Lund University
Computer Science Department

A Precise Framework for Source-Level Control-Flow Analysis

21st IEEE International Working Conference on Source Code Analysis and Manipulation

Idriss Riouak, Christoph Reichenbach, Görel Hedin and Niklas Fors

September 28, 2021

Introduction

Data-flow analysis plays an important role in software development, and helps developers to detect subtle bugs

Introduction

Data-flow analysis plays an important role in software development, and helps developers to detect subtle bugs

Source-level dataflow analysis ... why ?

- Easier integration with IDE
- Reports are directly linked to the source code

Introduction

Data-flow analysis plays an important role in software development, and helps developers to detect subtle bugs

Source-level dataflow analysis ... why ?

- Easier integration with IDE
- Reports are directly linked to the source code

The main challenges were:

- Large engineering effort for each source language
- The syntax doesn't always reflect the program's semantics

Our approach

We build the CFGs as extension of the AST using Reference Attribute Grammars (RAGs)

- **Declarative** specification
- Handle **implicit control flow**
- Overcome the limitations of an earlier framework

Research questions

- How can we reduce the engineering effort ?
- How can we fill the gap between *syntax and semantics* ?
- Is our new approach competitive performance-wise?

Intraprocedural RAG-based CFGs

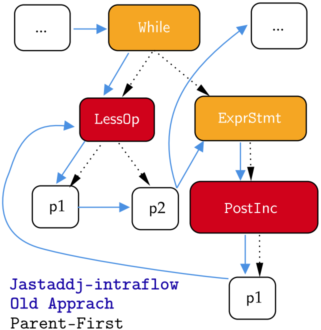
We removed the limitations of the previous approach

Code

```
while(p1<p2){
  p1++;
}
```

Legend

🟧🟥▨□ AST node	🟥 Misplaced	→ succ
▨ Not in CFG	🟧 Redundant	⋯► Child relation



Jastaddj-intraflow
Old Apprach
Parent-First

Intraprocedural RAG-based CFGs

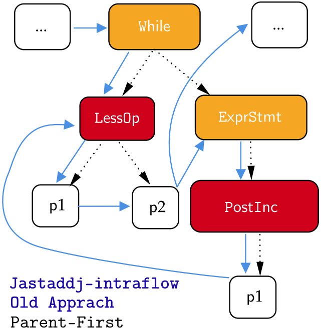
We removed the limitations of the previous approach

Code

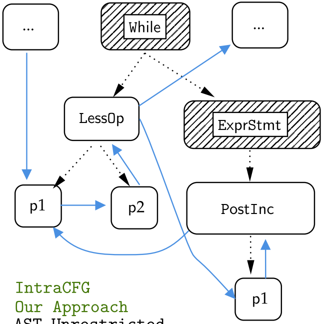
```
while(p1<p2){
  p1++;
}
```

Legend

🟧🟥▨□	AST node	🟥	Misplaced	→	succ
▨	Not in CFG	🟧	Redundant	⋯►	Child relation

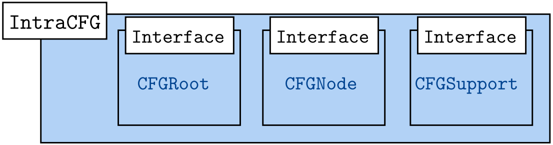


Jastaddj-intraflow
Old Apprach
Parent-First

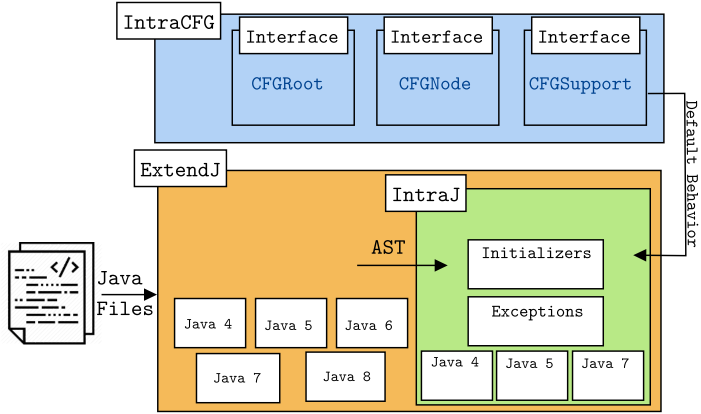


IntraCFG
Our Approach
AST-Unrestricted

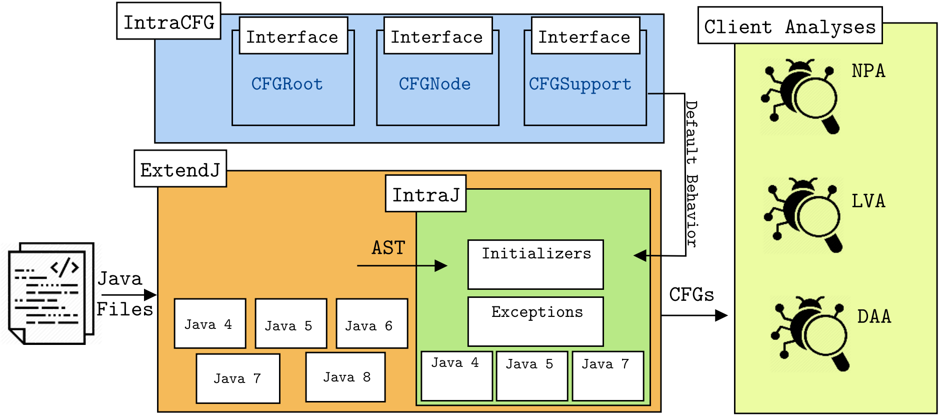
Modular architecture



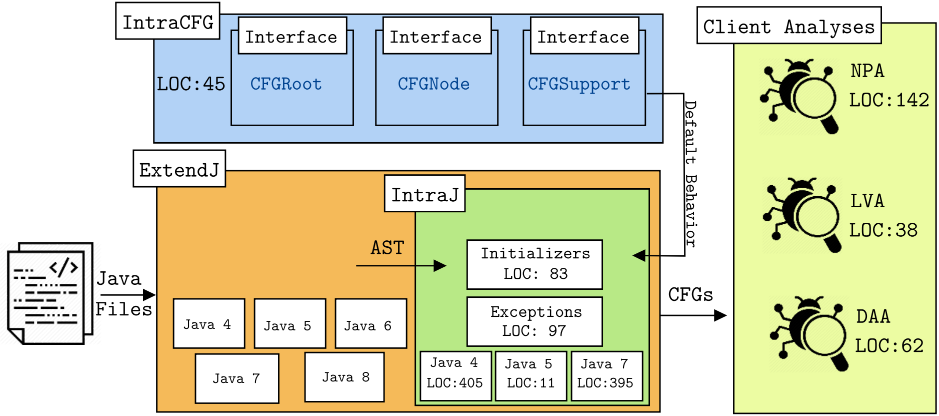
Modular architecture



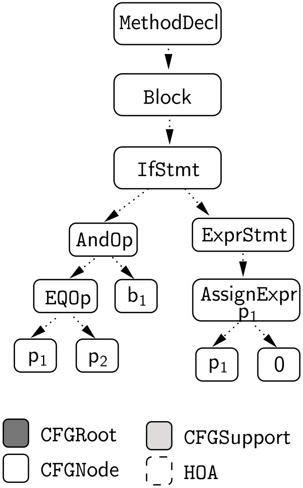
Modular architecture



Modular architecture

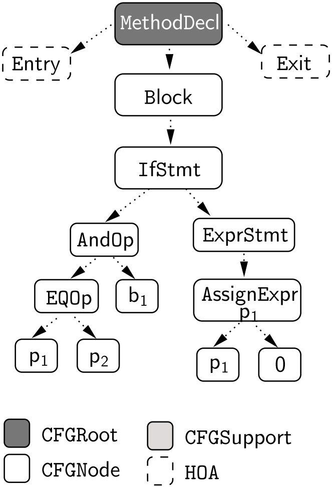


Framework overview



```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

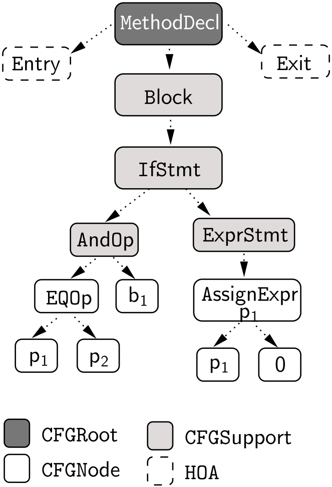
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

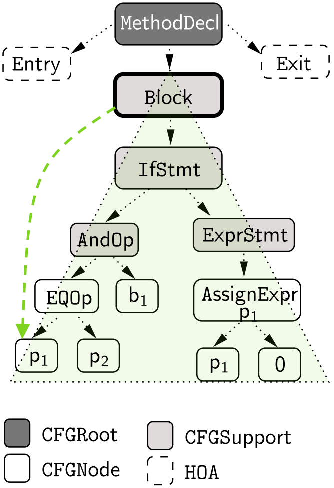
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

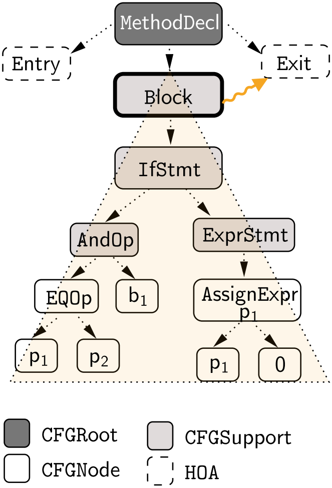
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:
  - firstNodes

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

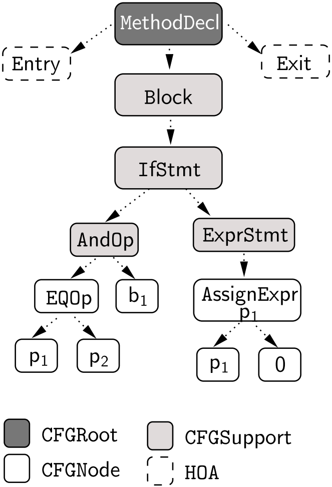
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:
  - firstNodes
  - nextNodes

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

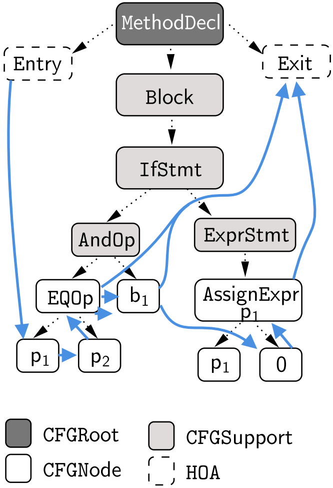
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:
  - firstNodes
  - nextNodes
- All the CFGNode are CFGSupport

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

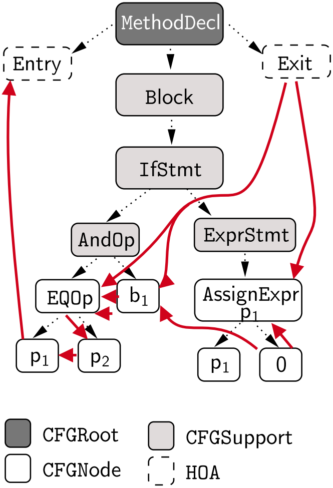
Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:
  - firstNodes
  - nextNodes
- All the CFGNode are CFGSupport
- Used firstNodes and nextNodes to compute the succ attribute

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

Framework overview



- CFGRoot extends the AST with two HOAs: Entry and Exit
- CFGSupport defines:
  - firstNodes
  - nextNodes
- All the CFGNode are CFGSupport
- Used firstNodes and nextNodes to compute the succ attribute
- The pred is computed as the inverse of succ

```
foo(int p1, int p2, Boolean b1){
  if(p1==p2 && b1)
    p1 = 0;
}
```

Challenges

- We used HOAs to extend the AST with new subtrees

Challenges

- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**

Challenges

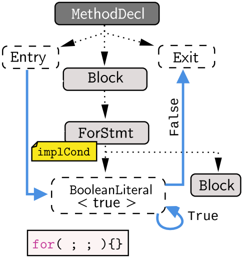
- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers

Challenges

- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers
  - Exception-sensitivity by reifying **Finally Blocks**.

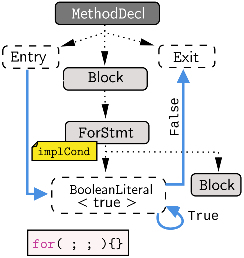
Challenges

- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers
  - Exception-sensitivity by reifying **Finally Blocks**.
  - Implicit condition in empty **For loops**



Challenges

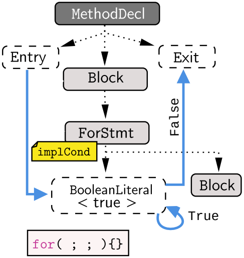
- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers
  - Exception-sensitivity by reifying **Finally Blocks**.
  - Implicit condition in empty **For loops**



- We used Circular attribute to compute mutually depended attributes

Challenges

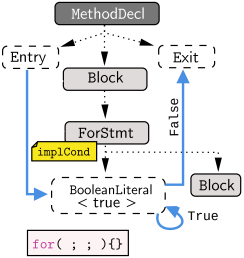
- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers
  - Exception-sensitivity by reifying **Finally Blocks**.
  - Implicit condition in empty **For loops**



- We used Circular attribute to compute mutually depended attributes
  - The attribute may depends on its own value

Challenges

- We used HOAs to extend the AST with new subtrees
  - Call to `close()` for resources in **Try With Resources**
  - **Static** and **Instance** initializers
  - Exception-sensitivity by reifying **Finally Blocks**.
  - Implicit condition in empty **For loops**



- We used Circular attribute to compute mutually depended attributes
  - The attribute may depends on its own value
  - Computes a fixpoint

Client analyses

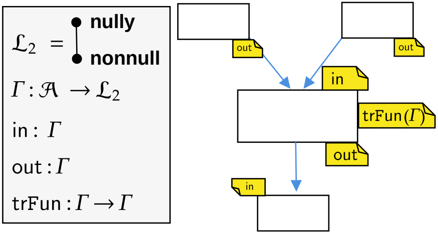
We validate IntraJ by implementing three different dataflow analyses:

- `NullPointerAnalysis` - **NPA** **MAY** - **FORWARD**
- `LiveVariableAnalysis` - **LVA** **MAY** - **BACKWARD**
- `DeadAssignmentAnalysis` - **DAA** uses **LVA**

Client analyses

We validate IntraJ by implementing three different dataflow analyses:

▶ `NullPointerAnalysis` - **NPA**	**MAY - FORWARD**
▶ `LiveVariableAnalysis` - **LVA**	**MAY - BACKWARD**
▶ `DeadAssignmentAnalysis` - **DAA**	uses **LVA**



$\mathcal{L}_2 =$ **nully** / **nonnull**

$\Gamma : \mathcal{A} \to \mathcal{L}_2$

in : Γ

out : Γ

trFun : $\Gamma \to \Gamma$

Client analyses

We validate IntraJ by implementing three different dataflow analyses:

- `NullPointerAnalysis` - **NPA** — **MAY - FORWARD**
- `LiveVariableAnalysis` - **LVA** — **MAY - BACKWARD**
- `DeadAssignmentAnalysis` - **DAA** — uses **LVA**

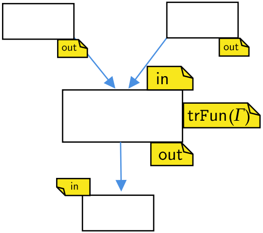
$\mathcal{L}_2 =$ ● **nully** | ● **nonnull**

$\Gamma : \mathcal{A} \to \mathcal{L}_2$

in : Γ

out : Γ

trFun : $\Gamma \to \Gamma$



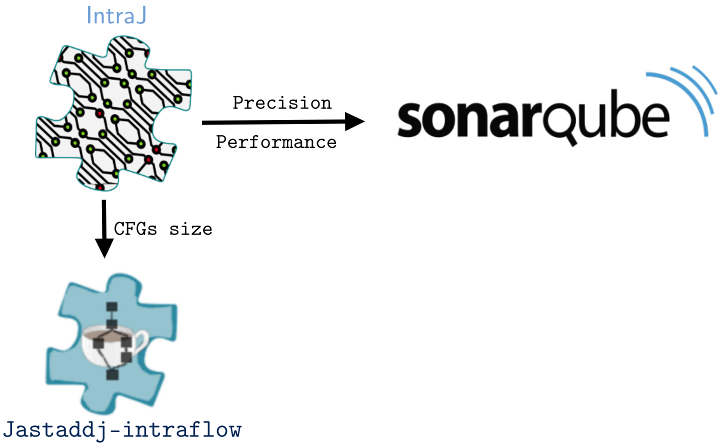
- Default behaviour for `CFGNodes`

```
trFun(Γ){
  return Γ;
}
```

- Specialised behaviour for `AssignExpr`

```
trFun(Γ){
  if(rhs.mayBeNull())
    Γ.put(lhs.decl(),nully);
  else
    Γ.put(lhs.decl(),nonnull);
  return Γ;
}
```

Overview



IntraJ

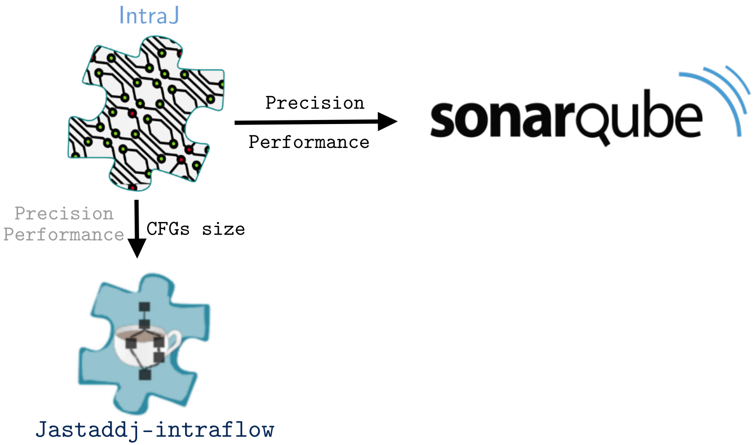
Precision

Performance

CFGs size

Jastaddj-intraflow

Overview



IntraJ

Precision

Performance

sonarqube

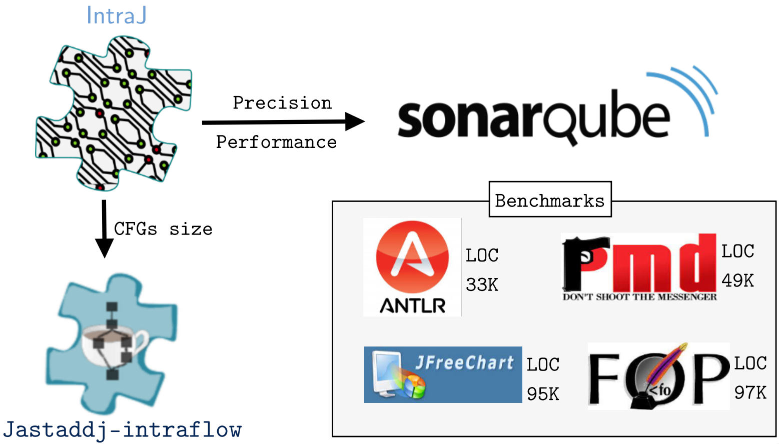
Precision

Performance

CFGs size

Jastaddj-intraflow

Overview



Benchmarks

ANTLR	LOC 33K
pmd DON'T SHOOT THE MESSENGER	LOC 49K
JFreeChart	LOC 95K
FOP	LOC 97K

Benchmark Projects

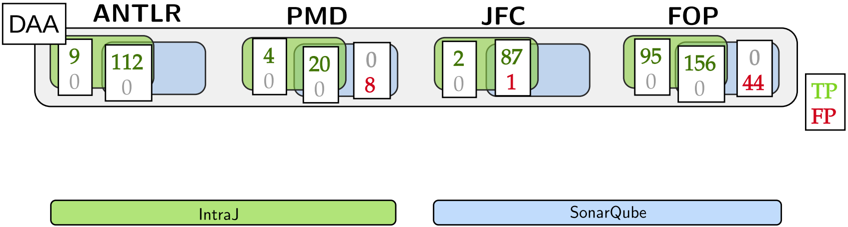
IntraJ reduces the CFGs size by 30% - 40%

Benchmark	Qty	IntraJ	JJI	%
ANTLR	Nodes	76˙925	116˙523	**-39.9**
	Edges	85˙028	136˙528	**-37.7**
PMD	Nodes	103˙739	182˙864	**-43.2**
	Edges	108˙639	202˙842	**-46.4**
JFC	Nodes	219˙419	331˙368	**-33.7**
	Edges	220˙256	363˙642	**-39.4**
FOP	Nodes	239˙096	347˙125	**-31.1**
	Edges	240˙068	379˙269	**-36.6**

By removing all the redundant nodes

IntraJ vs SonarQube

We evaluated IntraJ against SonarQube



DAA

ANTLR
9
0
112
0

PMD
4
0
20
0
0
8

JFC
2
0
87
1

FOP
95
0
156
0
0
44

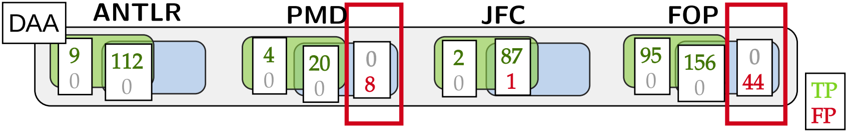
TP
FP

IntraJ

SonarQube

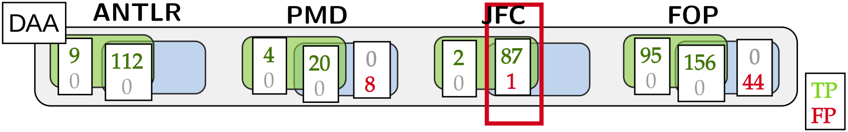
IntraJ vs SonarQube

We evaluated **IntraJ** against **SonarQube**



IntraJ vs SonarQube

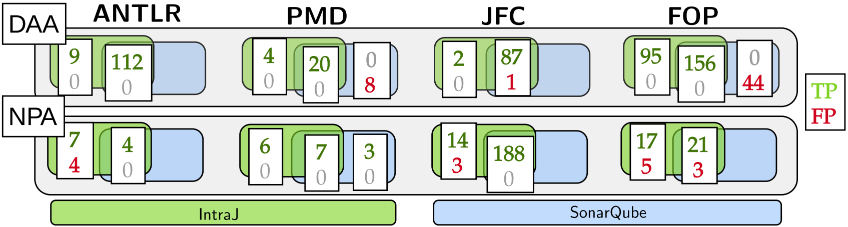
We evaluated IntraJ against SonarQube



IntraJ 🟩 SonarQube 🟦

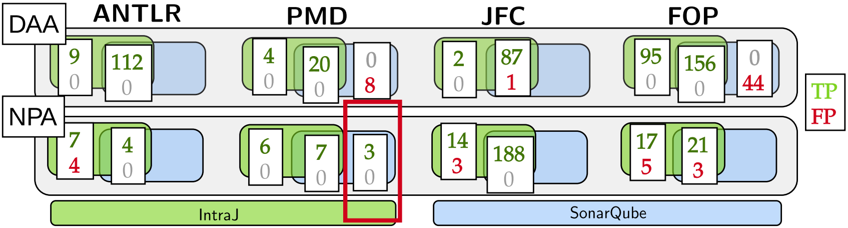
IntraJ vs SonarQube

We evaluated IntraJ against SonarQube



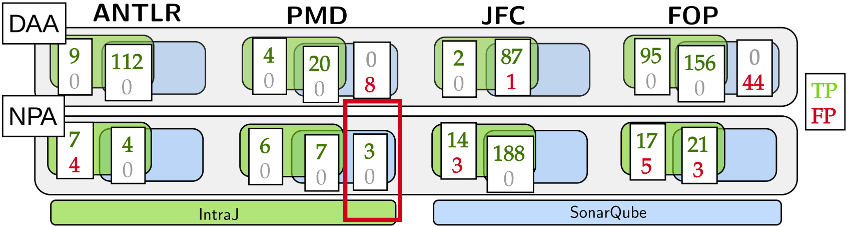
IntraJ vs SonarQube

We evaluated IntraJ against SonarQube



IntraJ vs SonarQube

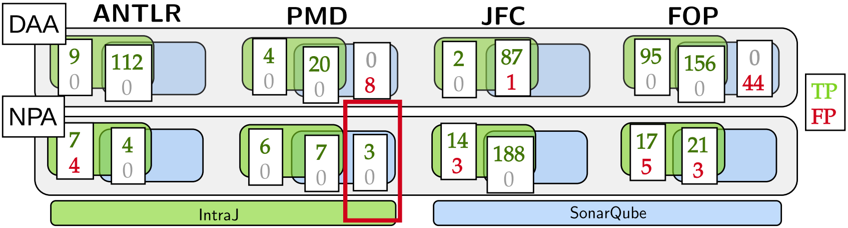
We evaluated IntraJ against SonarQube



Benchmark	Baseline (s)		DAA (s)		NPA (s)	
	IntraJ	SQ	IntraJ	SQ	IntraJ	SQ
ANTLR	2.14	4.91	0.53	0.24	0.90	12.35
PMD	3.56	10.76	0.47	0.18	0.80	12.40
JFC	4.29	10.81	0.75	0.24	1.62	10.71
FOP	4.42	17.20	0.67	0.34	1.42	19.25

IntraJ vs SonarQube

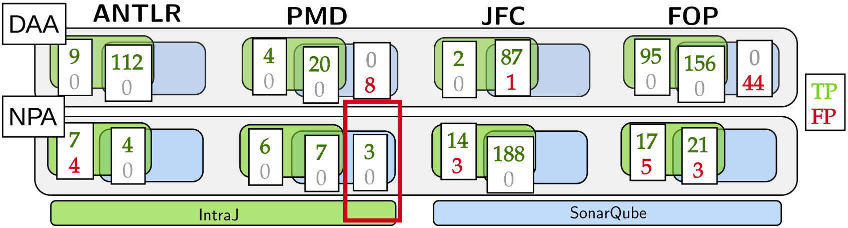
We evaluated **IntraJ** against **SonarQube**



Benchmark	**Baseline** (s)		**DAA** (s)		**NPA** (s)	
	IntraJ	SQ	IntraJ	SQ	IntraJ	SQ
ANTLR	2.14	4.91	0.53	0.24	0.90	12.35
PMD	3.56	10.76	0.47	0.18	0.80	12.40
JFC	4.29	10.81	0.75	0.24	1.62	10.71
FOP	4.42	17.20	0.67	0.34	1.42	19.25

IntraJ vs SonarQube

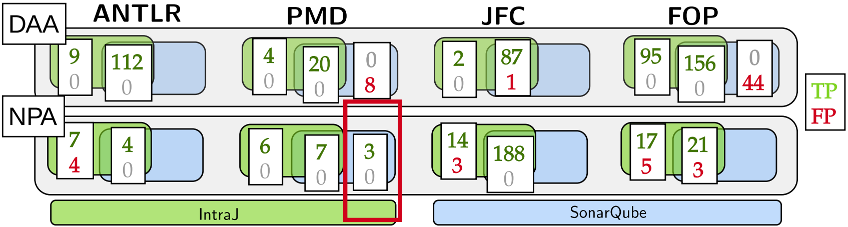
We evaluated **IntraJ** against **SonarQube**



Benchmark	Baseline (s)		DAA (s)		NPA (s)	
	IntraJ	SQ	IntraJ	SQ	IntraJ	SQ
ANTLR	2.14	4.91	0.53	0.24	0.90	12.35
PMD	3.56	10.76	0.47	0.18	0.80	12.40
JFC	4.29	10.81	0.75	0.24	1.62	10.71
FOP	4.42	17.20	0.67	0.34	1.42	19.25

IntraJ vs SonarQube

We evaluated **IntraJ** against **SonarQube**



Benchmark	Baseline (s)		DAA (s)		NPA (s)	
	IntraJ	SQ	IntraJ	SQ	IntraJ	SQ
ANTLR	2.14	4.91	0.53	0.24	0.90	12.35
PMD	3.56	10.76	0.47	0.18	0.80	12.40
JFC	4.29	10.81	0.75	0.24	1.62	10.71
FOP	4.42	17.20	0.67	0.34	1.42	19.25

IntraCFG & IntraJ

We presented IntraCFG, a language independent RAGs framework that overcomes the limitations of earlier approaches:

- **High precision**
- **$\geq$30% fewer nodes**
- **Better performance** for large code bases

IntraCFG & IntraJ

We presented IntraCFG, a language independent RAGs framework that overcomes the limitations of earlier approaches:

- **High precision**
- **$\geq$30% fewer nodes**
- **Better performance** for large code bases

Moreover, we presented:

- IntraJ,an instance of IntraCFG for Java 4-7

IntraCFG & IntraJ

We presented IntraCFG, a language independent RAGs framework that overcomes the limitations of earlier approaches:

- **High precision**
- **≥30% fewer nodes**
- **Better performance** for large code bases

Moreover, we presented:

- IntraJ,an instance of IntraCFG for Java 4-7
- Concise code for CFG and client analyses

IntraCFG & IntraJ

We presented IntraCFG, a language independent RAGs framework that overcomes the limitations of earlier approaches:

- **High precision**
- $\geq$**30% fewer nodes**
- **Better performance** for large code bases

Moreover, we presented:

- IntraJ,an instance of IntraCFG for Java 4-7
- Concise code for CFG and client analyses
- Competitive to **SonarQube**

IntraCFG & IntraJ

We presented IntraCFG, a language independent RAGs framework that overcomes the limitations of earlier approaches:

- **High precision**
- $\geq$**30% fewer nodes**
- **Better performance** for large code bases

Moreover, we presented:

- IntraJ,an instance of IntraCFG for Java 4-7
- Concise code for CFG and client analyses
- Competitive to **SonarQube**

Thank you for your attention!