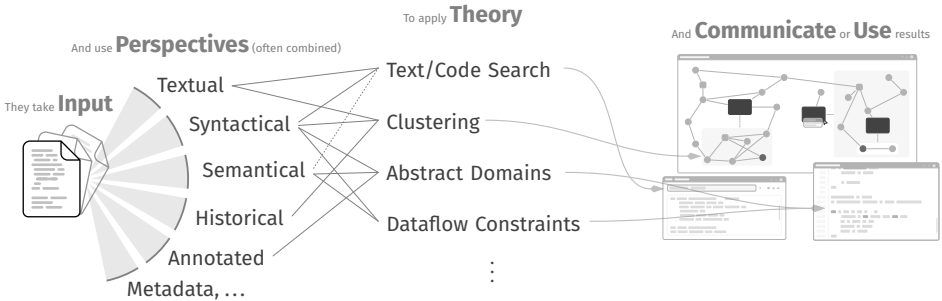


Static Analysis in the Real World

Software Engineering
Programming Languages

universität ulm

Outline

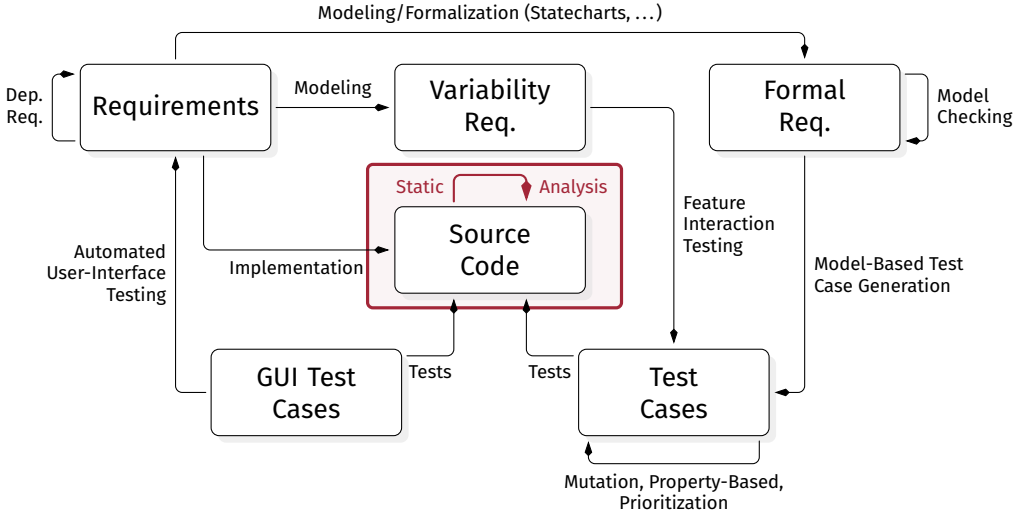
1. A Small Recap
2. Introduction
3. Real-World Static Analyzers
4. Conclusion

Hans Houdini, mag Kraulen mehr als Abstract Interpretation

Sieglinde, vollzeit-cutie, hoch-motiviert

1. A Small Recap

Embedding a Landscape



What is static analysis?

Discover *syntactic/semantic properties* of programs **without** running them.[RY20]

What is abstract interpretation (in this context)?

A theory and framework to *systematically* derive sound static analyses by **abstracting** program semantics.[Cou21]

A way to soundly over-approximate all possible program behaviors.

On the Bread and Butter of Static Analysis

1. What are examples for semantic properties?

2. How do we describe semantic properties of programs?

3. What is a (complete) lattice, why do we need it?

4. What is described by a Galois connection?

5. Given a closed interval domain $[l .. u]$, what are the abstract semantics of addition?

On the Bread and Butter of Static Analysis

1. What are examples for semantic properties?
   "We never dereference null pointers", "variable x is always positive", "array a is sorted at a given point", "we never divide by zero", …

2. How do we describe semantic properties of programs?
   We use (po-)sets containing all program states that satisfy the property.

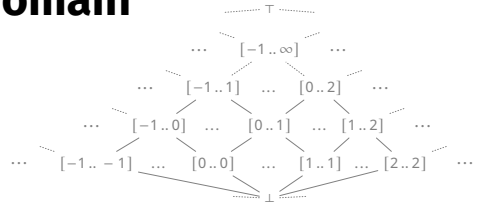
3. What is a (complete) lattice, why do we need it?
   A poset, in which all subsets have a least upper bound (join), a greatest lower bound (meet), as well as a top and bottom element. We need lattices to describe how to combine properties (e.g., at control-flow joins) and to ensure that our analysis terminates.

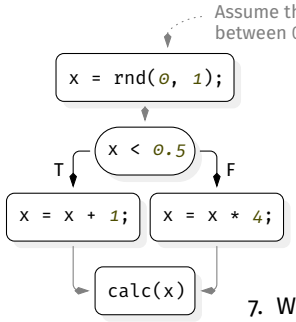
4. What is described by a Galois connection?
   Mapping concrete program states to abstract properties (abstraction, α) and back (concretization, γ).

5. Given a closed interval domain $[l \,..\, u]$, what are the abstract semantics of addition?
   $\mathbb{E}[\![\ a \ + \ b\ ]\!]\rho \stackrel{\text{def}}{=} [l_a + l_b \,..\, u_a + u_b] \qquad (a \in [l_a \,..\, u_a], b \in [l_b \,..\, u_b])$
   "We add the lower bounds and the upper bounds, assuming the environment ρ."
   (assuming a language like Java and ignoring overflows)

Recap: Applying the Interval Domain

6. Do we need widening (∇) for this domain?





We use these semantics (use your intuition for the rest):

$\mathbb{E}[\![\ \texttt{if(c) a else b}\ ]\!]\rho \stackrel{\text{def}}{=} \mathbb{E}[\![\ \texttt{a}\ ]\!](\mathbb{C}[\![\ \texttt{c}\ ]\!]\rho) \sqcup \mathbb{E}[\![\ \texttt{b}\ ]\!](\mathbb{C}[\![\ \neg\texttt{c}\ ]\!]\rho)$

"Evaluate a assuming that c is true, evaluate b assuming that c is false, and join the results. "

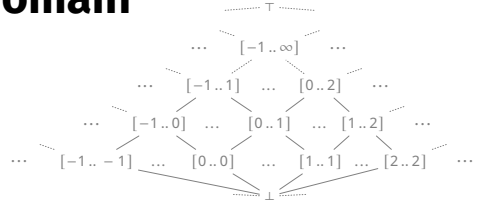
$\mathbb{E}[\![\ \texttt{a + b}\ ]\!]\rho \stackrel{\text{def}}{=} [l_a + l_b \,..\, u_a + u_b]$

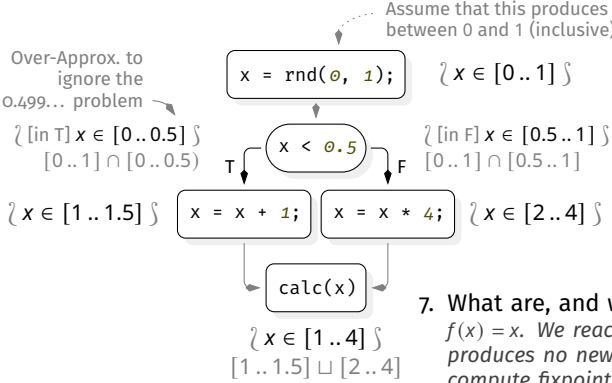
$\mathbb{E}[\![\ \texttt{a * b}\ ]\!]\rho \stackrel{\text{def}}{=} [\min S \,..\, \max S]$ with $S = \{\, l_a l_b, l_a u_b, u_a l_b, u_a u_b \,\}$
$(\texttt{a} \in [l_a \,..\, u_a], \texttt{b} \in [l_b \,..\, u_b])$

7. What are, and when do we need fixpoints?

Recap: Applying the Interval Domain

6. Do we need widening (∇) for this domain?
   Yes, because the lattice contains infinite ascending chains, e.g., $[0..0] \sqsubseteq [0..1] \sqsubseteq [0..2] \sqsubseteq \ldots$





We use these semantics (use your intuition for the rest):

$$\mathbb{E}[\![\ \texttt{if(c) a else b}\ ]\!]\rho \stackrel{\text{def}}{=} \mathbb{E}[\![\ \texttt{a}\ ]\!](\mathbb{C}[\![\ \texttt{c}\ ]\!]\rho) \sqcup \mathbb{E}[\![\ \texttt{b}\ ]\!](\mathbb{C}[\![\ \neg\texttt{c}\ ]\!]\rho)$$

"Evaluate a assuming that c is true, evaluate b assuming that c is false, and join the results."

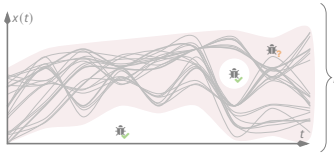
$$\mathbb{E}[\![\ \texttt{a + b}\ ]\!]\rho \stackrel{\text{def}}{=} [l_a + l_b \,..\, u_a + u_b]$$

$$\mathbb{E}[\![\ \texttt{a * b}\ ]\!]\rho \stackrel{\text{def}}{=} [\min S \,..\, \max S] \quad \text{with } S = \{\, l_a l_b, l_a u_b, u_a l_b, u_a u_b \,\} \quad (\texttt{a} \in [l_a \,..\, u_a], \texttt{b} \in [l_b \,..\, u_b])$$

7. What are, and when do we need fixpoints?
   $f(x) = x$. We reach fixpoints when the repeated application of semantics produces no new info. When analyzing loops or recursive functions, we compute fixpoints to find a stable state (of all code executions).

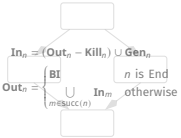
2. Introduction

What we have… **Theory**



Abstractions

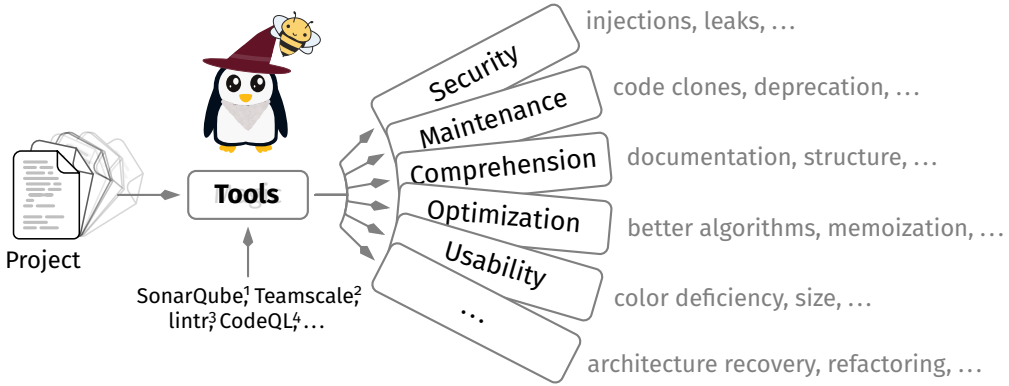
Abstractions



Data Flow Analysis

…

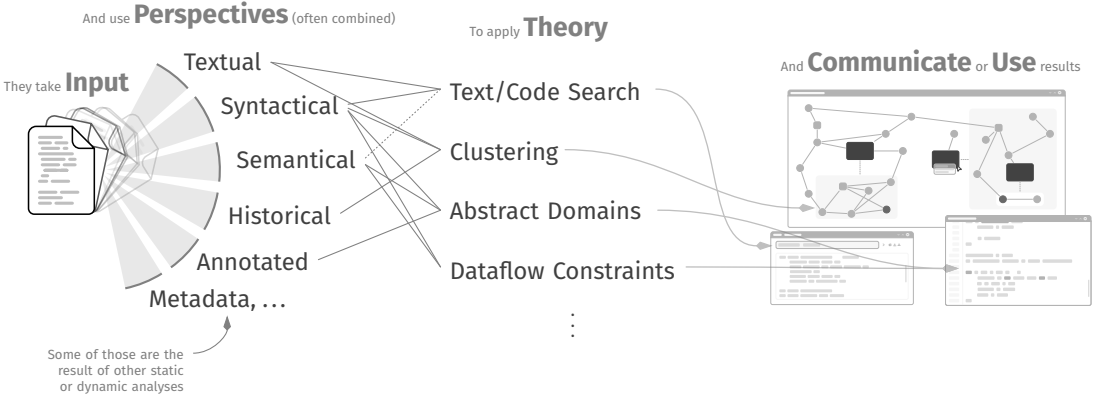
What we want… **Tools**



"Any sufficiently advanced technology is indistinguishable from magic." — Arthur C. Clarke

[1] sonarsource.com, [2] teamscale.com, [3] lintr.r-lib.org, [4] codeql.github.com

What do they… do?



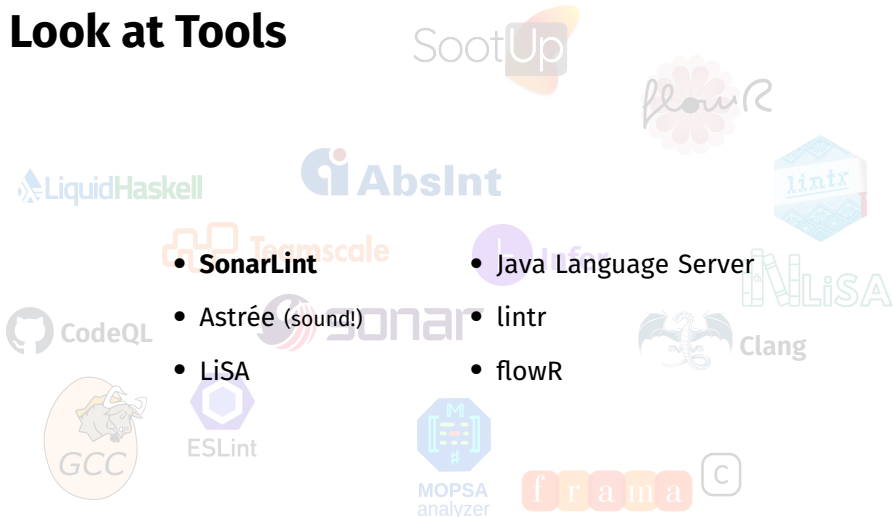
3. Real-World Static Analyzers

Let's Look at Tools

There are countless…

github.com/analysis-tools-dev/static-analysis

Let's Look at Tools



- **SonarLint**
- Astrée (sound!)
- LiSA
- Java Language Server
- lintr
- flowR

There are countless…

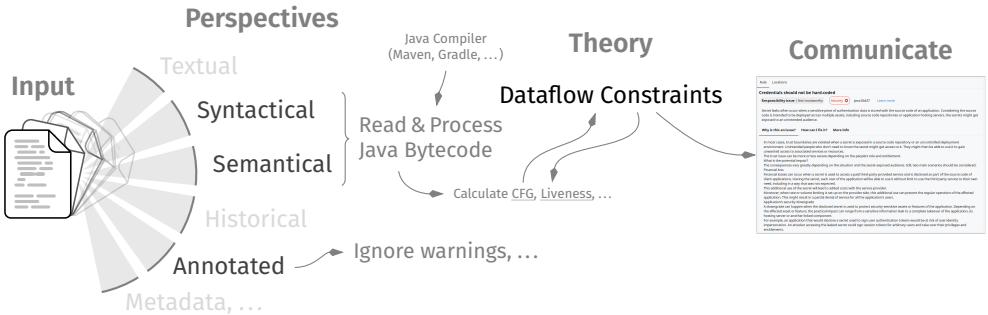
github.com/analysis-tools-dev/static-analysis

SonarLint

- Support for multiple languages (15+)
  - Widely varying support and rules
  - We focus on **sonar-java**

Separate frontends and analyzers per language!

733 rules!



github.com/SonarSource/sonar-java

SonarLint

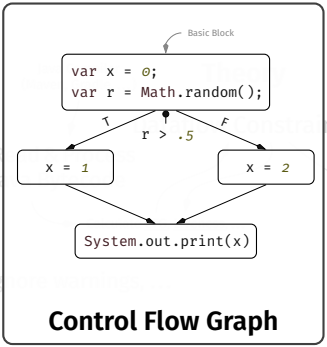
- Support for multiple languages (15+)
  - Widely varying support and rules
  - We focus on **sonar-java**

Separate frontends and analyzers per language!

733 rules!

```
var x = 0;
var r = Math.random();
if(r > .5) {
   x = 1;
} else {
   x = 2;
}
System.out.print(x);
```

Code

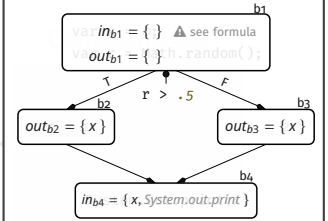


Control Flow Graph

$$\mathbf{In}_n = (\mathbf{Out}_n - \mathbf{Kill}_n) \cup \mathbf{Gen}_n$$

$$\mathbf{Out}_n = \begin{cases} \mathbf{BI} & n \text{ is End} \\ \bigcup_{m \in succ(n)} \mathbf{In}_m & \text{otherwise} \end{cases}$$

With fixpoint iteration



Liveness[KSK09, p. 26]

github.com/SonarSource/sonar-java

[KSK09] *Data Flow Analysis: Theory and Practice* (Khedker, Sanyal, and Karkare)

SonarLint — Unused Assignments

- Analyze “Dead Stores” (in 312 loc / 260 cloc):

```
int x = 0;
x = 42;
```

- Use liveness analysis to obtain out_n of each basic block in the CFG
- Check if assignments (`x =`, `x++`, …) are in out_n and resolved
- Check overwrites in the same basic block
- Minor special handling for **`try-finally`** blocks, …

Implementation of Rule S1854

SonarLint — Unused Assignments

- Analyze “Dead Stores” (in 312 loc / 260 cloc):

```
int x = 0;
x = 42;
```

- Use liveness analysis to obtain out_n of each basic block in the CFG
- Check if assignments (`x =`, `x++`, ...) are in out_n and resolved
- Check overwrites in the same basic block
- Minor speci

```
77 LiveVariables liveVariables = LiveVariables.analyze(cfg);
78 // Liveness analysis provides information only for block boundaries,
       so we should do analysis between elements within blocks
79 for(CFG.Block block : cfg.blocks()) {
80   checkElements(block, liveVariables.getOut(block), methodSymbol);
81 }
```

Implementation of Rule S1854

SonarLint — Hardcoded Credentials

- It does not always have to be that heavy! (116 loc / 87 cloc may suffice)
- For example, to identify hardcoded credentials:

```
new PasswordAuthentication("password", "secret".toCharArray());
```

- Traverse the Abstract Syntax Tree (AST)
- Check calls against a long list of signatures (currently 7664) with problematic indices
- Check if the arguments are “constant”
  visiting the dataflow links and checking for predefined “plain text”

Implementation of Rule S6437

SonarLint — Hardcoded Credentials

- It does not always have to be that heavy! (116 loc / 87 cloc may suffice)
- For example, to identify hardcoded credentials:

```
new PasswordAuthentication("password", "secret".toCharArray());
```

- Traverse the Abstract Syntax Tree (AST)
- Check calls against a long list of signatures (currently 7664) with problematic indices
- Cl… vi…

```
107 for (int targetArgumentIndex : method.indices) {
108   ExpressionTree argument = arguments.get(targetArgumentIndex);
109   var secondaryLocations = new ArrayList<JavaFileScannerContext.Location>();
110   if (isExpressionDerivedFromPlainText(argument,
                secondaryLocations, new HashSet<>())) {
111     reportIssue(argument, ISSUE_MESSAGE, secondaryLocations, null);
112   }
113 }
```

Implementation of Rule S6437

Let's Look at Tools

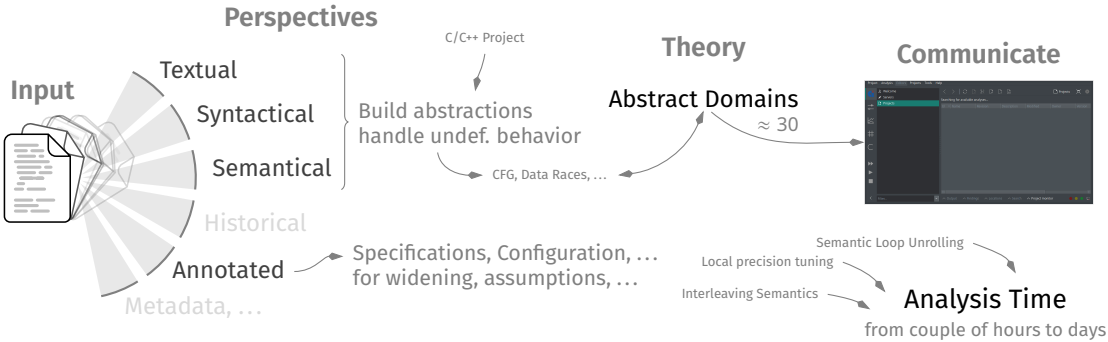
- SonarLint
- **Astrée (sound!)**
- LiSA
- Java Language Server
- lintr
- flowR

There are countless…

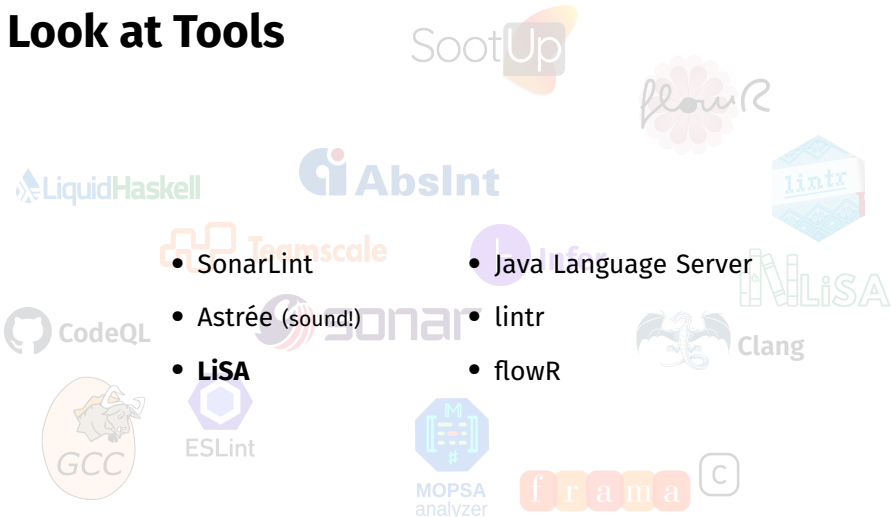
github.com/analysis-tools-dev/static-analysis

Astrée

- *Analyseur statique de logiciels temps-réel embarqués*
  Static analyzer for real-time embedded software
- Proprietary, sound*y* static analyzer for C/C++ — 100+ directives and intrinsics, 140+ options; 250 kloc of Ocaml, 240 kloc C/C++
- Uses abstract domains for timing validation, buffer overflows, …



Let's Look at Tools



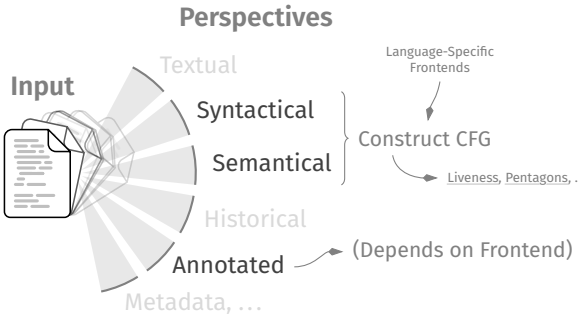
- SonarLint
- Astrée (sound!)
- **LiSA**
- Java Language Server
- lintr
- flowR

There are countless…

github.com/analysis-tools-dev/static-analysis

LiSA

- (Largely) Language Independent Library for Static Analysis → Similar to Mopsa or Apron
- Custom frontends for Rust, Go, Python, EVM, …



Let's have a closer look!

github.com/lisa-analyzer/lisa

LiSA — Interval Analysis[Cou21, p. 389]

lisa-analyses/src/main/java/it/unive/lisa/analysis/numeric/Interval.java (simplified)

```
57  Interval TOP = new Interval(IntInterval.INFINITY);
62  Interval BOTTOM = new Interval(null);

273 public Interval lubAux(Interval other) {
276   var newL = getLow().min(other.getLow());
277   var newH = getHigh().max(other.getHigh());
278   return new Interval(newLow, newHigh);
279 }

282 public Interval glbAux(Interval other) {
284   var newL = getLow().max(other.getLow());
285   var newH = getHigh().min(other.getHigh());
287   if(newLow.compareTo(newHigh) > 0) return bottom();
289   return new Interval(newLow, newHigh);
290 }
```

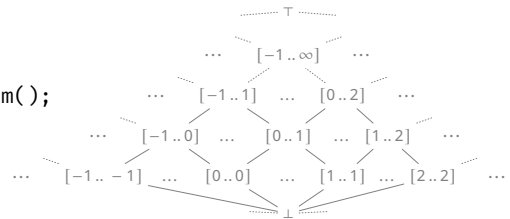
Widening, Narrowing, Assume, Satisfies, …

$$\top = [-\infty \,..\, \infty] \qquad \text{Top}$$

$$\bot = \varnothing \qquad \text{Bottom}$$

$$\bigsqcup_k [\ell_k \,..\, h_k] = [\min(\ell_k) \,..\, \max(h_k)] \qquad \text{Join}$$

$$\bigsqcap_k [\ell_k \,..\, h_k] = [\max(\ell_k) \,..\, \min(h_k)] \qquad \text{Meet}$$



[Cou21] "Principles of Abstract Interpretation" (Cousot)

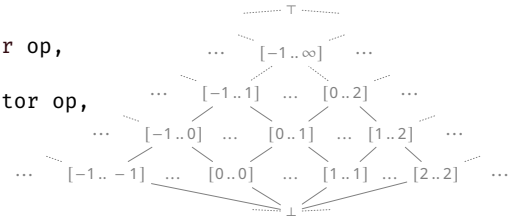
LiSA — Interval Analysis[Cou21, p. 389] Semantics

When to create which interval?

lisa-analyses/src/main/java/it/unive/lisa/analysis/numeric/Interval.java (simplified)

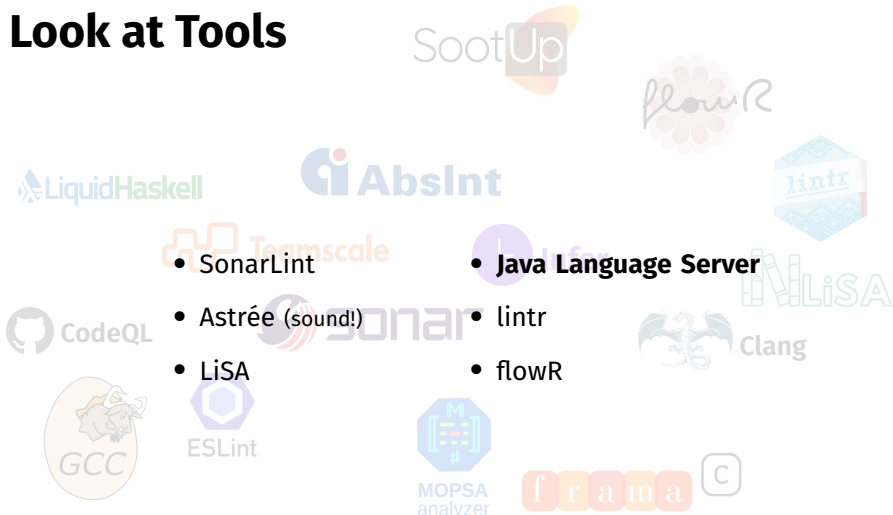
```
144 public Interval evalNonNullConstant(Constant constant,
146     ProgramPoint pp, SemanticOracle oracle) {
148   if(constant.getValue() instanceof Integer) {
149     var i = (Integer) constant.getValue();
150     return new Interval(i, i);
151   }
152   return top();
153 }

157 public Interval evalUnaryExpression(UnaryOperator op,
158     Interval arg, ...) { ... }
188 public Interval evalBinaryExpression(BinaryOperator op,
189     Interval left, Interval right, ...) { ... }
```

Fold-like Evaluation



[Cou21] "Principles of Abstract Interpretation" (Cousot)

Let's Look at Tools



- SonarLint
- Astrée (sound!)
- LiSA
- **Java Language Server**
- lintr
- flowR

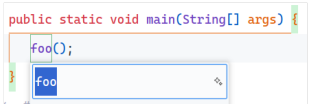
There are countless…

github.com/analysis-tools-dev/static-analysis

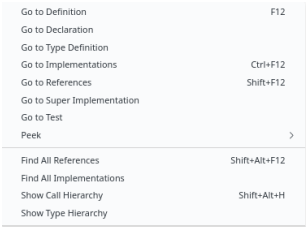
Java Language Server

- Uses the Language Server Protocol to provide static analysis for Java

Code Actions



Rename Refactoring



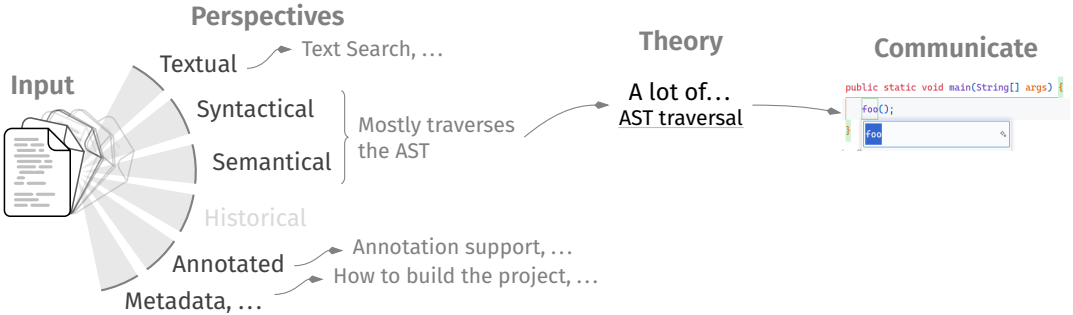
...

- Relies on the Eclipse JDT Language Server

github.com/eclipse-jdtls/eclipse.jdt.ls

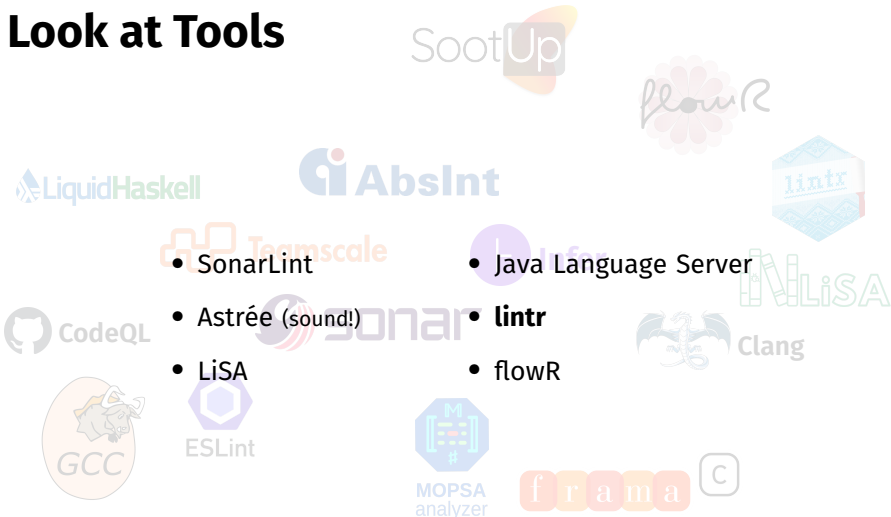
Java Language Server

- Uses the Language Server Protocol to provide static analysis for Java
  Renaming, Code Actions, …
- Relies on the Eclipse JDT Language Server



github.com/eclipse-jdtls/eclipse.jdt.ls

Let's Look at Tools



- SonarLint
- Astrée (sound!)
- LiSA
- Java Language Server
- **lintr**
- flowR

There are countless…

github.com/analysis-tools-dev/static-analysis

lintr

Why is this… special?

- A linter for the R programming language

```
x <- 4
f <- function() x
body(f) <- quote(y)
y <- 42
f() # 42
```

```
`if` <- function(...) 42
if(TRUE) print(3) # 42
```

```
x <- 2
`<-` <- `*`
x <- 21 # 42
```

- Common static analysis strategies have their… problems with R
- Most of R's users are no computer scientists (just a small set of existing work)
- So… how does *lintr* do it?
  - ~~Dataflow Constraints?~~
  - ~~Abstract Domains?~~
  - ~~Control Flow Graphs?~~
  - AST ~~Traversal?~~ (mostly) Pattern Matching and Evaluation!

github.com/r-lib/lintr

[Flü+19] "R melts brains: an IR for first-class environments and lazy effectful arguments" (Flückiger et al.)

lintr — Under the Hood

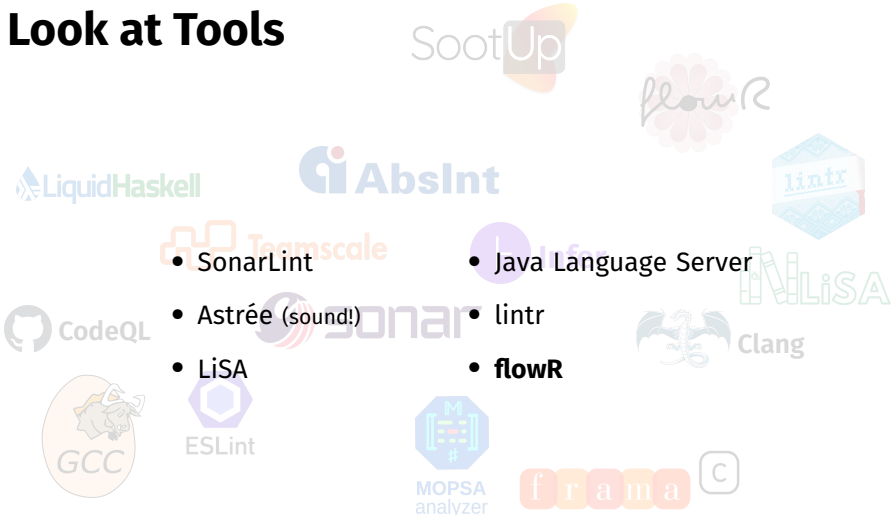
≈ 100 rules

R/object_usage_linter.R

```
magic <- "
   expr[LEFT_ASSIGN or EQ_ASSIGN]/expr[2][FUNCTION or OP-LAMBDA]
   | expr_or_assign_or_help[EQ_ASSIGN]/expr[2][FUNCTION or OP-LAMBDA]
   | equal_assign[EQ_ASSIGN]/expr[2][FUNCTION or OP-LAMBDA]
   | //SYMBOL_FUNCTION_CALL[text() = 'assign']/parent::expr/following-sibling::
       expr[2][FUNCTION or OP-LAMBDA]
   | //SYMBOL_FUNCTION_CALL[text() = 'setMethod']/parent::expr/following-sibling::
       expr[3][FUNCTION or OP-LAMBDA]"
```

- What does this do?
  This XPATH expression matches assignments to functions
- It is very rigid (no alias tracking, flow sensitivity, …)
- And it… cheats:
  `try_silently(eval(envir = env,parse(text = code, keep.source = TRUE)))`
  It simply runs (parts of) the program (including side-effects), …

Let's Look at Tools



- SonarLint
- Astrée (sound!)
- LiSA
- Java Language Server
- lintr
- **flowR**

There are countless…

github.com/analysis-tools-dev/static-analysis

Existing Work

		references	op. assignments	dyn. assignments	value trace	control flow	non-std. eval.	special operators	function calls	libraries	reflection	side effects	static scope	dynamic scope	type inference	pointer analysis	external files	pre-processors	hooks	FFI	expose info.	goal	method
	flowR	●	●	○	○	●	○	●	●	○	○	○	●	○		○					●	static analysis	AST visitor
[Cha23]	{checkglobals}	●	●	○			○	○	○	○			○							○		missing libs.	AST visitor
[Lan+23]	{CodeAnalysis}	●	●	○	○	●	○	○	○	○			●		○		○				●	static analysis	AST visitor
[Lan+18]	{CodeDepends}	●	●	○			○	○	○	○		○	●								●	static analysis	AST visitor
[Tie23]	{codetools}	●	●	○	○	○	○	○	○				●								●	static analysis	AST visitor
[Csa23]	{cyclocomp}					○							●									static analysis	AST visitor
[Rob16]	{DependenciesGraphs}							○	○												○	visualize	regex
[Kar21]	{dfgraph}	●	○			○		○													○	static analysis	AST visitor
[Fab23]	{flow}	●	○			●		○		○						○					○	vis., debug	AST visitor
[App24]	{foodwebr}							○													○	visualize	regex
[Ben22]	{globals}	●	●		○	○	○	○	○				●									distributed env.	AST visitor
[Lai23]	{languageserver}	●	●	○				○	○				●								○	editor support	XPath, visitor
[Hes+23]	{lintr}	●	●	○		○		○		○			●									linting	XPath, visitor
[Bra18]	{mvbutils}								○													utility	regex
[van23]	{PaRe}		○			○			○													code review	regex
[Pad21]	{pkgstats}	●	○					○	○				○									pkg. insights	ctags & gtags
[Lau22]	{Rclean}	●	●	○			○	○	○	○			●									debug/refactor	PDG traversal
[Rod21]	{rco}	●	●		●	○							●									optimization	AST visitor
[Pat19]	{rflowgraph}	●							○	○											○	call graph	AST visitor
[UT19]	{rstatic}	●	●		○	●		○					●								●	static analysis	AST visitor
[UT21]	{RTypeInference}	●	●		○	●		○					●		●						○	type inference	AST visitor
[BG20]	{SimilaR}	●	○			○			○				●									plagiarism	PDG, visitor
[Kal+14]	FastR	●	●		○			○	○	○			●									execute R	AST visitor
[R C23]	GNU R	●	●	○	●	○	○	●	○				●									execute R	bytecode
[19]	MRO	●	●	○	○	○																execute R	bytecode
[Nea14]	pqR	●	●	○	●	○	○	●	○				●									execute R	bytecode
[AS12]	Random	●	○	○	○	○				○			○									abstract int.	trace & visitor
[Garo4]	RCC	●	●		●	○	○		○		○	○	●									transpile to C	CFG, bytecode
[Ber13]	renjin	●	●	○	○	●		○	○			○	●		●							execute R	SSA, CFG
[Flü+20]	R̃	●	●	○	●	●	○	●	○				●	○	●							execute R	SSA, bytecode
[Sen+17]	ROSA	●	●			○			○		○		●		●	○						optimization	visitor
[Pos23]	RStudio	●	●	○				○	○				●									editor support	AST visitor

flowR

- A static analysis framework for R
- Developed here, at Ulm University 

  Florian Sihler, Julian Schubert, Oliver Gerstl, Lars Pfrenger, Johanna Scheck, Felix Schlegel, Ruben Dunkel, Thomas Schöller, Tim Schmidt, …
- Let's get back to R:

```r
x <- 4
f <- function() x
body(f) <- quote(y)
y <- 42
f() # 42
```

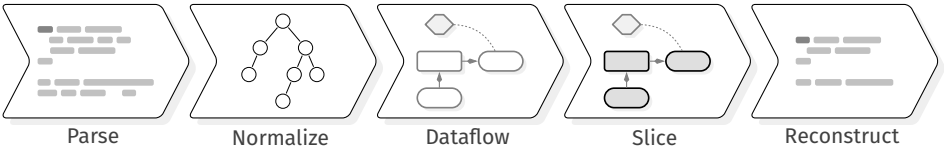
```r
`if` <- function(...) 42
if(TRUE) print(3) # 42
```

```r
x <- 2
`<-` <- `*`
x <- 21 # 42
```

- We have to intertwine dataflow- and control-flow analysis…

github.com/flowr-analysis/flowr

flowR — Architecture




```
sum  <- 0
prod <- 1
n    <- 10

for (i in 1:(n-1)) {
   sum  <- sum + i
   prod <- prod * i
}

cat("Sum:", sum, "\n")
cat("Product:", prod, "\n")
```

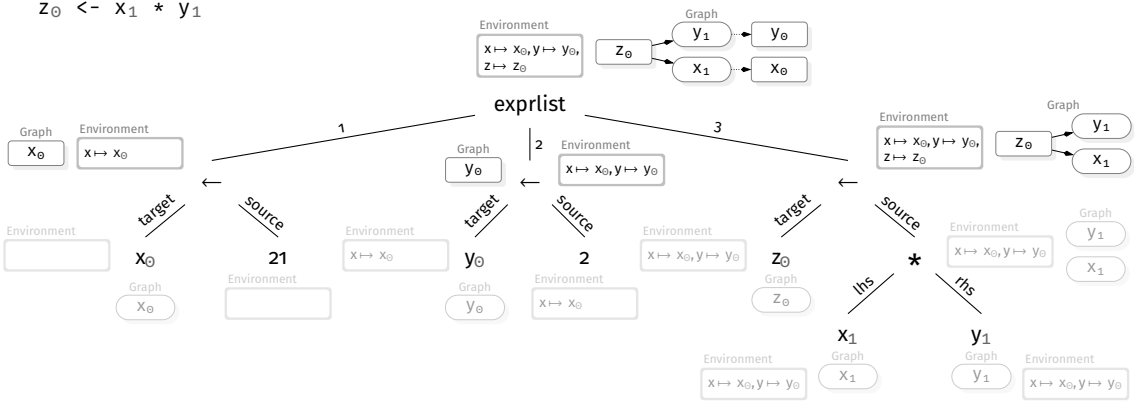
slice(10, sum)

```
sum  <- 0
prod <- 1
n    <- 10

for (i in 1:(n-1)) {
   sum  <- sum + i
   prod <- prod * i
}

cat("Sum:", sum, "\n")
cat("Product:", prod, "\n")
```

github.com/flowr-analysis/flowr/tree/main/src/dataflow

flowr — Dataflow

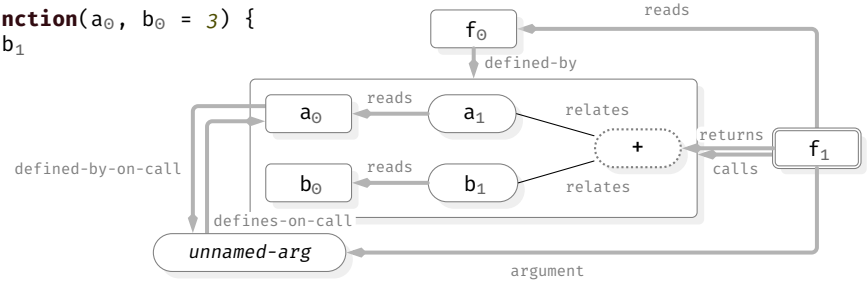
```
x₀ <- 21
y₀ <- 2
z₀ <- x₁ * y₁
```



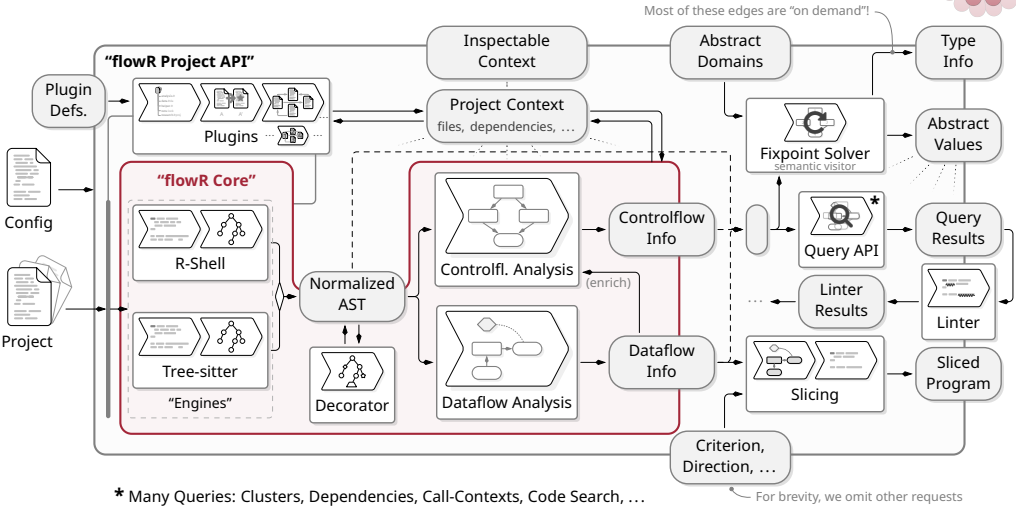
Without Value tracking

flowR — There Is More…

```
f₀ <- function(a₀, b₀ = 3) {
   a₁ + b₁
}
f₁(39)
```



flowR — A LOT More…



* Many Queries: Clusters, Dependencies, Call-Contexts, Code Search, …

4. Conclusion

Soundness and Completeness Revisited

- We want to prove properties of programs (e.g., no overflow, shapes, …)
- However, thanks to Rice [Ric53] we know:

  Rice's theorem states that all nontrivial semantic properties of programs are undecidable. [Cou21, p. 100]

Soundness

- All properties we derive are true (but we may miss some)
- If we report bugs for violated properties, we produce no false negative

Completeness

- We are able to infer all interesting properties in the program
- If we report bugs for violated properties, we produce no false positive

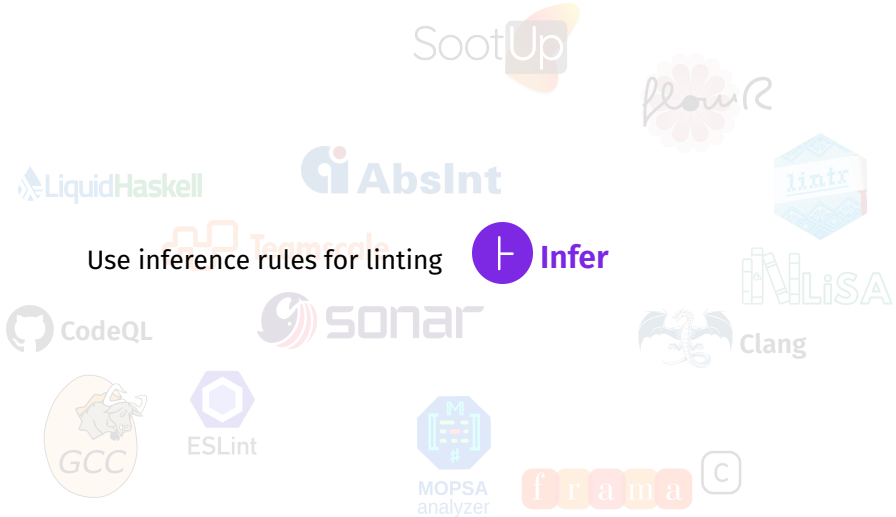
Handling Errors

- Analyzers tend to **favor soundyness over completeness**
  Better to report a false positive than to miss a bug
- Yet many cases are hard to handle (e.g., `eval` with *any* possible effect)
- Issues are usually ranked by severity (and sometimes confidence)
- Cutoffs may be applied if there are too many alarms
- These is still a lot of research required [ENO08; Cou21, p. 705]

Oh There Are so Many Tools…

Verify properties at compile time

Oh There Are so Many Tools…



Oh There Are so Many Tools…

Determine registers, scheduling, types …

For example:

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}$$

Clang

And There Are Many More Strategies

- Static Analysis
  analyzing the code without executing it, for all possible runtime scenarios
- Dynamic Analysis
  executing the program with specific input(s) to observe its behavior
- Hybrid Analysis
  combining both strategies

Program analysis is important. Additionally AND combined with testing.

Back to the Questions

1. How would you capture what a *property* is? ✔
2. How would you phrase that one property is “better” than another? ✔
3. For what operations would you *not* use a control-flow graph? ✔
4. Why can’t there be a fully automatic, sound, and complete static analyzer for general programs? ✔
5. What (big) additional challenges do you see in the real-world? ✔
6. How would you describe soundness and completeness? ✔
7. How would you describe Abstraction Interpretation? ✔

Thanksies!

slides and source code

feedback form

References I

[19] *Microsoft R Open Source*. 2019. URL: https://github.com/microsoft/microsoft-r-open.

[App24] Lewin Appleton-Fox. *foodwebr: Visualise Function Dependencies*. 2024. URL: https://github.com/lewinfox/foodwebr.

[AS12] Gianluca Amato and Francesca Scozzari. "Random: R-Based Analyzer for Numerical Domains". In: *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*. Ed. by Nikolaj S. Bjørner and Andrei Voronkov. Vol. 7180. Lecture Notes in Computer Science. Springer, 2012, pp. 375–382. DOI: 10.1007/978-3-642-28717-6_29. URL: https://doi.org/10.1007/978-3-642-28717-6%5C_29.

[Ben22] Henrik Bengtsson. *globals: Identify Global Objects in R Expressions*. R package version 0.16.2. 2022. URL: https://CRAN.R-project.org/package=globals.

[Ber13] Alexander Bertram. "Renjin: A new r interpreter built on the jvm". In: *The r user conference, useR! 2013 july 10-12 2013 university of castilla-la mancha, albacete, spain*. Vol. 10. 30. 2013, p. 105.

[BG20] Maciej Bartoszuk and Marek Gagolewski. *SimilaR: R Source Code Similarity Evaluation*. R package version 1.0.8. 2020. URL: https://CRAN.R-project.org/package=SimilaR.

[Bra18] Mark V. Bravington. *mvbutils: Workspace Organization, Code and Documentation Editing, Package Prep and Editing, Etc.* R package version 2.8.232. 2018. URL: https://CRAN.R-project.org/package=mvbutils.

[Cha23] Joris Chau. *checkglobals: Static Analysis of R-Code Dependencies*. R package version 0.1.0. 2023. URL: https://CRAN.R-project.org/package=checkglobals.

[Cou21] Patrick Cousot. "Principles of Abstract Interpretation". In: (2021).

[Csa23] Gabor Csardi. *cyclocomp: Cyclomatic Complexity of R Code*. R package version 1.1.1. 2023. URL: https://CRAN.R-project.org/package=cyclocomp.

[ENO8] Pär Emanuelsson and Ulf Nilsson. "A Comparative Study of Industrial Static Analysis Tools". In: *Proceedings of the 3rd International Workshop on Systems Software Verification, SSV 2008, Sydney, Australia, February 25-27, 2008*. Ed. by Ralf Huuck, Gerwin Klein, and Bastian Schlich. Vol. 217. Electronic Notes in Theoretical Computer Science. Elsevier, 2008, pp. 5–21. DOI: 10.1016/J.ENTCS.2008.06.039. URL: https://doi.org/10.1016/j.entcs.2008.06.039.

[Fab23] Antoine Fabri. *flow: View and Browse Code Using Flow Diagrams*. R package version 0.2.0. 2023. URL: https://CRAN.R-project.org/package=flow.

[Flü+19] Olivier Flückiger et al. "R melts brains: an IR for first-class environments and lazy effectful arguments". In: *Proceedings of the 15th ACM SIGPLAN International Symposium on Dynamic Languages, DLS 2019, Athens, Greece, October 20, 2019*. Ed. by Stefan Marr and Juan Fumero. ACM, 2019, pp. 55–66. DOI: 10.1145/3359619.3359744. URL: https://doi.org/10.1145/3359619.3359744.

References II

[Flü+20] Olivier Flückiger et al. “Sampling optimized code for type feedback”. In: *Proceedings of the 16th ACM SIGPLAN International Symposium on Dynamic Languages*. 2020, pp. 99–111.

[Gar04] John Garvin. *RCC: A compiler for the R language for statistical computing*. Rice University, 2004.

[Hes+23] Jim Hester et al. *lintr: A 'Linter' for R Code*. R package version 3.1.0. 2023. URL: https://CRAN.R-project.org/package=lintr.

[Kal+14] Tomas Kalibera et al. “A fast abstract syntax tree interpreter for R”. In: *ACM SIGPLAN Notices* 49.7 (2014), pp. 89–102.

[Kar21] Dan Kary. *dfgraph: Visualize R Code with Data Flow Graphs*. R package version 3.1.0. 2021. URL: https://github.com/dkary/dfgraph.

[KSK09] Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. *Data Flow Analysis: Theory and Practice*. 1st. USA: CRC Press, Inc., 2009. ISBN: 0849328802.

[Lai23] Randy Lai. *languageserver: Language Server Protocol*. R package version 0.3.16. 2023. URL: https://CRAN.R-project.org/package=languageserver.

[Lan+18] Duncan Lang et al. *CodeDepends. Analysis of R Code for Reproducible Research and Code Comprehension*. manual. 2018. URL: https://CRAN.R-project.org/package=CodeDepends.

[Lan+23] Duncan Lang et al. *CodeAnalysis. Tools for static analysis of R code*. manual. 2023. URL: https://github.com/duncantl/CodeAnalysis.

[Lau22] Matthew Lau. *Rclean: A Tool for Writing Cleaner, More Transparent Code*. R package version 1.1.8. 2022. URL: https://github.com/MKLau/Rclean.

[Nea14] Radford M Neal. “Speed Improvements in pqR: Current Status and Future Plans”. In: (2014). URL: https://glizen.com/radfordneal/ftp/pqR-dsc.pdf.

[Pad21] Mark Padgham. *pkgstats*. R package version 0.0.3. 2021. URL: https://github.com/ropensci-review-tools/pkgstats.

[Pat19] Evan Patterson. *flowgraph: Flow graps for R*. 2019. URL: https://github.com/IBM/rflowgraph.

[Pos23] Posit team. *RStudio: Integrated Development Environment for R*. Posit Software, PBC. Boston, MA, 2023. URL: http://www.posit.co/.

[R C23] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: https://www.R-project.org/.

[Ric53] Henry Gordon Rice. “Classes of recursively enumerable sets and their decision problems”. In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.

[Rob16] Titouan Robert. *DependenciesGraphs: Dependencies visualization between functions and environments*. R package version 0.3. 2016.

References III

[Rod21] Juan Cruz Rodriguez. *rco: The R Code Optimizer*. R package version 1.0.2. 2021. URL: https://CRAN.R-project.org/package=rco.

[RY20] Xavier Rival and Kwangkeun Yi. "Introduction to Static Analysis: An Abstract Interpretation Perspective". In: (2020).

[Sen+17] Rathijit Sen et al. *ROSA: R Optimizations with Static Analysis*. 2017. arXiv: 1704.02996 [cs.PL].

[Tie23] Luke Tierney. *codetools: Code Analysis Tools for R*. R package version 0.2-19. 2023. URL: https://CRAN.R-project.org/package=codetools.

[UT19] Nick Ulle and Duncan Temple Lang. *rstatic: Low-level Static Analysis Tools for R Code*. R package version 0.1.0-0006. 2019.

[UT21] Nick Ulle and Duncan Temple Lang. *RTypeInference: Infer Types of Inputs and Outputs for R Expressions*. R package version 0.5.0-0030. 2021.

[van23] Maarten van Kessel. *PaRe: A Way to Perform Code Review or QA on Other Packages*. R package version 0.1.12. 2023. URL: https://github.com/darwin-eu-dev/PaRe.