Provably Live Exception Handling

Bart Jacobs

DistriNet, KU Leuven

FTfJP 2015

Contents

1. Problem and Proposed Fix
2. Verification Approach (Ignoring Exceptions)
3. Verification Approach (With Exceptions)

Contents

1. Problem and Proposed Fix

2. Verification Approach (Ignoring Exceptions)

3. Verification Approach (With Exceptions)

Problem Statement

Does this Scala program, running on the JVM, always terminate?

```
val queue = new LinkedBlockingQueue[String]()
fork { queue.put(”Hello, world”) }
queue.take()
```

Problem Statement

Does this Scala program, running on the JVM, always terminate?

```
val queue = new LinkedBlockingQueue[String]()
fork { queue.put("Hello, world") }
queue.take()
```

- What if put fails?

Problem Statement

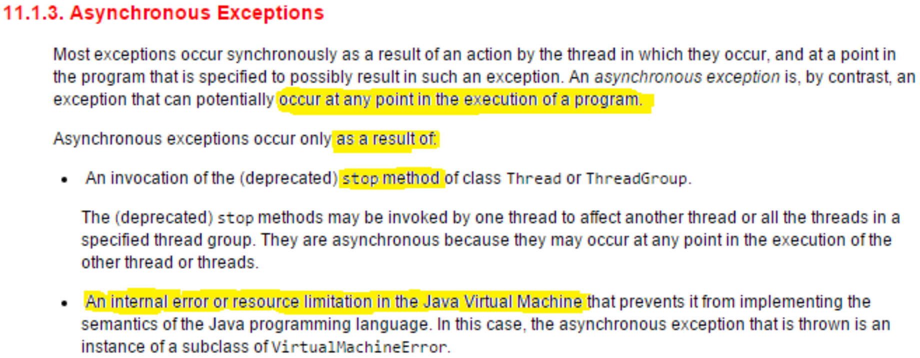
Does this Scala program, running on the JVM, always terminate?

```
val queue = new LinkedBlockingQueue[String]()
fork { queue.put(”Hello, world”) }
queue.take()
```

- What if put fails?
- What if the JVM fails?

Problem Statement

The Java Language Specification, Java SE 8 Edition:



11.1.3. Asynchronous Exceptions

Most exceptions occur synchronously as a result of an action by the thread in which they occur, and at a point in the program that is specified to possibly result in such an exception. An *asynchronous exception* is, by contrast, an exception that can potentially occur at any point in the execution of a program.

Asynchronous exceptions occur only as a result of:

- An invocation of the (deprecated) `stop` method of class `Thread` or `ThreadGroup`.

  The (deprecated) `stop` methods may be invoked by one thread to affect another thread or all the threads in a specified thread group. They are asynchronous because they may occur at any point in the execution of the other thread or threads.

- An internal error or resource limitation in the Java Virtual Machine that prevents it from implementing the semantics of the Java programming language. In this case, the asynchronous exception that is thrown is an instance of a subclass of `VirtualMachineError`.

First Fix Attempt

```
val queue = new LinkedBlockingQueue[String]()
fork {
  try {
    queue.put(”Hello, world”)
  } catch {
    case _ ⇒ System.exit(1)
  }
}
queue.take()
```

Proposed Fix

```
new Failbox().enter {
    val queue = new LinkedBlockingQueue[String]()
    Failbox.fork { queue.put(”Hello, world”) }
    queue.take()
}
```

Failboxes (Partial)

```
class Failbox {
  val threads = new ArrayList[Thread]()
  def enter(body: ⇒ Unit) {
    synchronized { threads.add(Thread.currentThread()) }
    try {
      try { body } finally {
        synchronized { threads.remove(Thread.currentThread()) }
      }
    } catch {
      case e ⇒
        synchronized { for (t ← threads) t.interrupt() }
        throw e
    }
  }
}
```

Proposed Fix

```
new Failbox().enter {
    val queue = new LinkedBlockingQueue[String]()
    Failbox.fork { queue.put(”Hello, world”) }
    queue.take()
}
```

Proposed Fix (Alternative)

```
val fb = new Failbox()
val queue = new LinkedBlockingQueue[String]()
fb.enter {
  Failbox.fork { queue.put("Hello, world") }
}
fb.enter {
  queue.take()
}
```

Failboxes (Partial)

```
class Failbox {
  var failed = false
  val threads = new ArrayList[Thread]()
  def enter(body: ⇒ Unit) {
    synchronized { if (failed) throw new FailboxException
                   threads.add(Thread.currentThread())       }
    try {
      try { body } finally {
        synchronized { threads.remove(Thread.currentThread()) }
      }
    } catch {
      case e ⇒
        failed = true
        synchronized { for (t ← threads) t.interrupt() }
        throw e
    }
  }
}
```

Proposed Fix (Alternative)

```
val fb = new Failbox()
val queue = new LinkedBlockingQueue[String]()
fb.enter {
   Failbox.fork { queue.put("Hello, world") }
}
fb.enter {
   queue.take()
}
```

Contents

1 Problem and Proposed Fix

2 Verification Approach (Ignoring Exceptions)

3 Verification Approach (With Exceptions)

Formal Programming Language

$$c ::= \textbf{new sema} \mid \textbf{fork}\ c \mid x.\textbf{V} \mid x.\textbf{P} \mid \textbf{let}\ x := c\ \textbf{in}\ c$$

Formal Programming Language

$$c ::= \textbf{new sema} \mid \textbf{fork } c \mid x.\mathbf{V} \mid x.\mathbf{P} \mid \textbf{let } x := c \textbf{ in } c$$

$$\textbf{let } \mathsf{s} := \textbf{new sema in fork } \mathsf{s}.\mathbf{V};\ \mathsf{s}.\mathbf{P}$$

Programming Language Semantics

$$(h,\, T \uplus \{\!\{\mathbf{new\ sema}; \xi\}\!\}) \rightsquigarrow (h \uplus \{s \mapsto 0\},\, T \uplus \{\!\{s; \xi\}\!\})$$

$$(h,\, T \uplus \{\!\{\mathbf{fork}\ c; \xi\}\!\}) \rightsquigarrow (h,\, T \uplus \{\!\{\mathrm{tt}; \xi, c; \mathbf{done}\}\!\})$$

$$(h \uplus \{s \mapsto n\},\, T \uplus \{\!\{x.\mathbf{V}; \xi\}\!\}) \rightsquigarrow (h \uplus \{s \mapsto n+1\},\, T \uplus \{\!\{\mathrm{tt}; \xi\}\!\})$$

$$(h \uplus \{s \mapsto n+1\},\, T \uplus \{\!\{x.\mathbf{P}; \xi\}\!\}) \rightsquigarrow (h \uplus \{s \mapsto n\},\, T \uplus \{\!\{\mathrm{tt}; \xi\}\!\})$$

$$(h,\, T \uplus \{\!\{\mathbf{let}\ x := c\ \mathbf{in}\ c'; \xi\}\!\}) \rightsquigarrow (h,\, T \uplus \{\!\{c; \mathbf{let}\ x := []\ \mathbf{in}\ c'; \xi\}\!\})$$

$$(h,\, T \uplus \{\!\{v; \mathbf{let}\ x := []\ \mathbf{in}\ c'; \xi\}\!\}) \rightsquigarrow (h,\, T \uplus \{\!\{c'[v/x]; \xi\}\!\})$$

$$(h,\, T \uplus \{\!\{v; \mathbf{done}\}\!\}) \rightsquigarrow (h,\, T)$$

$$c\ \text{deadlocks} \Leftrightarrow \exists h, T.\ (\emptyset, \{\!\{c; \mathbf{done}\}\!\}) \rightsquigarrow^* (h,\, T) \wedge (h,\, T) \not\rightsquigarrow \wedge T \neq \emptyset$$

Assertion Language

$$P ::= b \mid s.\mathrm{credit} \mid \mathrm{obs}(S) \mid P * P$$

Ghost Reachability

$$\frac{P \Rightarrow P'}{P \sqsubseteq P'} \qquad \frac{P \sqsubseteq P'}{P * R \sqsubseteq P' * R}$$

$$\mathbf{obs}(S) \sqsupseteq\sqsubseteq \mathbf{obs}(S \uplus \{\!\{s\}\!\}) * s.\mathrm{credit} \qquad \frac{P \sqsubseteq P' \qquad P' \sqsubseteq P''}{P \sqsubseteq P''}$$

Proof Rules (1/2)

$$\vdash \{\mathrm{true}\}\ \textbf{new sema}\ \{\mathrm{w}(\mathrm{res}) = w\}$$

$$\frac{\vdash \{\mathrm{obs}(S') * P\}\ c\ \{\mathrm{obs}(\emptyset) * \mathrm{true}\}}{\vdash \{\mathrm{obs}(S \uplus S') * P\}\ \textbf{fork}\ c\ \{\mathrm{obs}(S)\}} \qquad \vdash \{\mathrm{true}\}\ s.\mathbf{V}\ \{s.\mathrm{credit}\}$$

$$\vdash \{\mathrm{obs}(S) * s.\mathrm{credit} \wedge \mathrm{w}(s) < \mathrm{w}(S)\}\ s.\mathbf{P}\ \{\mathrm{obs}(S)\}$$

Proof Rules (2/2)

$$\frac{\vdash \{P\}\ c\ \{Q\} \qquad \forall v.\ \vdash \{Q[v/\mathrm{res}]\}\ c'[v/x]\ \{R\}}{\vdash \{P\}\ \mathbf{let}\ x := c\ \mathbf{in}\ c'\ \{R\}}$$

$$\frac{\vdash \{P\}\ c\ \{Q\}}{\vdash \{P * R\}\ c\ \{Q * R\}} \qquad \frac{P \sqsubseteq P' \qquad \vdash \{P'\}\ c\ \{Q\} \qquad Q \sqsubseteq Q'}{\vdash \{P\}\ c\ \{Q'\}}$$

Soundness

Lemma (Invariant)

$$\forall s.\ \#\mathrm{credits}(s) \leq \#\mathrm{obligations}(s) + \mathrm{value}(s)$$

Theorem (Soundness)

If $\vdash \{\mathrm{obs}(\emptyset)\}\ c\ \{\mathrm{obs}(\emptyset) * \mathrm{true}\}$, *then* c *does not deadlock.*

Proof.

Deadlock implies cycle in wait graph, implies false. □

Example Proof

```
$\{\mathsf{obs}(\emptyset)\}$
let s := new sema in
$\{\mathsf{obs}(\emptyset)\}$
$\{\mathsf{obs}(\{\!\!\{s\}\!\!\}) * \mathsf{s.credit}\}$
fork
    $\{\mathsf{obs}(\{\!\!\{s\}\!\!\})\}$
    s.V;
    $\{\mathsf{obs}(\{\!\!\{s\}\!\!\}) * \mathsf{s.credit}\}$
    $\{\mathsf{obs}(\emptyset)\}$
$\{\mathsf{obs}(\emptyset) * \mathsf{s.credit}\}$
s.P
$\{\mathsf{obs}(\emptyset)\}$
```

Contents

1. Problem and Proposed Fix
2. Verification Approach (Ignoring Exceptions)
3. Verification Approach (With Exceptions)

Formal Programming Language

$$c ::= \cdots \mid \textbf{new failbox} \mid x.\textbf{enter}\ c \mid \textbf{throw} \mid \textbf{fork}_{\overline{x}}\ c$$

Formal Programming Language

$$c ::= \cdots \mid \textbf{new failbox} \mid x.\textbf{enter}\ c \mid \textbf{throw} \mid \textbf{fork}_{\overline{x}}\ c$$

```
let fb := new failbox in
fb.enter (
   let s := new sema in
   fork$_{\mathrm{fb}}$ s.V;  s.P
)
```

Programming Language Semantics (1/2)

$$\begin{array}{ll} & (h, F, T \uplus \{\!\!\{(\overline{f}, \textbf{new failbox}; \xi)\}\!\!\}) \\ \rightsquigarrow & (h, F \uplus \{f \mapsto \mathrm{ok}\}, T \uplus \{\!\!\{(\overline{f}, f; \xi)\}\!\!\}) \end{array}$$

$$\begin{array}{ll} & (h, F, T \uplus \{\!\!\{(\overline{f}, f.\textbf{enter}\ c; \xi)\}\!\!\}) \\ \rightsquigarrow & (h, F, T \uplus \{\!\!\{(f \cdot \overline{f}, c; \textbf{leave}_{f}; \xi)\}\!\!\}) \end{array}$$

$$\begin{array}{ll} & (h, F, T \uplus \{\!\!\{(\overline{f} \cdot \overline{f}', v; \textbf{leave}_{\overline{f}}; \xi)\}\!\!\}) \\ \rightsquigarrow & (h, F, T \uplus \{\!\!\{(\overline{f}', v; \xi)\}\!\!\}) \end{array}$$

$$\begin{array}{ll} & (h, F, T \uplus \{\!\!\{(\overline{f}, \textbf{fork}_{\overline{f}'}\ c; \xi)\}\!\!\}) \\ \rightsquigarrow & (h, F, T \uplus \{\!\!\{(\overline{f}, \mathrm{tt}; \xi), (\overline{f}', c; \textbf{leave}_{\overline{f}'}; \textbf{done})\}\!\!\}) \end{array}$$

Programming Language Semantics (2/2)

$$
\begin{aligned}
& (h, F, T \uplus \{\!\![(\overline{f}, c; \xi)]\!\!\}) \\
\overset{\text{fail}}{\leadsto}\quad & (h, F, T \uplus \{\!\![(\overline{f}, \textbf{throw}; \xi)]\!\!\})
\end{aligned}
$$

$$
\begin{aligned}
& (h, F, T \uplus \{\!\![(\overline{f}, \textbf{throw}; \textbf{let } x := [] \textbf{ in } c'; \xi)]\!\!\}) \\
\leadsto\quad & (h, F, T \uplus \{\!\![(\overline{f}, \textbf{throw}; \xi)]\!\!\})
\end{aligned}
$$

$$
\begin{aligned}
& (h, F, T \uplus \{\!\![(f \cdot \overline{f}, c; \xi)]\!\!\}) \\
\leadsto\quad & (h, F, T \uplus \{\!\![(f \cdot \overline{f}, \textbf{throw}; \xi)]\!\!\}) \\
& \text{if } F(f) = \text{failed}
\end{aligned}
$$

$$
\begin{aligned}
& (h, F, T \uplus \{\!\![(\overline{f} \cdot \overline{f}', \textbf{throw}; \textbf{leave}_{\overline{f}}; \xi)]\!\!\}) \\
\leadsto\quad & (h, F[\overline{f} := \text{failed}], T \uplus \{\!\![(\overline{f}', \textbf{throw}; \xi)]\!\!\})
\end{aligned}
$$

$$
\begin{aligned}
& (h, F, T \uplus \{\!\![(\epsilon, \tilde{v}; \textbf{done})]\!\!\}) \\
\leadsto\quad & (h, F, T)
\end{aligned}
$$

Assertion Language

$$P ::= b \mid s.\mathrm{credit} \mid \mathrm{obs}(\overline{f}, S) \mid P * P$$

Ghost Reachability

$$\mathbf{obs}(\overline{f}, S) \land \mathsf{f}(S') \subseteq \overline{f} \sqsupseteq\sqsubseteq \mathbf{obs}(\overline{f}, S \uplus S') * S'.\mathsf{credit}$$

Proof Rules

$$\vdash \{\mathrm{true}\}\ \textbf{new sema}\ \{\mathrm{w}(\mathrm{res}) = w \wedge \mathrm{f}(\mathrm{res}) = f\}$$

$$\frac{\mathrm{f}(S') \subseteq \overline{f}' \qquad \vdash \{\mathrm{obs}(\overline{f}', S') * P\}\ c\ \{\mathrm{obs}(\overline{f}', \emptyset) * \mathrm{true}\}}{\vdash \{\mathrm{obs}(\overline{f}, S \uplus S') * P\}\ \textbf{fork}_{\overline{f}'}\ c\ \{\mathrm{obs}(\overline{f}, S)\}}$$

$$\vdash \{\mathrm{true}\}\ s.\textbf{V}\ \{s.\mathrm{credit}\}$$

$$\vdash \{\mathrm{obs}(\mathrm{f}(s) \cdot \overline{f}, S) * s.\mathrm{credit} \wedge \mathrm{w}(s) < \mathrm{w}(S)\}\ s.\textbf{P}\ \{\mathrm{obs}(\mathrm{f}(s) \cdot \overline{f}, S)\}$$

$$\vdash \{\mathrm{true}\}\ \textbf{new failbox}\ \{\mathrm{true}\}$$

$$\frac{\vdash \{\mathrm{obs}(f \cdot \overline{f}, S) * P\}\ c\ \{\mathrm{obs}(f \cdot \overline{f}, S') * Q \wedge \mathrm{f}(S') \subseteq \overline{f}\}}{\vdash \{\mathrm{obs}(\overline{f}, S) * P\}\ f.\textbf{enter}\ c\ \{\mathrm{obs}(\overline{f}, S') * Q\}}$$

$$\vdash \{\mathrm{true}\}\ \textbf{throw}\ \{\mathrm{false}\}$$

Soundness

Lemma (Invariants)

- $\forall t.\ t.\mathrm{obs}(\overline{f}, S) \Rightarrow \mathrm{f}(S) \subseteq \overline{f}$
- $\forall s.\ F(s) = \mathrm{ok} \Rightarrow \#\mathrm{credits}(s) \leq \#\mathrm{obligations}(s) + \mathrm{value}(s)$

Theorem (Soundness)

If $\vdash \{\mathrm{obs}(\epsilon, \emptyset)\}\ c\ \{\mathrm{obs}(\epsilon, \emptyset) * \mathrm{true}\}$*, then* c *does not deadlock.*

Example Proof

```
$\{\mathsf{obs}(\epsilon, \emptyset)\}$
let fb := new failbox in
$\{\mathsf{obs}(\epsilon, \emptyset)\}$
fb.enter (
    $\{\mathsf{obs}(\mathsf{fb}, \emptyset)\}$
    let s := new sema in
    $\{\mathsf{obs}(\mathsf{fb}, \emptyset) \land \mathsf{f}(\mathsf{s}) = \mathsf{fb}\}$
    $\{\mathsf{obs}(\mathsf{fb}, \{\!\!\{\mathsf{s}\}\!\!\}) * \mathsf{s.credit}\}$
    fork
        $\{\mathsf{obs}(\mathsf{fb}, \{\!\!\{\mathsf{s}\}\!\!\})\}$
        s.V;
        $\{\mathsf{obs}(\mathsf{fb}, \{\!\!\{\mathsf{s}\}\!\!\}) * \mathsf{s.credit}\}$
        $\{\mathsf{obs}(\mathsf{fb}, \emptyset)\}$
    $\{\mathsf{obs}(\mathsf{fb}, \emptyset) * \mathsf{s.credit}\}$
    s.P
    $\{\mathsf{obs}(\mathsf{fb}, \emptyset)\}$
)
$\{\mathsf{obs}(\epsilon, \emptyset)\}$
```

Conclusion

Contributions

- First sound verification approach for deadlock-freedom of concurrent Java programs with wait-signal-style synchronisation
- Encoded as API specs into VeriFast for Java; verified the example program

Future Work

- Integration with locks and thread joining
- Catching exceptions
- Further experimentation and validation

Acknowledgements

Thanks to anonymous reviewer for suggesting using semaphores as obligations instead of wait levels!