Lattice Based Modularization of Static Analyses

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

Software Technology Group

TECHNISCHE UNIVERSITÄT DARMSTADT

The Initial Challenge (aka Research Question)

What do we need to *prove* that instances of java.lang.String are immutable.

```
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) { ha
            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.

```
public final class String extends Traversable<Byte
```

We need to model the

Too much for **one** analysis.

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

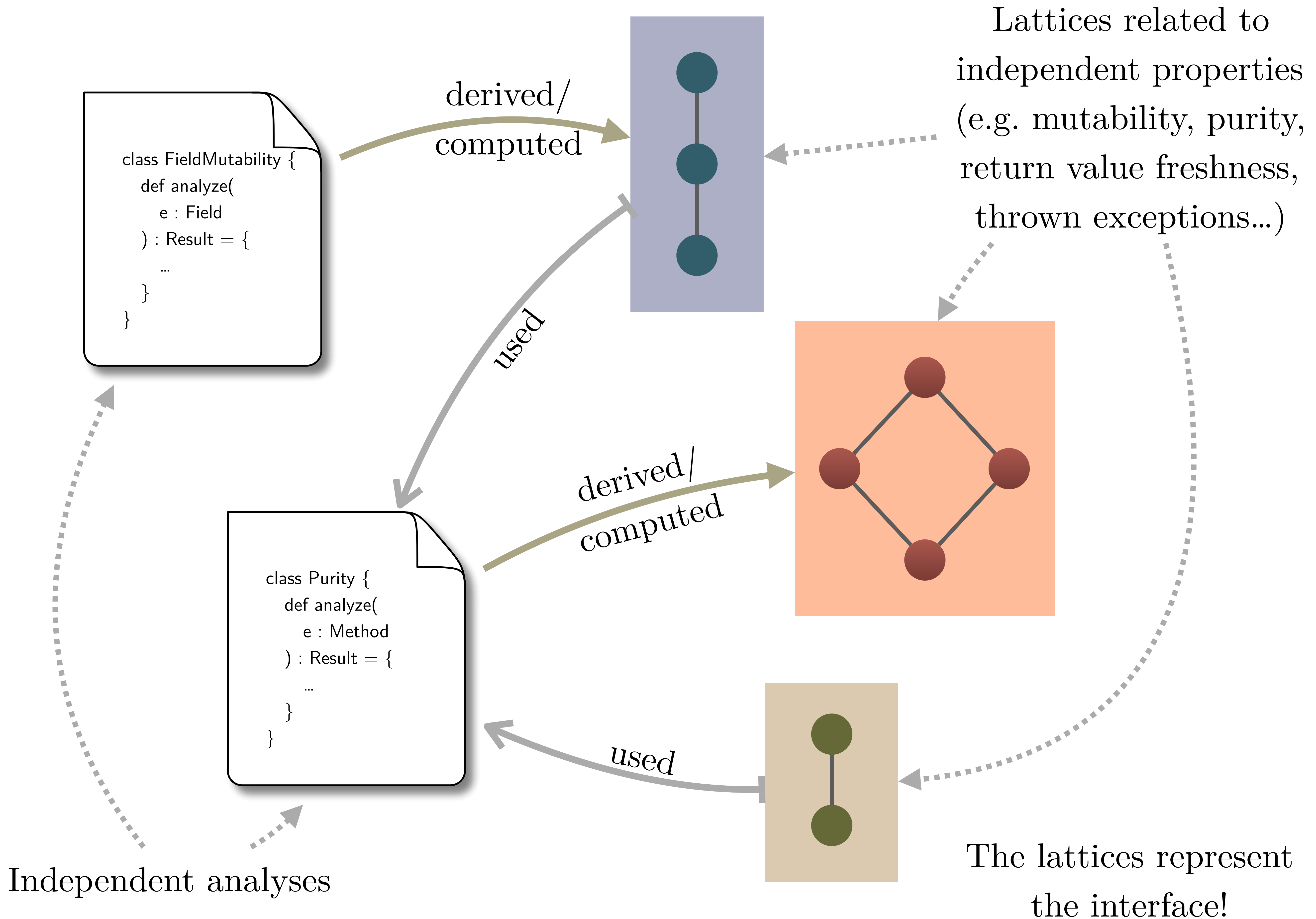
Req. 2: we have to facilitate incremental development
(we can’t develop all analyses in one step; we need to be able to test parts of it!)

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...)
derived/ computed
class FieldMutability {
def analyze(
e : Field
) : Result = {
...
}
}
used
derived/ computed
class Purity {
def analyze(
e : Method
) : Result = {
...
}
}
used
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; }
}
```

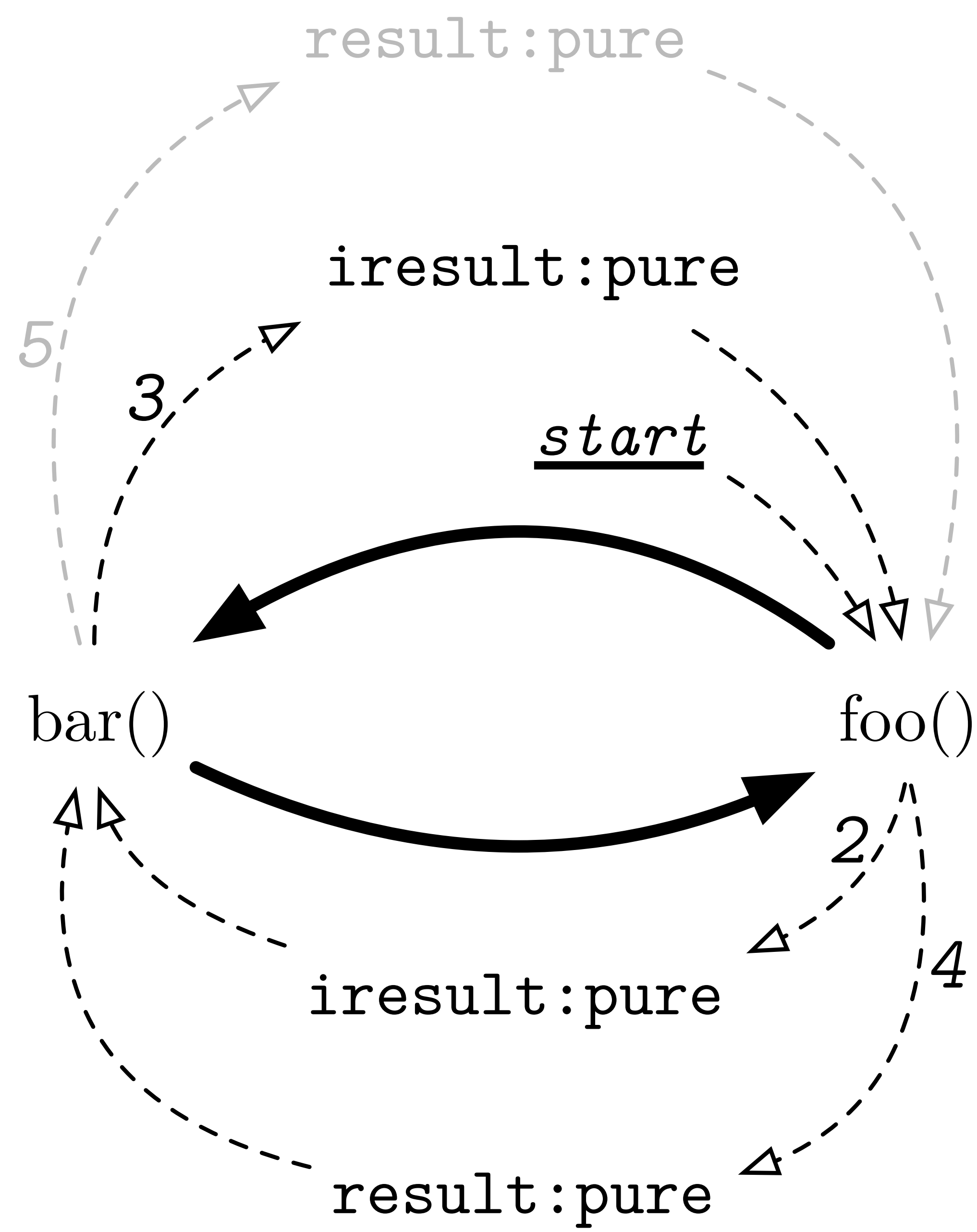
Java Code

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()}

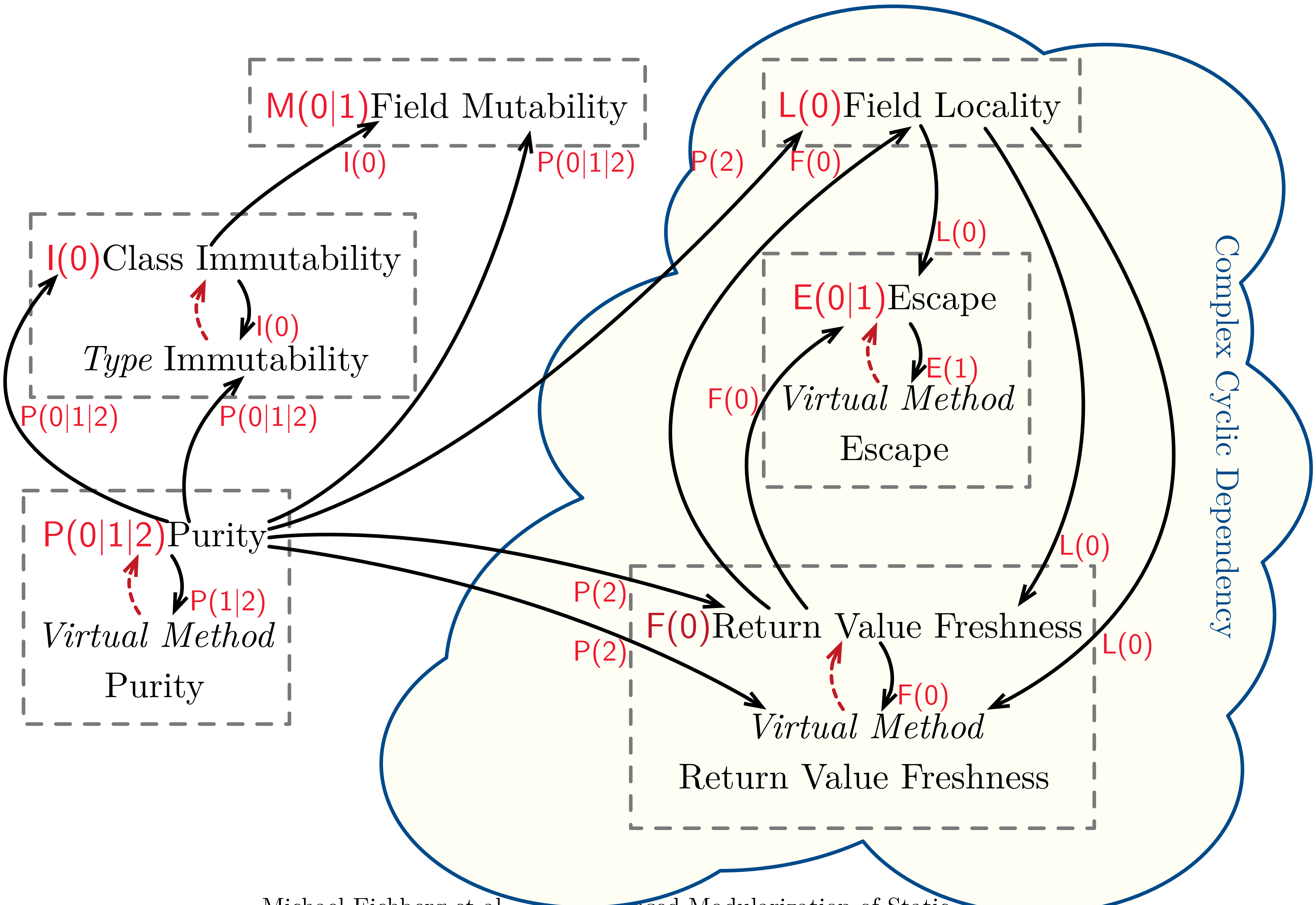
def bar() {foo()}



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.

	Most Precise Purity Analysis and Best Supporting Analyses
Analysis configuration	P2/**E1**/F0/L0/M1/I0
Pure	52 628 (20.78%)
Side Effect Free	32 951 (13.01%)
Contextually Pure/Side Effect Free	11 614 (4.59%)
Impure	156 089 (61.63%)
Relative execution time	100%

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

Analysis configuration	Most Precise Purity Analysis and Best Supporting Analyses P2/**E1**/F0/L0/M1/I0	Most Precise Purity Analysis and Most Basic Supporting Analyses P2/**E0**/F0/L0/M1/I0
Pure	52 628 (20.78%)	52 602 (20.77%)
Side Effect Free	32 951 (13.01%)	32 964 (13.01%)
Contextually Pure/Side Effect Free	11 614 (4.59%)	11 459 (4.52%)
Impure	156 089 (61.63%)	156 257 (61.69%)
Relative execution time	100%	100%

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

	Most Precise Purity Analysis and Best Supporting Analyses	Most Precise Purity Analysis and Most Basic Supporting Analyses	Most Precise Purity Analysis and No Supporting Analyses (Using Fallbacks ~Bottom)
Analysis configuration	P2/**E1**/F0/L0/M1/I0	P2/**E0**/F0/L0/M1/I0	P2
Pure	52 628 (20.78%)	52 602 (20.77%)	49 849 (19.68%)
Side Effect Free	32 951 (13.01%)	32 964 (13.01%)	35 654 (14.08%)
Contextually Pure/Side Effect Free	11 614 (4.59%)	11 459 (4.52%)	11 173 (4.41%)
Impure	156 089 (61.63%)	156 257 (61.69%)	156 606 (61.83%)
Relative execution time	100%	100%	75%

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

	Most Precise Purity Analysis and Best Supporting Analyses	Most Precise Purity Analysis and Most Basic Supporting Analyses	Most Precise Purity Analysis and No Supporting Analyses (Using Fallbacks ~Bottom)	Most Basic Purity Analysis
Analysis configuration	P2/**E1**/F0/L0/M1/I0	P2/**E0**/F0/L0/M1/I0	P2	P0/M0/I0
Pure	52 628 (20.78%)	52 602 (20.77%)	49 849 (19.68%)	11645 (4.60%)
Side Effect Free	32 951 (13.01%)	32 964 (13.01%)	35 654 (14.08%)	– –
Contextually Pure/Side Effect Free	11 614 (4.59%)	11 459 (4.52%)	11 173 (4.41%)	– –
Impure	156 089 (61.63%)	156 257 (61.69%)	156 606 (61.83%)	241456 (95.40%)
Relative execution time	100%	100%	75%	15%

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.

Program	**Batik**
ReIm	
Side Effect Free methods	6 072 (37.88%)
#Analyzed methods	16 029
OPAL	
Pure methods	4 009 (25.20%)
Side Effect Free (incl. Pure) methods	6 780 (42.61%)
Contextually Pure/SEF methods	987 (6.20%)
#Analyzed methods	15 911

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

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

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