A Unified Lattice Model and Framework for Purity Analyses

Dominik Helm   Florian Kübler   Michael Eichberg
Michael Reif   Mira Mezini

TECHNISCHE UNIVERSITÄT DARMSTADT

SOFTWARE TECHNOLOGY GROUP

```
static double radius, area;

void computeArea1(){
    area = radius * radius * Math.PI;
}

double computeArea2(){
    return radius * radius * Math.PI;
}
```

```
static double radius, area;

void computeArea1(){
    area = radius * radius * Math.PI;
}

double computeArea2(){
    return radius * radius * Math.PI;
}
```

- Program comprehension
- Bug/code smell detection
- Optimizations

```java
static double radius;

double computeArea2(){
    return radius * radius * Math.PI;
}

double computeArea3(double _radius){
    return _radius * _radius * Math.PI;
}
```

- Program comprehension
- Optimizations
- Formal specification/verification

```
static double radius;

double computeArea2(){
    return radius * radius * Math.PI;
}

double computeArea3(double _radius){
    return _radius * _radius * Math.PI;
}
```

Unifying definitions:

Side-effect freeness (SEF):

- execution can not be observed by other methods

Purity:

- additionally deterministic
- i.e. can not observe the execution of other methods

Unifying definitions:

Side-effect freeness (SEF):

- execution can not be observed by other methods

Purity:

- additionally deterministic
- i.e. can not observe the execution of other methods

Sufficient definitions, but valuable extensions exist

External purity [BF09]:

- only (observable) side-effect is modification of receiver object
- field setters, initialization methods, . . .
- respects abstraction boundaries
- allows finding confined side-effects

[BF09] Benton, W.C. and Fischer, C.N.
Mostly-functional behavior in Java programs, VMCAI'09

External purity [BF09]:

- only (observable) side-effect is modification of receiver object
- field setters, initialization methods, ...
- respects abstraction boundaries
- allows finding confined side-effects

Generalization to **contextual purity**:

- side-effects restricted to parameters
- e.g. `System.arraycopy`
- allows finding more confined side-effects

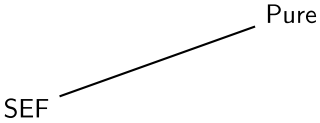
[BF09] Benton, W.C. and Fischer, C.N.
Mostly-functional behavior in Java programs, VMCAI'09

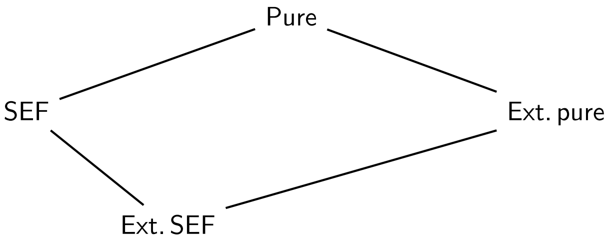
Domain-specific purity:

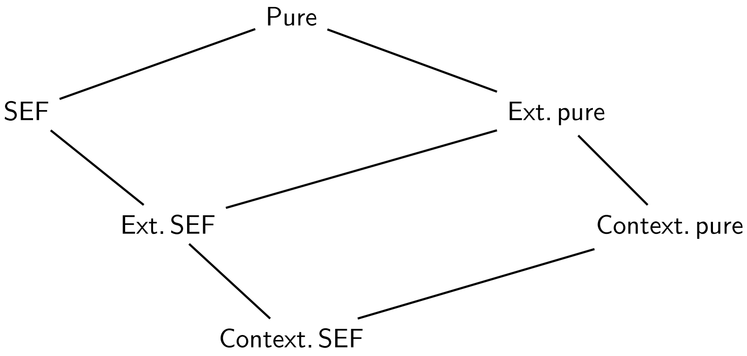
- restricted forms of simpurity
- relevant for some but not all domains
- e.g. logging [SCD16], exceptions

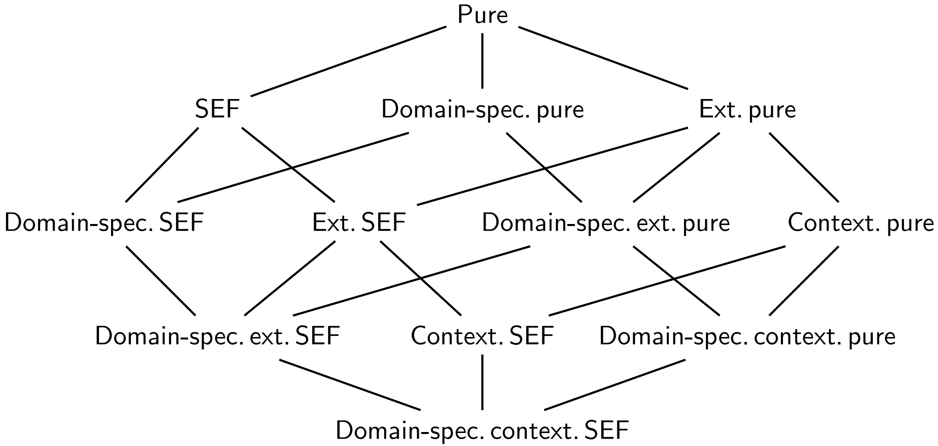
```
int divide(int a, int b) {
    Log.log("Performing␣division");
    return a/b;
}
```

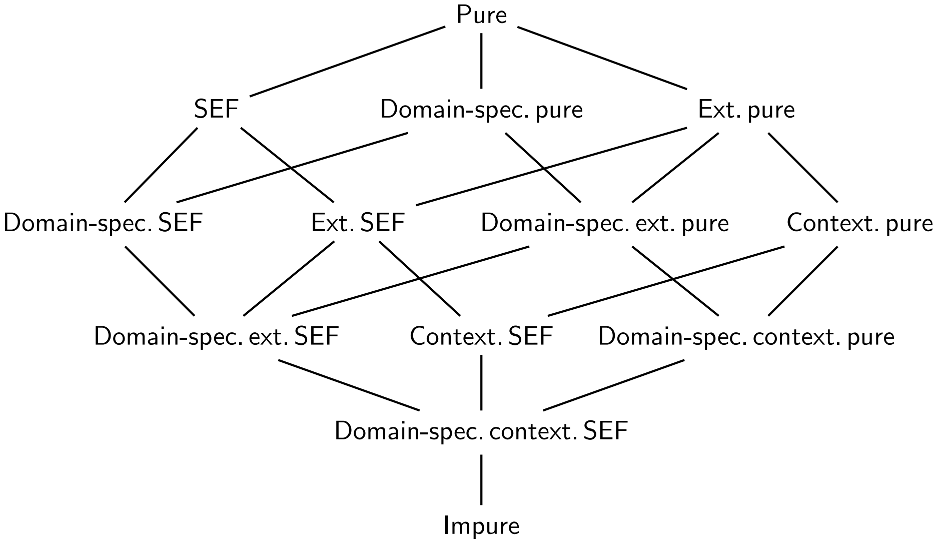
[SCD16] Stewart, A., Cardell-Oliver, R., Davies, R.
Fine-grained classification of side-effect free methods in real-world Java code and applications to software security, ACSW'16


Pure
SEF


Pure
SEF
Ext. pure
Ext. SEF


Pure
SEF
Ext. pure
Ext. SEF
Context. pure
Context. SEF


Pure
SEF
Domain-spec. pure
Ext. pure
Domain-spec. SEF
Ext. SEF
Domain-spec. ext. pure
Context. pure
Domain-spec. ext. SEF
Context. SEF
Domain-spec. context. pure
Domain-spec. context. SEF


Pure
SEF
Domain-spec. pure
Ext. pure
Domain-spec. SEF
Ext. SEF
Domain-spec. ext. pure
Context. pure
Domain-spec. ext. SEF
Context. SEF
Domain-spec. context. pure
Domain-spec. context. SEF
Impure

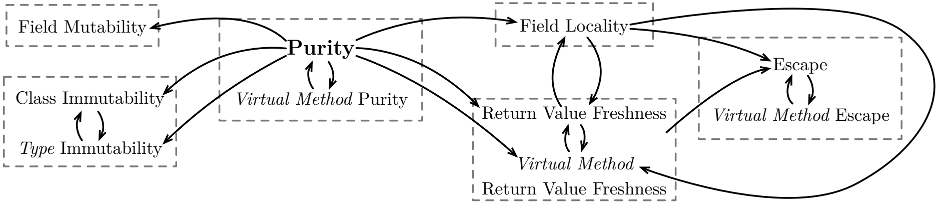
Implemented a purity analysis in OPAL[1] that

- analyses each method assuming **purity**
- scans the intermediate representation,
  enriched with abstract interpretation results
  (refined types, def-use chains, . . .)
- searching for counterexamples
- and collecting dependencies

[1] http://www.opal-project.de

Implemented a purity analysis in OPAL[1] that

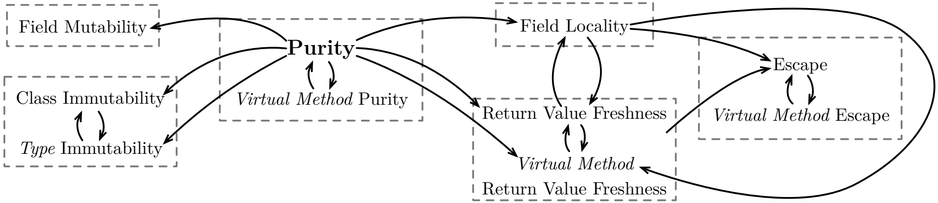
- analyses each method assuming **purity**
- scans the intermediate representation, enriched with abstract interpretation results (refined types, def-use chains, ...)
- searching for counterexamples
- and collecting dependencies



[1] http://www.opal-project.de

Implemented a purity analysis in OPAL[1] that

- analyses each method assuming **purity**
- scans the intermediate representation, enriched with abstract interpretation results (refined types, def-use chains, ...)
- searching for counterexamples
- and collecting dependencies
- which are resolved automatically



[1]http://www.opal-project.de

Corpus	**XCorpus**		**JDK8**	
Methods	469 727		253 282	
Pure	73 701	(15.69%)	44 378	(17.52%)
Domain-specific pure	9 628	(2.05%)	8 250	(3.26%)
Side-Effect Free	45 268	(9.64%)	18 234	(7.20%)
Domain-spec. SEF	22 056	(4.70%)	14 717	(5.81%)
Externally pure	19 467	(4.14%)	4 229	(1.67%)
Externally SEF	2 380	(0.51%)	3 414	(1.35%)
Domain-spec. ext. pure	639	(0.14%)	354	(0.14%)
Domain-spec. ext. SEF	2 467	(0.53%)	1 738	(0.69%)
Contextually pure	4	(0.00%)	7	(0.00%)
Contextually SEF	7	(0.00%)	48	(0.02%)
Dom.-spec. cont. pure	522	(0.11%)	547	(0.22%)
Dom.-spec. cont. SEF	1 523	(0.32%)	1 277	(0.50%)
Impure	292 065	(62.18%)	156 089	(61.63%)

Program	**Batik**	**Xalan**
ReIm [HM12]		
Analyzed methods	16 029	10 386
At least Side-Effect Free	6 072 (37.88%)	3 942 (37.95%)
Execution time	103s	140s
OPIUM		
Analyzed methods	15 911	10 763
At least Side-Effect Free	6 780 (42.61%)	4 390 (40.79%)
Pure	4 009 (25.20%)	2 492 (23.15%)
Ext./Context. Pure/SEF	+987 (6.20%)	+748 (6.95%)
Execution time	197 s	187 s

[HM12] Huang, W. and Milanova, A.
ReImInfer: Method purity inference for Java FSE'12

Program	**Batik**	**Xalan**
ReIm [HM12]		
Analyzed methods	16 029	10 386
At least Side-Effect Free	6 072 (37.88%)	3 942 (37.95%)
Execution time	103s	140s
OPIUM (our tool)		
Analyzed methods	15 911	10 763
At least Side-Effect Free	6 780 (42.61%)	4 390 (40.79%)
Pure	4 009 (25.20%)	2 492 (23.15%)
Ext./Context. Pure/SEF	+987 (6.20%)	+748 (6.95%)
Execution time	103 s ~~197 s~~	104 s ~~187 s~~

[HM12] Huang, W. and Milanova, A.
ReImInfer: Method purity inference for Java FSE'12

Recap:

- Definitions for side-effect freeness and purity
- External and contextual purity
- Domain-specific purity
- Unified lattice model
- Fast and precise analysis

Try it yourself: http://www.opal-project.de/OPIUM.html

Current and future work:

- Modularize and improve more analyses
  - Call graphs
  - Immutability of Classes/Fields
  - Points-To/Alias analysis
  - ...
- New concepts for IFDS/IDE analysis
- Parallelizing computations in FPCF