Specifying and Exploiting Advice-Execution Ordering using Dependency State Machines

Eric Bodden
Finite-state properties

“When disconnecting a connection c, don’t write to c until c is reconnected.”
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }

Advice-execution ordering matters
Advice-execution ordering matters

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
  call(* Connection.close()) && target(c) {
    closed.add(c);
  }

after(Connection c) returning:
  call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
  }

after(Connection c) returning:
  call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
      error("May not write to "+c+, as it is closed!");
  }

only "externally visible" code

Montag, 15. März 2010
Advice-execution ordering matters

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }

only “externally visible” code
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:

call(* Connection.close()) && target(c) {
    closed.add(c);
}

after(Connection c) returning:

call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
}

after(Connection c) returning:

call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
        error("May not write to "+c+, as it is closed!");
}
Advice-execution ordering matters

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to " + c + ", as it is closed!");
    }

only "externally visible" code
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
  call(* Connection.close()) && target(c) {
    closed.add(c);
  }

after(Connection c) returning:
  call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
  }

after(Connection c) returning:
  call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
      error("May not write to "+c+", as it is closed!");
  }

Advice-execution ordering matters

only "externally visible" code
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:

call(* Connection.close()) && target(c) {
    closed.add(c);
}

after(Connection c) returning:

call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
}

after(Connection c) returning:

call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
        error("May not write to "+c+", as it is closed!");
}
Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
  call(* Connection.close()) && target(c) {
    closed.add(c);
  }

after(Connection c) returning:
  call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
  }

after(Connection c) returning:
  call(* Connection.write(..)) && target(c) {
    if(closed.contains(c))
      error("May not write to "+c+-", as it is closed!);
  }
Advice-execution ordering matters

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+, as it is closed!");
    }

only "externally visible" code
Advice-execution ordering matters

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+, as it is closed!");
    }

only "externally visible" code
Monitoring Aspects

JavaMOP

Various spec. languages

J-L0

LTL

S2A

LSCs

M2Aspects

MSCs

MOFScript

Java-STAIRS aspects

Tracematches

Relational aspects

abc
Dependency State Machines (DSMs)

Set closed = new WeakIdentityHashSet();

after(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

after(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

after(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }
Dependency State Machines (DSMs)

Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
   call(* Connection.close()) && target(c) {
      closed.add(c);
   }

dependent after reconnect(Connection c) returning:
   call(* Connection.reconnect()) && target(c) {
      closed.remove(c);
   }

dependent after write(Connection c) returning:
   call(* Connection.write(..)) && target(c) {
      if(closed.contains(c))
         error("May not write to "+c+", as it is closed!");
   }
Dependency State Machines (DSMs)

Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+, as it is closed!");
    }

Montag, 15. März 2010
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
   call(* Connection.close()) && target(c) {
      closed.add(c);
   }

dependent after reconnect(Connection c) returning:
   call(* Connection.reconnect()) && target(c) {
      closed.remove(c);
   }

dependent after write(Connection c) returning:
   call(* Connection.write(..)) && target(c) {
      if(closed.contains(c))
         error("May not write to "+c+, as it is closed!");
   }
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
call(* Connection.close()) && target(c) {
    closed.add(c);
}

dependent after reconnect(Connection c) returning:
call(* Connection.reconnect()) && target(c) {
    closed.remove(c);
}

dependent after write(Connection c) returning:
call(* Connection.write(..)) && target(c) {
    if (closed.contains(c))
        error("May not write to "+c+" as it is closed!");
}
Set closed = new WeakIdentityHashSet();

dependent after disconnect(Connection c) returning:
    call(* Connection.close()) && target(c) {
        closed.add(c);
    }

dependent after reconnect(Connection c) returning:
    call(* Connection.reconnect()) && target(c) {
        closed.remove(c);
    }

dependent after write(Connection c) returning:
    call(* Connection.write(..)) && target(c) {
        if(closed.contains(c))
            error("May not write to "+c+", as it is closed!");
    }

dependency{
    disconnect, write, reconnect;
    initial connected: disconnect -> connected,
        write -> connected,
        reconnect -> connected,
        disconnect -> disconnected;
    disconnect: disconnect -> disconnected,
        write -> error;
    final error: write -> error;
}
Semantics of Dependency State Machines

c.close();

if(...)

c.write(..);

c.reconnect();

c.write(..);
Semantics of Dependency State Machines

c.close();

if(...)

c.write(..); ✓

c.reconnect();

c.write(..);
Semantics of Dependency State Machines

j1: `c.close();`

if(...)

j2: `c.write(..);` ✓

j3: `c.reconnect();`

j4: `c.write(..);`
Semantics of Dependency State Machines

advice “close” must execute at \textit{j1} if

\begin{tikzpicture}
  \node (j1) at (0,0) {\texttt{j1}: \texttt{c.close()}; \quad \textit{if(\ldots)}};
  \node (j2) at (-3,-3) {\texttt{j2}: \texttt{c.write(\ldots);} \quad \checkmark}
  \node (j3) at (3,-3) {\texttt{j3}: \texttt{c.reconnect()});
  \node (j4) at (3,-6) {\texttt{j4}: \texttt{c.write(\ldots)};

  \draw[->,orange] (j1) -- (j2);
  \draw[->,orange] (j1) -- (j3);
  \draw[->,orange] (j3) -- (j4);
\end{tikzpicture}
Semantics of Dependency State Machines

```
j1: c.close();
if(...)
```

```
j2: c.write(..);
```

```
j3: c.reconnect();
```

```
j4: c.write(..);
```

advice “close” must execute at \( j_1 \) if there exists a path

Montag, 15. März 2010
Semantics of Dependency State Machines

advice "close" must execute at \text{j1} if there exists a path such that omitting "close" at \text{j1}

\text{j1} \colon \text{c.close();}
\text{j2} \colon \text{c.write(..);} \checkmark
\text{j3} \colon \text{c.reconnect();}
\text{j4} \colon \text{c.write(..);}

Montag, 15. März 2010
advice "close" must execute at j1 if there exists a path such that omitting "close" at j1 would change the joinpoints at which a DSM reaches an accepting state.
advice "close" must execute at j1 if there exists a path such that omitting "close" at j1 would change the joinpoints at which a DSM reaches an accepting state.

```
j1: c.close();
  if(...)
    j2: c.write(..); ✓
    j3: c.reconnect();
    j4: c.write(..);
```
Inverse case: match-preventing shadows

advice "reconnect" must execute at \textcolor{green}{j1} if there exists a path such that omitting "reconnect" at \textcolor{green}{j1} would change the joinpoints at which a DSM reaches an accepting state
Inverse case: match-preventing shadows

advice "reconnect" must execute at $j_1$ if there exists a path such that omitting "reconnect" at $j_1$ would change the joinpoints at which a DSM reaches an accepting state.
Inverse case: match-preventing shadows

advice “reconnect” must execute at \textcolor{green}{j1} if there exists a path such that omitting “reconnect” at \textcolor{green}{j1} would change the joinpoints at which a DSM reaches an accepting state.

\begin{itemize}
  \item \textcolor{green}{c.close()};
  \item \textcolor{green}{j1}: \textcolor{green}{c.reconnect()};
  \item \textcolor{green}{if(...)}
  \item \textcolor{orange}{j2}: \textcolor{orange}{c.write(..)};
  \item \textcolor{orange}{j3}: \textcolor{orange}{c.reconnect()};
  \item \textcolor{orange}{j4}: \textcolor{orange}{c.write(..)};
\end{itemize}
Variable bindings matter

c.close();

j1: c.reconnect();

if(...)

j2:

j3: c.reconnect();

j4: c.write(..);

Montag, 15. März 2010
Variable bindings matter

\[
\begin{align*}
  &c.\text{close}() \\
  \iff &j_1: c.\text{reconnect}() \\
  \iff &j_2: c2.\text{write}(..) \\
  \iff &j_3: c.\text{reconnect}() \\
  \iff &j_4: c.\text{write}(..)
\end{align*}
\]
Variable bindings matter

c.close();

\[ j_1: \text{c.reconnect}(); \]

if(…)

\[ j_3: \text{c.reconnect}(); \]

j4: c.write(…);
Variable bindings matter

c.close();

if(...)

\textcolor{red}{j_1}: \texttt{c.reconnect();}

\textcolor{red}{j_3}: \texttt{c.reconnect();}

\textcolor{orange}{j_4}: \texttt{c.write(..);}
Variable bindings matter

c.close();

\[ j_1: \text{c.reconnect()}; \]

if(...)

\[ j_3: \text{c.reconnect()}; \]

\[ j_4: \text{c.write(..)}; \]

Montag, 15. März 2010
Specifying and Exploiting Advice-Execution Ordering using Dependency State Machines
Specifying and Exploiting Advice-Execution Ordering using Dependency State Machines
The Clara Framework
The Clara Framework

Compile-time approximation of runtime analyses
The Clara Framework

dependent after(): call(...)
The Clara Framework

dependent after(): call(...)
The Clara Framework

“no write after close”

dependent after(): call(...)

Montag, 15. März 2010
The Clara Framework

“no write after close”

JavaMOP, abc, ...

dependent after(): call(…)

Montag, 15. März 2010
The Clara Framework

public class ClaraTest {
    public static void main(String args[]) {
        Connection cl = new Connection(args[0]);
        cl.close();
        cl.close();
        cl.close();
        cl.close();
        cl.close();
        cl.close();
    }
}

"no write after close"

JavaMOP, abc, ...

dependent after(): call(...)

compile & weave

public class ClaraTest {
    public static void main(String args[]) {
        Connection cl = new Connection(args[0]);
        cl.close();
        cl.reconnect();
        cl.close();
        cl.close();
        cl.write(args[1]);
        cl.close();
        cl.reconnect();
        cl.write(args[1]);
    }
}
The Clara Framework

public class ClaraTest {
    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.close();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }
}

public class NopShadowsAnalysis {
    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.close();
        c1.close();
        c1.write(args[1]);
        c1.close();
        c1.reconnect();
        c1.write(args[1]);
    }
}
The Clara Framework

```
public class ClaraTest {
    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.close();
        c1.close();
        c1.close();
        c1.close();
    }
}
```

```
public class ClaraTest {
    public static void main(String args[]) {
        Connection c1 = new Connection(args[0]);
        c1.close();
        c1.reconnect();
        c1.close();
        c1.close();
        c1.close();
        c1.close();
        c1.write(args[1]);
    }
}
```

JavaMOP, abc, ...

dependent after(): call(...)

"no write after close"

Quick Check

Orphan-Shadows Analysis

Nop-Shadows Analysis

compile & weave

Montag, 15. März 2010
The Clara Framework

"no write after close"

JavaMOP, abc, ...

dependent after(): call(...)

public class ClaraTest {
    public static void main(String args[]) {
        Connection cl = new Connection(args[0]);
        cl.close();
        cl.connect();
        cl.write(args[1]);
        cl.close();
        cl.disconnect();
        cl.close();
        cl.write(args[1]);
    }
}

Montag, 15. März 2010
Idea:

For every joinpoint shadow s:

— Identify states that are “equivalent” at s.
— If s may transition only between equivalent states then disable s.
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);

Montag, 15. März 2010
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.close();
c1.write(...);
c1.close();
c1.reconnect();
c1.write(...);
c1.close();
c1.close();
c1.reconnect();
c1.close();
\[\text{x}\]
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.write(...);
c1.close();
c1.reconnect();
c1.write(...);
c1.close();
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{}</td>
<td>{1,2}</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

```java
c1.close();
c1.reconnect();
c1.close();
c1.write(...);
c1.close();
c1.reconnect();
c1.write(...);
c1.close();
```
c1.close();
c1.reconnect();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();
c1.close();
c1.reconnect();

{0,1,2}
{0,1,2}
{1,2}
{2}
{2}
{}
c1.close();
c1.reconnect();
c1.close();
c1.close();
c1.write(..);
c1.close();
c1.reconnect();

c1.close();
c1.reconnect();

c1.close();

c1.close();

c1.reconnect();

{0,1,2}
{0,1,2}
{0,1,2}
{1,2}
{2}
{0,1,2}
{}
c1.close();
c1.write(..);

c1.write(..);
### Tested Properties

<table>
<thead>
<tr>
<th>ASyncContainsAll</th>
<th>FailSafeIterMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASyncIterC</td>
<td>HasNextElem</td>
</tr>
<tr>
<td>ASyncIterM</td>
<td>HasNext</td>
</tr>
<tr>
<td>FailSafeEnum</td>
<td>LeakingSync</td>
</tr>
<tr>
<td>FailSafeEnumHT</td>
<td>Reader</td>
</tr>
<tr>
<td>FailSafeIter</td>
<td>Writer</td>
</tr>
</tbody>
</table>

Montag, 15. März 2010
## Benchmark programs

<table>
<thead>
<tr>
<th>antlr</th>
<th>jython</th>
</tr>
</thead>
<tbody>
<tr>
<td>bloat</td>
<td>luindex</td>
</tr>
<tr>
<td>chart</td>
<td>lusearch</td>
</tr>
<tr>
<td>fop</td>
<td>pmd</td>
</tr>
<tr>
<td>hsqldb</td>
<td>xalan</td>
</tr>
</tbody>
</table>

(whole DaCapo benchmark suite, except eclipse)
Overall success

<table>
<thead>
<tr>
<th></th>
<th>antlr</th>
<th>bloat</th>
<th>chart</th>
<th>fop</th>
<th>hsqldb</th>
<th>jython</th>
<th>luindex</th>
<th>lusearch</th>
<th>pmd</th>
<th>xalan</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASyncContainsAll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASyncIterC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASyncIterM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeEnum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeEnumHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeIter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeIterMap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HasNextElem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HasNext</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeakingSync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Montag, 15. März 2010
Related Work

Static analysis of Aspects:

• Dependent Advice (Bodden, Chen & Rosu, AOSD ’09)

Static analysis of Runtime Monitors:

• Tracematches (Naeem & Lhotak, OOPSLA ’08)
• Tracematches (Bodden et al., ECOOP ’07 and FSE ’08)
• General aspects with DSMs (Bodden, ICSE ’10)
Related Work

Aspect-internal optimizations:

- Indexing and Leak Elimination for Tracematches (Avgustinov, Tibble & de Moor, OOPSLA ‘07)
- Formalism-independent Indexing (Chen & Rosu, OOPSLA ‘07)
- PQL (Goldsmith, O’Callahan & Eiken, OOPSLA ‘05)
Related Work

Model Checking for aspects:

- Goldman & Katz, TACAS ’07
Related Work

Generating history-based aspects:

- Association Aspects (Sakurai et al., AOSD ’04)
- Tracematches (Avgustinov et al., AOSD ’05)
- MOP (Chen & Rosu, OOPSLA ’07)
- Relational Aspects (Bodden et al., AOSD ’08)
- M2Aspects (Krüger et al., SCESM ’06)
- S2A (Maoz & Harel, FSE ’06)
- J-LO (Bodden et al., RV ’07)
- Java-STAIRS Aspects (Oldevik, Haugen, AOSD ’09)

... what will we see this year?
Conclusion
Conclusion

dependency{
    disconnect, write, reconnect;
    initial  connected: disconnect -> connected,
        write -> connected,
        reconnect -> connected,
        disconnect -> disconnected;
    disconnect: disconnect -> disconnected,
        write -> error;
    final     error: write -> error;
}

<table>
<thead>
<tr>
<th>Class</th>
<th>antlr</th>
<th>bloat</th>
<th>chart</th>
<th>fop</th>
<th>honda</th>
<th>jython</th>
<th>hindex</th>
<th>jresarch</th>
<th>pand</th>
<th>xalan</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASyncContainsAll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASyncIterC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASyncIterM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeEnum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeEnumHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeIter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FailSafeIterMap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HasNextElem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HasNext</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeakingSync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rings represent the aspect's runtime overhead after optimization. Shadows are relevant. Gray slices represent shadows that are irrelevant. Black slices represent shadows that we failed to confirm to be relevant, through manual inspection. The overhead is depicted as a chart with annotations for each class and property. The chart shows the fraction of relevant shadows as a percentage of the total shadows for each property.
Conclusion

dependency{
    disconnect, write, reconnect;
    initial connected: disconnect -> connected,
        write -> connected,
        reconnect -> connected,
        disconnect -> disconnected;
    disconnect: disconnect -> disconnected,
        write -> error;
    final error: write -> error;
}

Table 1: Relevant typestate properties and their names

Table 2: Irrelevant vs. potentially relevant shadows. White: 15%, dotted: no overhead. OOME = OutOfMemoryException during the advice dispatch. Solid: overhead ≥ 1994.

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
Type & Antlr & Bleat & Chart & Fop & Houghl & Jython & Hindex & BaseSearch & Pmd & Xalan \\
\hline
ASynContainsAll & & & & & & & & & & \\
ASynFilterC & & & & & & & & & & \\
ASynFilterM & & & & & & & & & & \\
FailSafeEnum & & & & & & & & & & \\
FailSafeEnumHT & & & & & & & & & & \\
FailSafeIter & & & & & & & & & & \\
FailSafeIterMap & & & & & & & & & & \\
HasNext & & & & & & & & & & \\
HasNextElem & & & & & & & & & & \\
LeakingSync & & & & & & & & & & \\
Reader & & & & & & & & & & \\
Writer & & & & & & & & & & \\
\hline
\end{tabular}
Conclusion

dependency{
disconnect, write, reconnect;
initial connected: disconnect -> connected,
write -> connected,
reconnect -> connected,
disconnect -> disconnected;
disconnect: disconnect -> disconnected,
write -> error;
final error: write -> error;
}

www.bodden.de/clara/
Conclusion

dependency{
  disconnect, write, reconnect;
  initial  connected: disconnect -> connected,
           write -> connected,
           reconnect -> connected,
           disconnect -> disconnected;
  disconnect: disconnect -> disconnected,
           write -> error;
  final     error: write -> error;
}

What else can we use DSMs for?

www.bodden.de/clara/