Constructing a Static Program Slicer

Specifically for R Programs

Usage of R

“The R Journal”[8]
JIF: 1.673 JCR 2021

Rank 7 Worldwide[5]
PYPL Index May 2023

> 19 000 Packages[7]
CRAN 2023

> 2 Million Users[4]
Oracle 2012

[7] https://cran.r-project.org/
[4] https://www.oracle.com/us/corporate/press/1515738 [archived]
[8] https://journal.r-project.org/
[5] https://pypl.github.io/

Existing (Analysis) Support for R

- RStudio IDE[6] posit.co
  - Syntax-highlighting and auto-completion
  - Simple debugger
  - Refactorings (rename, extract functions and variables)
- R language server github.com/REditorSupport
  - Syntax-highlighting and auto-completion
  - Reference tracing
  - Refactorings (rename)
- {lintr} github.com/r-lib/lintr
  - Style & syntax errors
  - Potential semantic errors
- CodeDepends[1] github.com/duncantl/CodeDepends
  - Dependency analysis
  - Creation of call-graphs

Existing (Analysis) Support for R

- RStudio IDE[6] posit.co
  - Syntax-highlighting and auto-completion
  - Simple debugger
  - ~~Refactorings (rename, extract functions and variables)~~ ← Often wrong (simple heuristics)
- R language server github.com/REditorSupport
  - Syntax-highlighting and auto-completion
  - ~~Reference tracing~~
  - ~~Refactorings (rename)~~

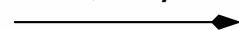
  Often wrong
  Based on XPath-expressions
- {lintr} github.com/r-lib/lintr
  - Style & syntax errors
  - ~~Potential semantic errors~~

  XPath-expressions, packages
- CodeDepends[1] github.com/duncantl/CodeDepends
  - ~~Dependency analysis~~ ← Only top scope
  - Creation of call-graphs

The Goal

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 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")
```

Requirements

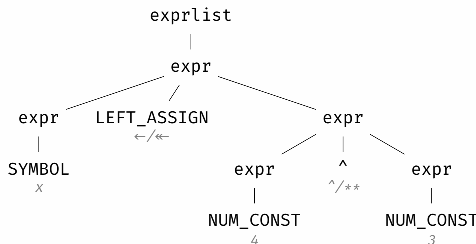
```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 cat("Product:", prod, "\n")
```

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")
```

1. Control-flow information (AST)
   - R provides a `parse` function to parse R code
   - But the produced AST is inconsistent
2. Data-flow information
3. Type information
4. Slicing algorithm

```
parse(text="x ← 4^3")
parse(text="x ↞ 4**3")
```



Requirements

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 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")
```

1. Control-flow information (AST) partially
2. Data-flow information
   - There is `CodeDepends`[1] which does not differentiate bodies
   - Otherwise: No existing data-flow analysis.
3. Type information
4. Slicing algorithm

```
assign("a", 1)
evalq(a ← 1, envir=x)
```

[1] Lang et al., *CodeDepends* (2018)

Requirements

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 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")
```

1. Control-flow information (AST) partially
2. Data-flow information nothing
3. Type information
   - `types`, `modes`, and `storage.modes` primarily at runtime
   - No existing static type inference
4. Slicing algorithm

Requirements

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 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")
```

1. Control-flow information (AST) partially
2. Data-flow information nothing
3. Type information nothing
4. Slicing algorithm
   - Basic slicing algorithm by Weiser[3]
   - Slicing with data-flow is relatively simple

[3] Weiser, "Program Slicing" (1984)

Requirements

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 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")
```

1. Control-flow information (AST) partially
2. Data-flow information nothing
3. Type information nothing
4. Slicing algorithm algorithm

Research Questions

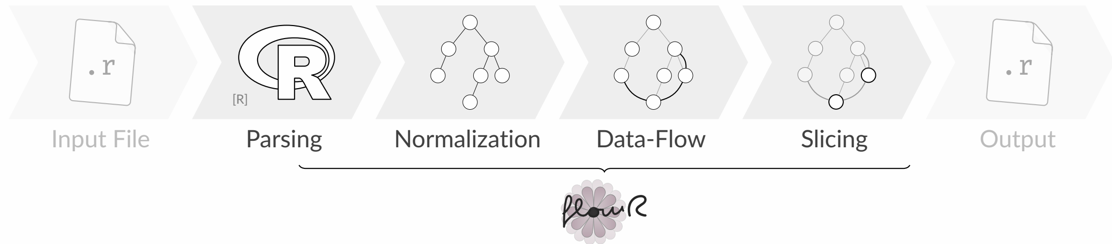
RQ1: How to normalize the AST and extract data-flow from R code?

RQ2: What are common features in R programs?

RQ3: How to deal with these common features?

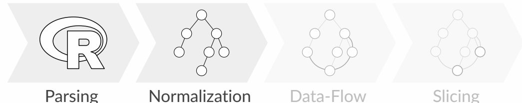
RQ4: How well performs static slicing?

The Program

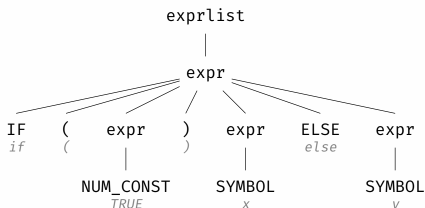


[R] https://www.r-project.org/logo/

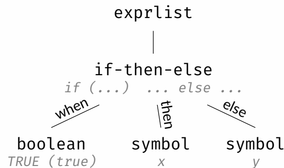
RQ1: Normalization




```
parse(text="if (TRUE) x else y")
```



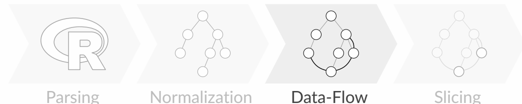
normalized



- Normalizing constants, namespacing, operators, …
- We use the "R language definition"[2] as a basis

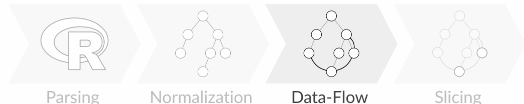
[2] R Core Team, *R Language Definition* (2023)

RQ1: Data-Flow




```
x ← 21
y ← 2
z ← x * y
```

RQ1: Data-Flow




```
x ← 21
y ← 2
z ← x * y
```

Environments
x, y, z
Graph
z_0 ← y_1 ← y_0
z_0 ← x_1 ← x_0

exprlist

1 2 3

Graph
x_0
Environments
x

Graph
y_0
Environments
y

Environments
z
Graph
z_0 ← y_1
z_0 ← x_1

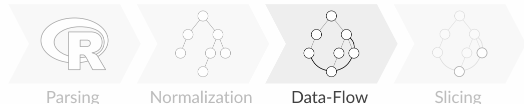
← target x, source 21
Environments
Graph
x_0
Environments

← target y, source 2
Environments
Graph
y_0
Environments

← target z, source *
Environments
Graph
z_0
Environments
Graph
y_1
x_1

* lhs x, rhs y
Environments
Graph
x_1
Graph
y_1
Environments

RQ2: Features



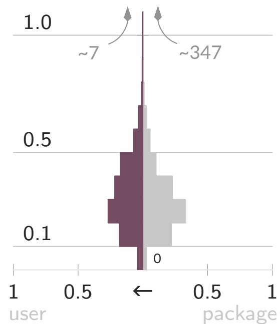
- There are many ways to modify data in R, like:
  - `a <- 1, a <<- 1, a = 1, 1 -> a, 1 ->> a`
  - `assign("a", 1), b <- "a"; assign(b, 1)`
  - `setGeneric("props", function(object) object)`
- Environments can be changed manually
- Functions can be modified at will (and at any time)
- There are different class systems, variable length arguments, and more…

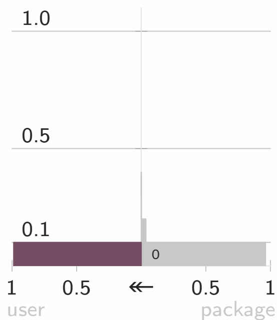
RQ2: Features, II

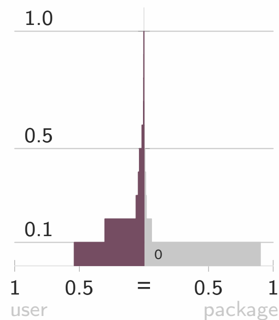
- *Assumption*: “UserRs write different code from package authors.”

UseRs	Package Authors
published scripts in social science 4230 files	top 500 CRAN packages 25 691 files

RQ2: Features, III



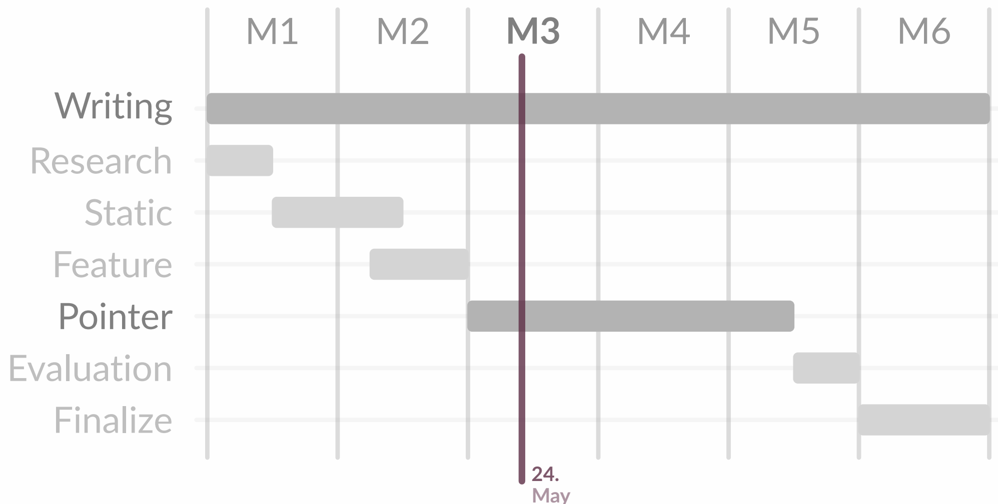




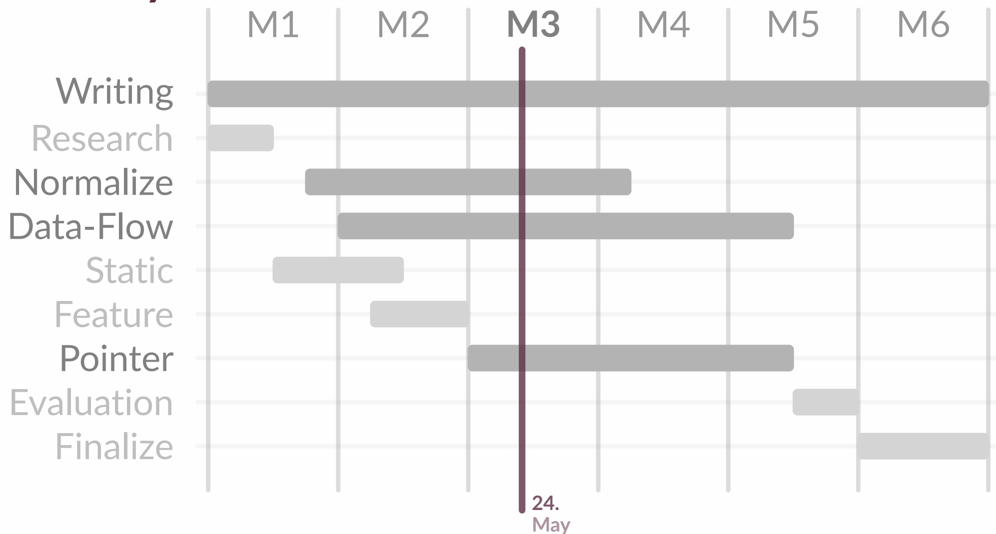
RQ2: Features, IV

- Most used packages allow to prioritize special support
- `setGeneric`, `assign`,… used very often in package code
  - Namespaces are very rarely manipulated, we do not plan supporting that.
- Data types are mostly accessed by name which allows for pointer analysis
- `.C` and `.Fortran` are used seldomly

The Plan



The Reality



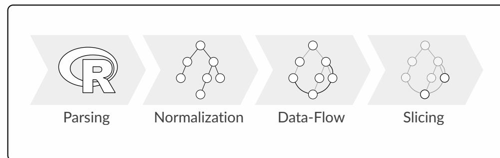
Overview

```
1  sum  ← 0
2  prod ← 1
3  n    ← 10
4
5  for (i in 1:(n-1)) {
6     sum  ← sum + i
7     prod ← prod * i
8  }
9
10 cat("Sum:", sum, "\n")
11 cat("Product:", prod, "\n")
```

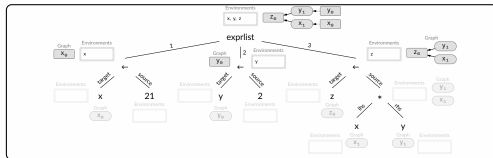
slice(10, sum) →

```
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")
```

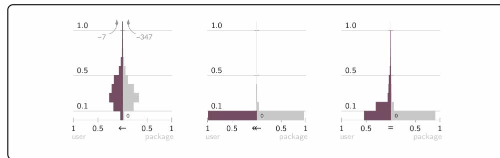
Goal



Program



Data-Flow



Features

Bibliography

[1] Duncan Lang et al. *CodeDepends. Analysis of R Code for Reproducible Research and Code Comprehension.* 2018

[2] R Core Team. *R Language Definition.* 2023

[3] Mark Weiser. "Program Slicing". July 1984

[4] *Oracle Announces Availability of Oracle Advanced Analytics for Big Data.* Feb. 2012

[5] *PYPL – PopularitY of Programming Language index.* May 2023

[6] RStudio Team. *RStudio: Integrated Development Environment for R.* 2022

[7] *The Comprehensive R Archive Network — cran.r-project.org.*

[8] *The R Journal.*

Overview of Backup-Material

Quote

Statistics

Languageserver

More Dataflow

R Fun

I have been worried for some time that R isn’t going to provide the base that we’re going to need for statistical computation in the future.

Ross Ihaka
Co-Creator of R

Used Packages

In UseR scripts:

1. ggplot2 plotting
2. dplyr data manipulation
3. tidyverse packages for data science
4. lme4 mixed-effect models
5. plyr more data manipulation

In package code:

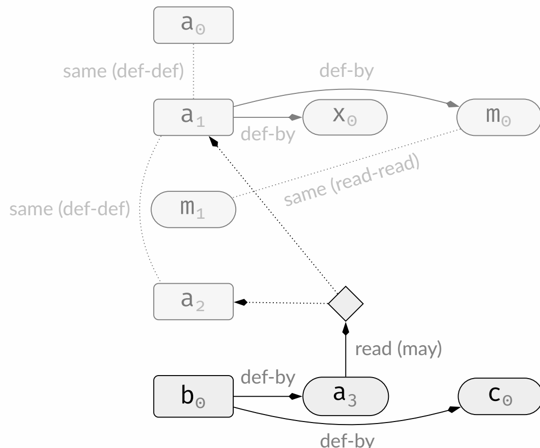
1. stats statisticsal functions
2. utils basic programming functions
3. rlang working with types
4. withr temproarily modify global state
5. testthat testing framework

Definition-Retrieval

```
paste(
   "(*|descendant-or-self::exprlist/*)[self::FUNCTION or self::OP-LAMBDA]/following-sibling
       ::SYMBOL_FORMALS[text() = '{token_quote}' and @line1 <= {row}]",
   "(*|descendant-or-self::exprlist/*)[LEFT_ASSIGN[preceding-sibling::expr[count(*)=1]/
       SYMBOL[text() = '{token_quote}' and @line1 <= {row}] and following-sibling::expr[
       @start > {start} or @end < {end}]]]",
   "(*|descendant-or-self::exprlist/*)[RIGHT_ASSIGN[following-sibling::expr[count(*)=1]/
       SYMBOL[text() = '{token_quote}' and @line1 <= {row}] and preceding-sibling::expr[
       @start > {start} or @end < {end}]]]",
   "(*|descendant-or-self::exprlist/*)[EQ_ASSIGN[preceding-sibling::expr[count(*)=1]/SYMBOL[
       text() = '{token_quote}' and @line1 <= {row}] and following-sibling::expr[@start > {
       start} or @end < {end}]]]",
   "forcond/SYMBOL[text() = '{token_quote}' and @line1 <= {row}]",
sep = "|")
```

Example Dataflow

```
> a₀ ← 3
  a₁ ← x₀ * m₀

  if(m₁ > 3) {
    a₂ ← 5
  }

  b₀ ← a₃ + c₀
```



Modifying Environments and Functions

```r
x ← new.env()
evalq(a ← 1, envir=x)
evalq(a, envir=x)
```

```r
f ← function(x) { y ← x * 3; y }
body(f)[[3]] ← quote(x)
f(2) # 2
```

Modifying Assignments

```r
f <- function(x) { body(f)[[2]] <<- 3 }
f(2) # <invisible>
f(2) # 3
```

```r
f <- function(x) a + b
f(2) # Error in f(2) : object 'a' not found
formals(f) <- alist(a=,b=40)
f(2) # 42
```

Florian Sihler

Ulm May 27, 2023