Lattice Based Modularization of Static Analyses

Michael Eichberg, Florian Kübler, Dominik Helm, Michael Reif, Guido Salvaneschi and Mira Mezini
The Initial Challenge (aka Research Question)
What do we need to prove that instances of `java.lang.String` are immutable.

```java
public final class String extends Traversable<Byte> {
    private final byte[] buf;
    private int hash;

    public String(byte[] buf) {
        this.buf = Arrays.copyOf(buf, buf.length);
    }

    String(byte[] buf, boolean cloned) {
        assert (cloned);
        this.buf = buf;
    }

    ... 

    @Override public int hashCode() {
        if (this.hash == 0) {
            int hash = 0;
            for (byte v : buf) {
                hash += v;
            }
            this.hash = hash;
        }
        return this.hash;
    }
}
```

We need to model the effect of (selected) native methods.

We need points-to/escape information.

We need to understand lazy initialization patterns.
What do we need to prove that instances of java.lang.String are immutable.

```java
public final class String extends Traversable<Byte> {
    private final byte[] buf;
    private int hash;
    public String(byte[] buf) {
        this.buf = Arrays.copyOf(buf, buf.length);
    }
    String(byte[] buf, boolean cloned) {
        assert (cloned);
        this.buf = buf;
    }
    ...
    @Override public int hashCode() {
        if (this.hash == 0) {
            int hash = 0;
            for (byte v : buf) { hash += v; }
            this.hash = hash;
        }
        return this.hash;
    }
}
```

We need to model the effect of (selected) native methods.

We need points-to/escape information.

We need to understand lazy initialization patterns.

Too much for one analysis.

Req. 1: the “overall” analysis has to be modularized

Req. 2: we have to facilitate incremental development
(we can’t develop all analyses in one step; we need to be able to test parts of it!)

Req. 3: only compute required information (efficiently)

We need a framework for the efficient execution of independently developed, but mutually dependent fix-point computations.

return this.hash;

}   }
Based on the theoretical foundations of **fix-point computations on lattices**, we developed a Scala based framework to develop strictly modularized static analyses. We use lattices as the inter-analyses interfaces.
Lattices related to independent properties (e.g. mutability, purity, return value freshness, thrown exceptions...)

Independent analyses

The lattices represent the interface!
Analyses are basically just a simple Scala function.

def analyze(e: Entity): ComputationResult

type ComputationResult =
    (Entity, Property, Dependees, OnUpdateContinuation)

type Dependees = Traversable[(Entity, PropertyKind)]

type OnUpdateContinuation =
    (Entity, Property, State) => ComputationResult
Example: Computing the purity for “getA”.

class X {
    private int a;
    public X(int a) { this.a = a; }
    public int getA() { return this.a; }
}

1. analyze(getA) =>
   p=Pure, dependees=(X.a|FieldMutability), c=<cont.>
2. (computation of the field mutability)
3. c(X.a,EffectivelyFinal,Final) =>
   p=Pure, dependees={}, c=N/A
Handling cyclic computations.

def foo() {bar()}
def bar() {foo()}

Scenario 1

cycle resolution after step 3
Does it work?
(Implicit) dependencies between the 10 basic analyses + 4 supporting analyses developed to evaluate the framework.
The precision and recall effects of an analysis using different support analyses can easily be evaluated.

<table>
<thead>
<tr>
<th>Analysis configuration</th>
<th>P2/E1/F0/L0/M1/I0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>52 628 (20.78%)</td>
</tr>
<tr>
<td>Side Effect Free</td>
<td>32 951 (13.01%)</td>
</tr>
<tr>
<td>Contextually Pure/Side Effect Free</td>
<td>11 614 (4.59%)</td>
</tr>
<tr>
<td>Impure</td>
<td>156 089 (61.63%)</td>
</tr>
<tr>
<td>Relative execution time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100%</td>
</tr>
</tbody>
</table>
The precision and recall effects of an analysis using different support analyses can easily be evaluated.

<table>
<thead>
<tr>
<th>Analysis configuration</th>
<th>P2/E0/F0/L0/M1/I0</th>
<th>P2/E1/F0/L0/M1/I0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>52 628 (20.78%)</td>
<td>52 602 (20.77%)</td>
</tr>
<tr>
<td>Side Effect Free</td>
<td>32 951 (13.01%)</td>
<td>32 964 (13.01%)</td>
</tr>
<tr>
<td>Contextually Pure/Side Effect Free</td>
<td>11 614 (4.59%)</td>
<td>11 459 (4.52%)</td>
</tr>
<tr>
<td>Impure</td>
<td>156 089 (61.63%)</td>
<td>156 257 (61.69%)</td>
</tr>
</tbody>
</table>

Relative execution time

100%

100%
The precision and recall effects of an analysis using different support analyses can easily be evaluated.

<table>
<thead>
<tr>
<th>Analysis configuration</th>
<th>P2/E1/F0/L0/M1/I0</th>
<th>P2/E0/F0/L0/M1/I0</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>52 628 (20.78%)</td>
<td>52 602 (20.77%)</td>
<td>49 849 (19.68%)</td>
</tr>
<tr>
<td>Side Effect Free</td>
<td>32 951 (13.01%)</td>
<td>32 964 (13.01%)</td>
<td>35 654 (14.08%)</td>
</tr>
<tr>
<td>Contextually Pure/Side Effect Free</td>
<td>11 614 (4.59%)</td>
<td>11 459 (4.52%)</td>
<td>11 173 (4.41%)</td>
</tr>
<tr>
<td>Impure</td>
<td>156 089 (61.63%)</td>
<td>156 257 (61.69%)</td>
<td>156 606 (61.83%)</td>
</tr>
<tr>
<td>Relative execution time</td>
<td>100%</td>
<td>100%</td>
<td>75%</td>
</tr>
</tbody>
</table>
The precision and recall effects of an analysis using different support analyses can easily be evaluated.

<table>
<thead>
<tr>
<th>Analysis configuration</th>
<th>P2/E1/F0/L0/M1/I0</th>
<th>P2/E0/F0/L0/M1/I0</th>
<th>P2</th>
<th>P0/M0/I0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>52 628 (20.78%)</td>
<td>52 602 (20.77%)</td>
<td>49 849 (19.68%)</td>
<td>11 645 (4.60%)</td>
</tr>
<tr>
<td>Side Effect Free</td>
<td>32 951 (13.01%)</td>
<td>32 964 (13.01%)</td>
<td>35 654 (14.08%)</td>
<td>-</td>
</tr>
<tr>
<td>Contextually Pure/Side Effect Free</td>
<td>11 614 (4.59%)</td>
<td>11 459 (4.52%)</td>
<td>11 173 (4.41%)</td>
<td>-</td>
</tr>
<tr>
<td>Impure</td>
<td>156 089 (61.63%)</td>
<td>156 257 (61.69%)</td>
<td>156 606 (61.83%)</td>
<td>24 1456 (95.40%)</td>
</tr>
<tr>
<td>Relative execution time</td>
<td>100%</td>
<td>100%</td>
<td>75%</td>
<td>15%</td>
</tr>
</tbody>
</table>
A comparison with related work demonstrates the effectiveness of the proposed approach.

---

**Program**

**ReIm**

*Side Effect Free methods*

#Analyzed methods

**OPAL**

*Pure methods*

*Side Effect Free (incl. Pure) methods*

*Contextually Pure/SEF methods*

#Analyzed methods
A comparison with related work demonstrates the effectiveness of the proposed approach.

<table>
<thead>
<tr>
<th>Program</th>
<th>Batik</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ReIm</strong></td>
<td></td>
</tr>
<tr>
<td>Side Effect Free methods</td>
<td>6 072 (37.88%)</td>
</tr>
<tr>
<td>#Analyzed methods</td>
<td>16 029</td>
</tr>
<tr>
<td><strong>OPAL</strong></td>
<td></td>
</tr>
<tr>
<td>Pure methods</td>
<td>4 009 (25.20%)</td>
</tr>
<tr>
<td>Side Effect Free (incl. Pure) methods</td>
<td>6 780 (42.61%)</td>
</tr>
<tr>
<td>Contextually Pure/SEF methods</td>
<td>987 (6.20%)</td>
</tr>
<tr>
<td>#Analyzed methods</td>
<td>15 911</td>
</tr>
</tbody>
</table>
A comparison with related work demonstrates the effectiveness of the proposed approach.

<table>
<thead>
<tr>
<th>Program</th>
<th>Batik</th>
<th>Xalan</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReIm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Side Effect Free methods</td>
<td>6 072 (37.88%)</td>
<td>3 942 (37.95%)</td>
</tr>
<tr>
<td>#Analyzed methods</td>
<td>16 029</td>
<td>10 386</td>
</tr>
<tr>
<td>OPAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pure methods</td>
<td>4 009 (25.20%)</td>
<td>2 492 (23.15%)</td>
</tr>
<tr>
<td>Side Effect Free (incl. Pure) methods</td>
<td>6 780 (42.61%)</td>
<td>4 390 (40.79%)</td>
</tr>
<tr>
<td>Contextually Pure/SEF methods</td>
<td>987 (6.20%)</td>
<td>748 (6.95%)</td>
</tr>
<tr>
<td>#Analyzed methods</td>
<td>15 911</td>
<td>10 763</td>
</tr>
</tbody>
</table>
The framework enables...

- fine-grained modularization
- assessing the contribution of supporting analyses on a primary analysis
- inherent parallelization
- on-demand computations
- laziness by refining upper and lower bounds
Lattice Based Modularization of Static Analyses

Michael Eichberg, Florian Kübler, Dominik Helm, Michael Reif, Guido Salvaneschi and Mira Mezini

www.opal-project.de