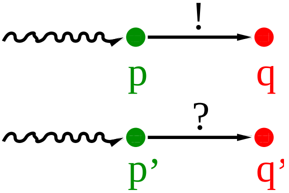
Local correctness:



Interference freedom:



Cooperation test:



Local correctness

$\{p\}$	try	$\{p_0\}$	$\ldots \{q_0\}$ throw $e \ldots$	$\{p'_0\}$	
	catch$(c_1\ u_1)$	$\{p_1\}$	stm_1	$\{p'_1\}$	$\ldots$
	catch$(c_n\ u_n)$	$\{p_n\}$	stm_n	$\{p'_n\}$	
	finally	$\{p_{\text{fin}}\}$	stm_{fin}	$\{p'_{\text{fin}}\}$	
	yrt			$\{p_{\text{exc}}\}^{exc}$	$\{p'\}$

$$\models_{\mathcal{L}} \quad \{q_0 \wedge e \neq \text{null} \wedge \text{hastype}(e, c_i) \wedge \forall 1 \leq j < i. \neg\, \text{hastype}(e, c_j)\}$$

$$u_i := e;$$

$$\{p_i\}$$

Local correctness

$$
\begin{array}{llllll}
\{p\} & \text{try} & \{p_0\} & \ldots\{q_0\}\ \text{throw}\ e\ldots & \{p_0'\} & \\
& \text{catch}(c_1\ u_1) & \{p_1\} & stm_1 & \{p_1'\} & \ldots \\
& \text{catch}(c_n\ u_n) & \{p_n\} & stm_n & \{p_n'\} & \\
& \text{finally} & \{p_{\text{fin}}\} & stm_{\text{fin}} & \{p_{\text{fin}}'\} & \\
& \text{yrt} & & & \{p_{\text{exc}}\}^{exc} & \{p'\}
\end{array}
$$

$$
\models_{\mathcal{L}} \quad \{q_0 \wedge e \neq \text{null} \wedge \text{hastype}(e, c_i) \wedge \forall 1 \leq j < i. \neg\, \text{hastype}(e, c_j)\}\ u_i := e;\ \{p_i\}
$$

$$
\models_{\mathcal{L}} \quad \{q_0 \wedge e \neq \text{null} \wedge \forall 1 \leq j \leq n. \neg\, \text{hastype}(e, c_j)\}\ u := e;\ \{p_{\text{fin}}\}
$$

Interference freedom

- Variables shared within one instance $\Rightarrow$ interference
- When exactly can different method executions interfere?
  - different threads, except one is starting the other
  - the same thread, except *matching* communication pairs

$$\models_{\mathcal{L}} pre(\vec{y} := \vec{e}) \wedge q' \wedge \text{interferes}(q', \vec{y} := \vec{e}) \rightarrow q'[\vec{e}/\vec{y}]$$

where $\text{interferes}(q', \vec{y} := \vec{e})$ is defined as

$$\begin{aligned} \text{thread} = \text{thread}' &\rightarrow \text{waits\_for\_ret}(q, \vec{y} := \vec{e}) \wedge \\ \text{thread} \neq \text{thread}' &\rightarrow \neg \text{self\_start}(q, \vec{y} := \vec{e})\,. \end{aligned}$$

Cooperation test for exceptions

caller: $u_{ret} := e_0.m(\vec{e})\ldots \quad \{p_1\}^{wait} \quad \{p_2\}^{?ret}\ \langle \vec{y}_4 := \vec{e}_4\rangle^{?ret} \quad \{p_3\}^{exc}$

callee: $m(\vec{u})\{\ldots \quad \{q_1\}\ \text{throw}\ e \quad \{q_2\}^{throw}\ \langle \vec{y}_3 := \vec{e}_3\rangle^{throw} \quad \ldots\}$

$$\models_{\mathcal{G}} \quad \{GI \wedge P_1(z) \wedge Q_1'(z') \wedge \text{comm}\}\ \ \text{exc} := E'(z')\ \ \{P_2(z) \wedge Q_2'(z')\}$$

$$\models_{\mathcal{G}} \quad \{GI \wedge P_1(z) \wedge Q_1'(z') \wedge \text{comm}\}\ \ \text{exc} := E'(z');\quad z'.\vec{y}_3' := \vec{E}_3'(z');\quad z.\vec{y}_4 := \vec{E}_4(z)\ \ \{GI \wedge P_3(z)\}$$

with

$$\text{comm} = E_0(z) = z' \wedge \vec{u}' = \vec{E}(z) \wedge E'(z') \neq \text{null} \wedge z \neq \text{null} \wedge z' \neq \text{null}$$

Cooperation test for exceptions

caller:	$u_{ret} := e_0.m(\vec{e})\ldots$	$\{p_1\}^{wait}$	$\{p_2\}^{?ret}$	$\langle \vec{y}_4 := \vec{e}_4\rangle^{?ret}$	$\{p_3\}^{exc}$
callee:	$m(\vec{u})\{\ldots$	$\{q_1\}$ throw e	$\{q_2\}^{throw}$	$\langle \vec{y}_3 := \vec{e}_3\rangle^{throw}$	$\ldots\}$

$$\models_{\mathcal{G}} \quad \begin{array}{c}\{GI \wedge P_1(z) \wedge Q_1'(z') \wedge \mathrm{comm}\}\\ \mathrm{exc} := E'(z')\\ \{P_2(z) \wedge Q_2'(z')\}\end{array}$$

$$\models_{\mathcal{G}} \quad \begin{array}{c}\{GI \wedge P_1(z) \wedge Q_1'(z') \wedge \mathrm{comm}\}\\ \mathrm{exc} := E'(z'); \quad z'.\vec{y}_3' := \vec{E}_3'(z'); \quad z.\vec{y}_4 := \vec{E}_4(z)\\ \{GI \wedge P_3(z)\}\end{array}$$

with

$$\begin{array}{rcl}\mathrm{comm} & = & E_0(z) = z' \wedge \vec{u}' = \vec{E}(z) \wedge E'(z') \neq \mathrm{null} \wedge \\ & & z \neq \mathrm{null} \wedge z' \neq \mathrm{null}\end{array}$$

Results & tool support

- modular proof system

> $\Rightarrow$ *The proof system is sound and (relative) complete*

- *Verger*
  - takes a proof outline as input,
  - generates the verification conditions, which are
  - verified in *PVS* interactively
  - no exceptions yet

Related work

- Pierik, de Boer [8]
  - inheritance, subtyping
  - sequential
- de Boer, Amerika (Pool) [4] ...
- exceptions in Jacobs/Huisman [7, 6]
- Poetzsch-Heffter, Müller [9], sequential *Java*.
- M. Huismann, B. Jacobs, et.al (Loop, PVS+Isabelle) [5] ...
- etc.

Conclusion

Future work:

- PVS optimization
- automatic generation of annotation/augmentation
- inheritance etc.
- compositionality

References I

[1] E. Ábrahám.
An Assertional Proof System for Multithreaded Java — Theory and Tool Support.
PhD thesis, University of Leiden, 2004.
defended 20.1.2005.

[2] E. Ábrahám, F. S. de Boer, W.-P. de Roever, and M. Steffen.
An assertion-based proof system for multithreaded Java.
Theoretical Computer Science, 331, 2005.

[3] E. Ábrahám, F. S. de Boer, W.-P. de Roever, and M. Steffen.
Inductive proof outlines for exceptions in multithreaded Java.
In F. Arbab and M. Sirjani, editors, *FSEN '05: IPM International Workshop on Foundations of Software Engineering (Theory and Practice). Oct. 1 – 3, 2005)*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2005.

[4] P. America and F. S. de Boer.
Reasoning about dynamically evolving process structures.
Formal Aspects of Computing, 6(3):269–316, 1993.

[5] U. Hensel, M. Huisman, B. Jacobs, and H. Tews.
Reasoning about classes in object-oriented languages: Logical models and tools.
In C. Hankin, editor, *Proceedings of ESOP '98*, volume 1381 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[6] M. Huisman and B. Jacobs.
Java program verification via a Hoare logic with abrupt termination.
In T. Maibaum, editor, *Proceedings of FASE'00*, volume 1783 of *Lecture Notes in Computer Science*, pages 284–303. Springer-Verlag, 2000.

References II

[7] B. Jacobs.
A formalisation of Java's exception mechanism.
In D. Sands, editor, *Proceedings of ESOP 2001*, volume 2028 of *Lecture Notes in Computer Science*, pages 284–301. Springer-Verlag, 2001.

[8] C. Pierik and F. S. de Boer.
A syntax-directed Hoare logic for object-oriented programming concepts.
In E. Najm, U. Nestmann, and P. Stevens, editors, *FMOODS '03*, volume 2884 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, Nov. 2003.
An extended version appeared as University of Utrecht Technical Report UU-CS-2003-010.

[9] A. Poetzsch-Heffter and P. Müller.
A programming logic for sequential Java.
In S. Swierstra, editor, *Programming Languages and Systems*, volume 1576 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1999.