



Constructing a Static Program Slicer

Specifically for R Programs

Usage of R

Usage of R

Rank 7 Worldwide^[5]

PYPL Index May 2023

[5] <https://pypl.github.io/>

Usage of R

“The R Journal”^[8]

JIF: 1.673 JCR 2021

Rank 7 Worldwide^[5]

PYPL Index May 2023

[8] <https://journal.r-project.org/>

[5] <https://pypl.github.io/>

Usage of R

“The R Journal”^[8]

JIF: 1.673 JCR 2021

Rank 7 Worldwide^[5]

PYPL Index May 2023

> 2 Million Users^[4]

Oracle

2012

[4] <https://www.oracle.com/us/corporate/press/1515738> [archived]

[8] <https://journal.r-project.org/>

[5] <https://pypl.github.io/>

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

Existing (Analysis) Support for R

- › RStudio IDE^[6] posit.co

Existing (Analysis) Support for R

- › RStudio IDE^[6] posit.co
 - Syntax-highlighting and auto-completion
 - Simple debugger
 - Refactorings (rename, extract functions and variables)

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

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)

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

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

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

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)
- {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)
- {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
- 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
 - 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

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")
```

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**)
→

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")
```

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)

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)
2. Data-flow information

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)
2. Data-flow information
3. Type information

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)
2. Data-flow information
3. Type information
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)
 - R provides a parse function to parse R code
2. Data-flow information
3. Type information
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)
 - 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

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

parse(text="x ← 4^3")

2. Data-flow information

3. Type information

4. Slicing algorithm

Requirements

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

```
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")
```

parse(text="x ← 4^3")
parse(text="x ← 43")**

Requirements

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

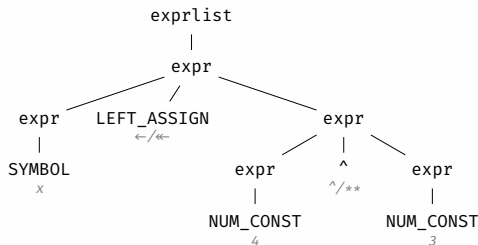
4. Slicing algorithm

```
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")
```

parse(text="x ← 4^3")
parse(text="x ← 43")**



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
3. Type information
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
 - There is CodeDepends^[1] which does not differentiate bodies
3. Type information
4. Slicing algorithm

[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
 - There is CodeDepends^[1] which does not differentiate bodies
 - Otherwise: No existing data-flow analysis.
3. Type information
4. Slicing algorithm

[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
 - There is CodeDepends^[1] which does not differentiate bodies
 - Otherwise: No existing data-flow analysis.
3. Type information
4. Slicing algorithm

assign("a", 1)

[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
 - 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
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
 - types, modes, and storage .modes primarily at runtime
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
 - 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

Requirements

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

```
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")
```

[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

- 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

Research Questions

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

Research Questions

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

RQ2: What are common features in R programs?

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?

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?

Research Questions

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

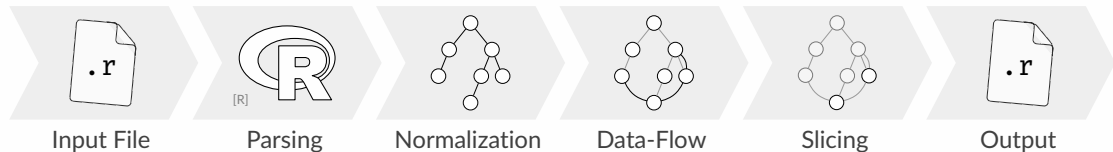
RQ2: What are common features in R programs?

RQ3: How to deal with these common features?

RQ4: How well performs static slicing?

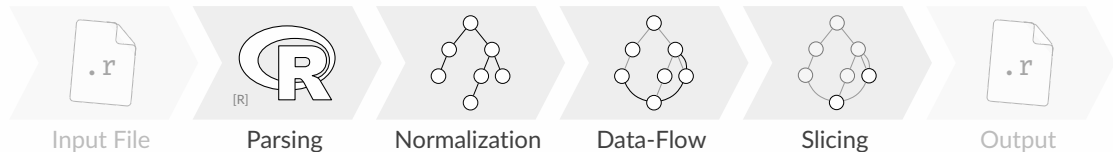
The Program

The Program



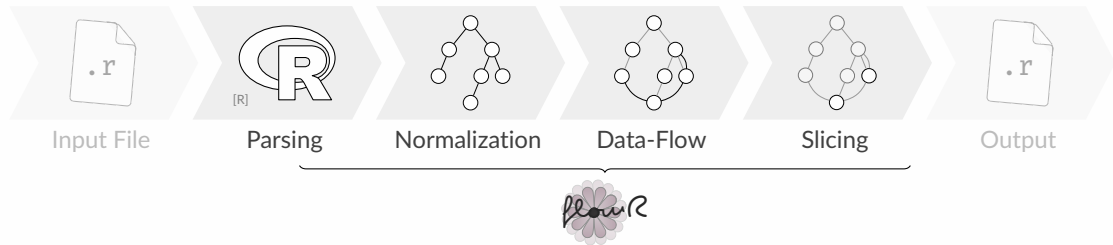
[R] <https://www.r-project.org/logo/>

The Program



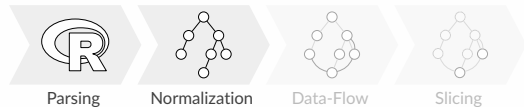
[R] <https://www.r-project.org/logo/>

The Program



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

RQ1: Normalization



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

RQ1: Normalization



Parsing



Normalization

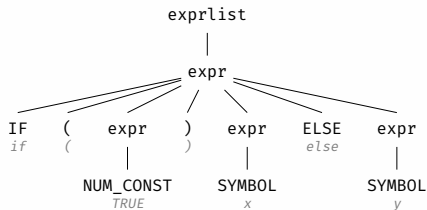


Data-Flow



Slicing

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



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

RQ1: Normalization



Parsing



Normalization

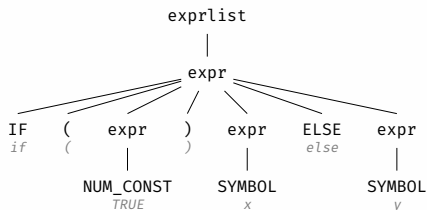


Data-Flow

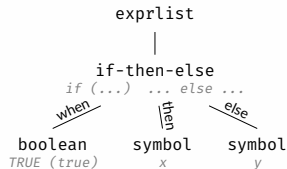


Slicing

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



normalized



RQ1: Normalization



Parsing



Normalization

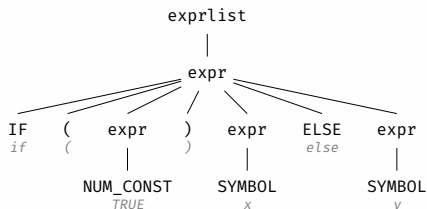


Data-Flow

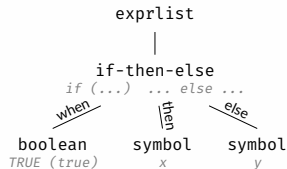


Slicing

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



normalized



> Normalizing constants, namespacing, operators, ...

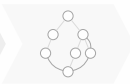
RQ1: Normalization



Parsing



Normalization

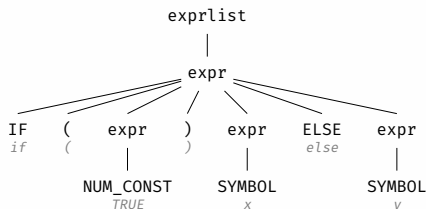


Data-Flow

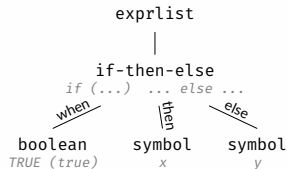


Slicing

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



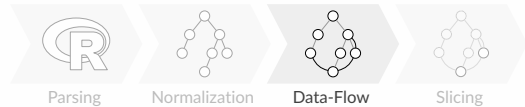
normalized



- > 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



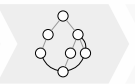
RQ1: Data-Flow



Parsing



Normalization



Data-Flow



Slicing

```
x ← 21  
y ← 2  
z ← x * y
```

RQ1: Data-Flow

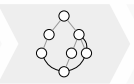
```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



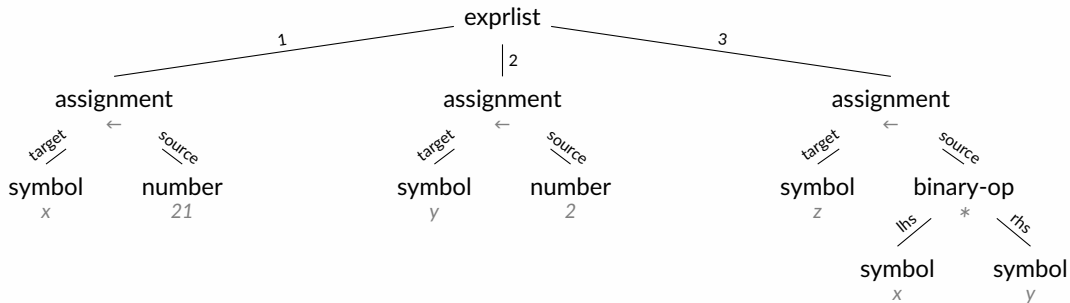
Normalization



Data-Flow



Slicing



RQ1: Data-Flow

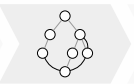
```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



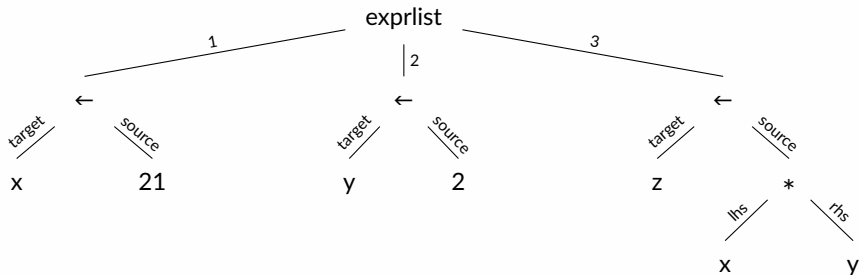
Normalization



Data-Flow



Slicing



RQ1: Data-Flow

```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



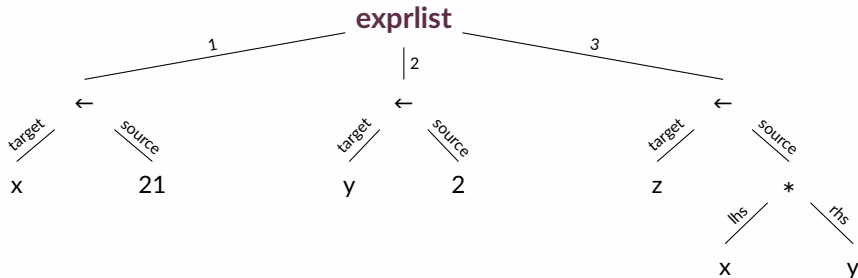
Normalization



Data-Flow



Slicing



RQ1: Data-Flow

```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



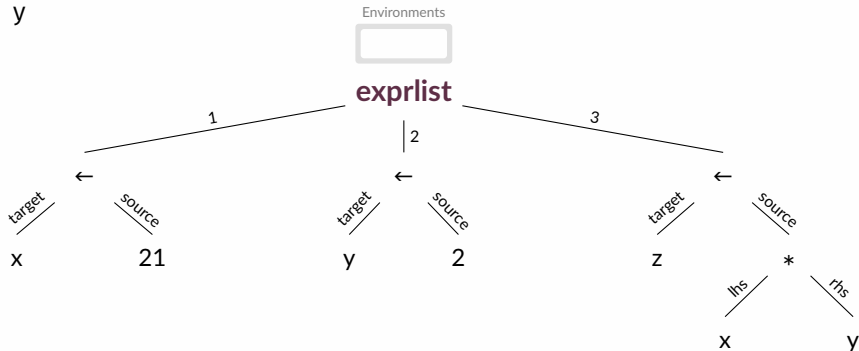
Normalization



Data-Flow



Slicing



RQ1: Data-Flow

```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



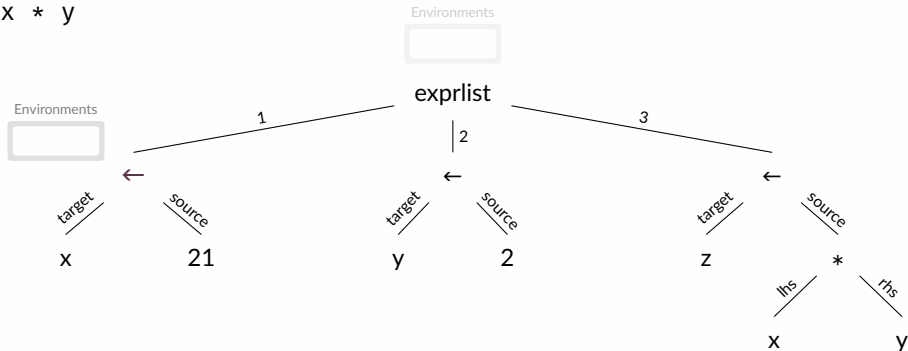
Normalization



Data-Flow

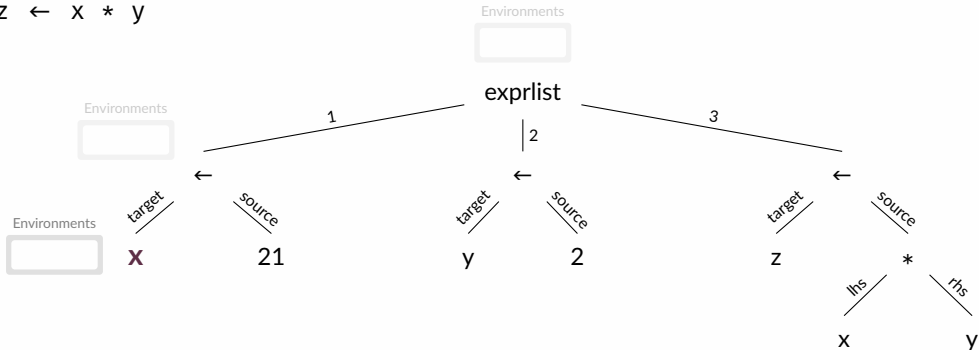


Slicing



RQ1: Data-Flow

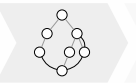
```
x ← 21
y ← 2
z ← x * y
```



Parsing



Normalization



Data-Flow

Slicing

RQ1: Data-Flow

$x \leftarrow 21$
 $y \leftarrow 2$
 $z \leftarrow x * y$



Parsing



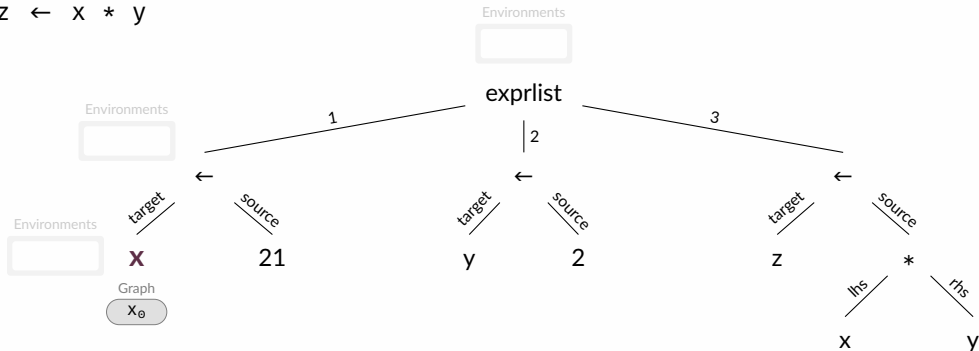
Normalization



Data-Flow



Slicing



RQ1: Data-Flow

```
x ← 21  
y ← 2  
z ← x * y
```



Parsing



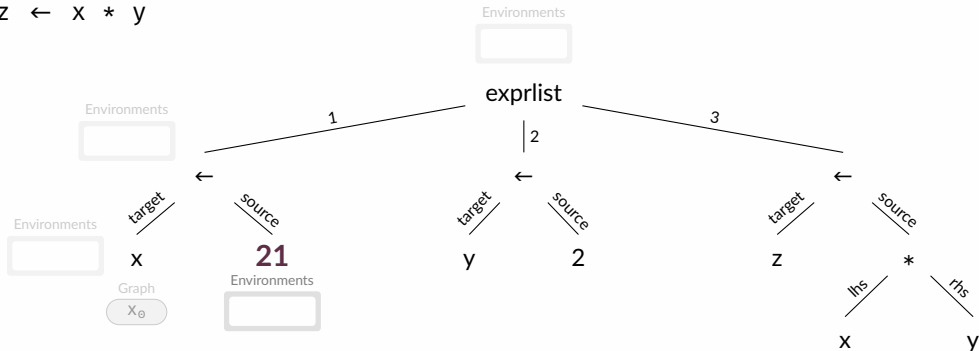
Normalization



Data-Flow



Slicing



RQ1: Data-Flow



Parsing



Normalization

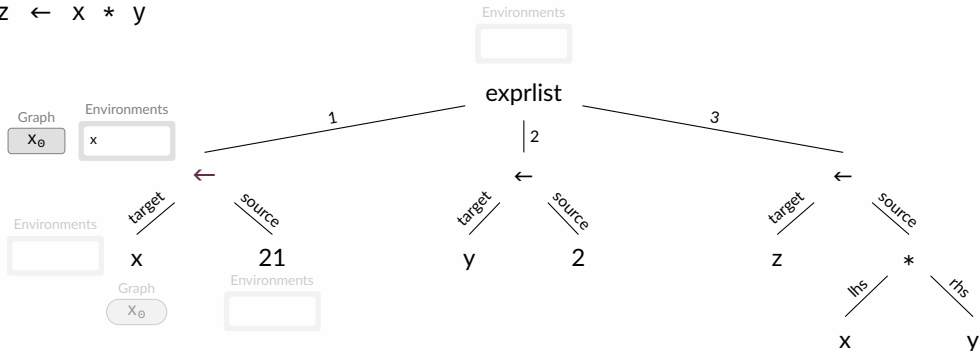


Data-Flow



Slicing

```
x ← 21
y ← 2
z ← x * y
```



RQ1: Data-Flow

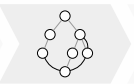
```
x ← 21
y ← 2
z ← x * y
```



Parsing



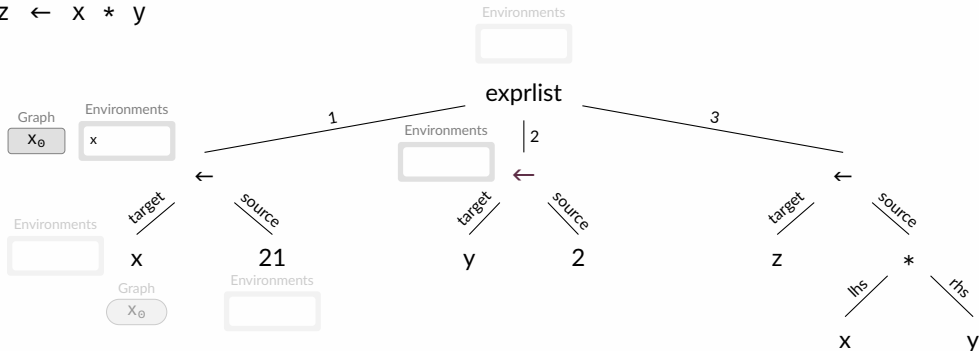
Normalization



Data-Flow



Slicing



RQ1: Data-Flow



Parsing



Normalization

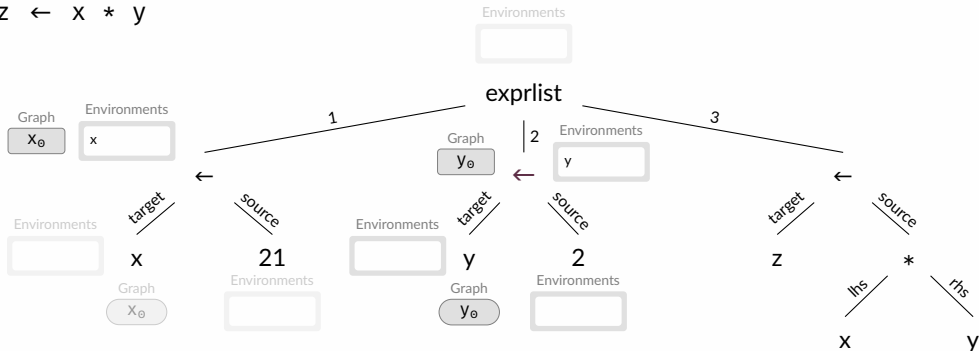


Data-Flow



Slicing

```
x ← 21
y ← 2
z ← x * y
```



RQ1: Data-Flow



Parsing



Normalization

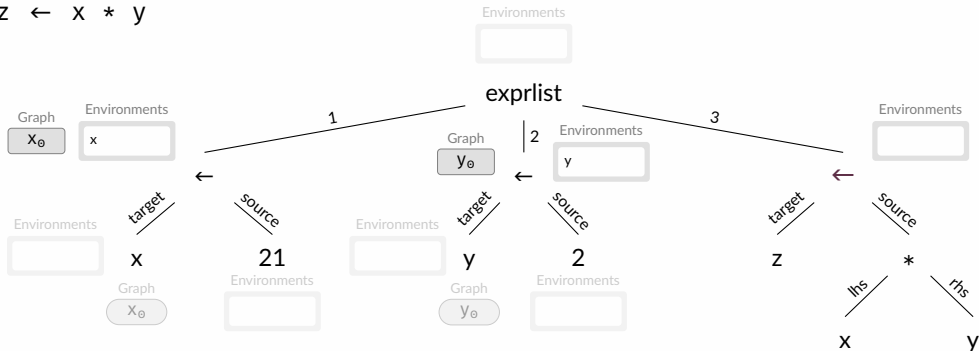


Data-Flow



Slicing

```
x ← 21
y ← 2
z ← x * y
```



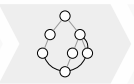
RQ1: Data-Flow



Parsing



Normalization

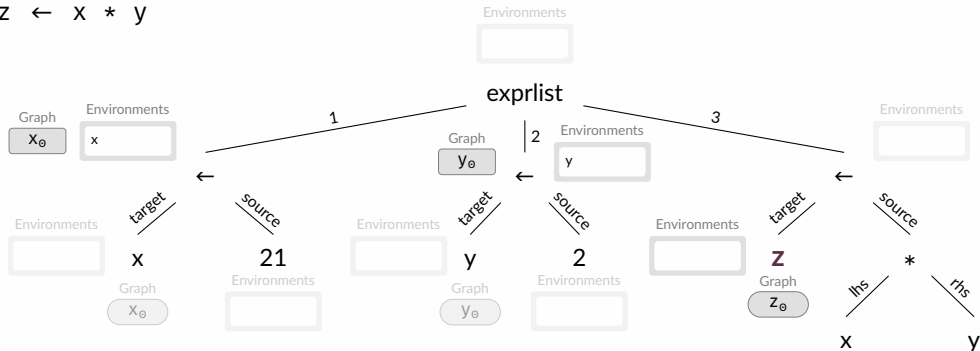


Data-Flow



Slicing

```
x ← 21
y ← 2
z ← x * y
```



RQ1: Data-Flow



Parsing



Normalization

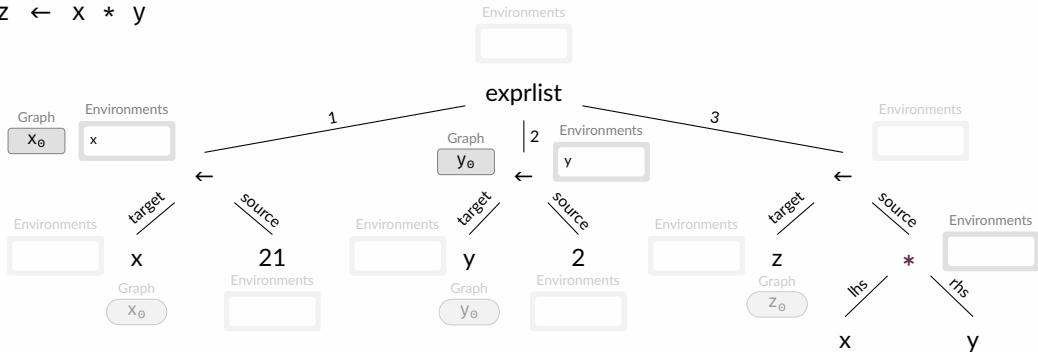


Data-Flow



Slicing

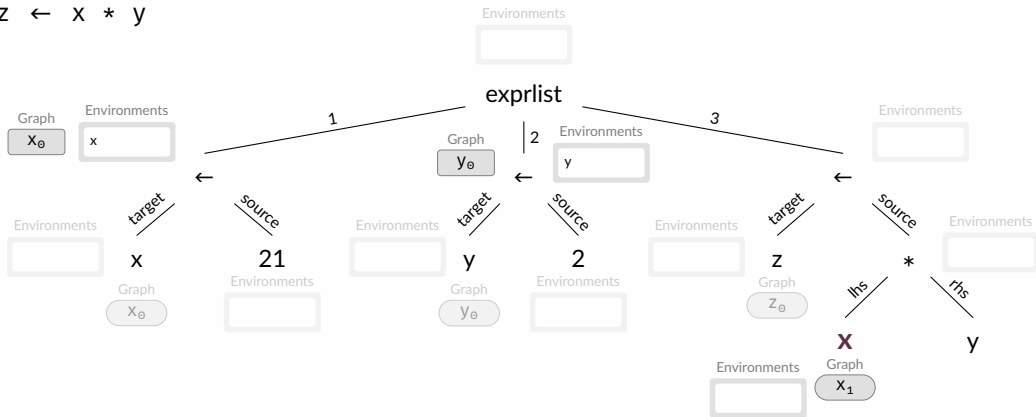
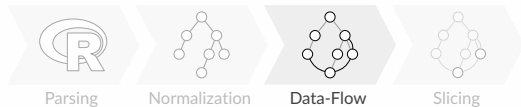
```
x ← 21
y ← 2
z ← x * y
```



RQ1: Data-Flow

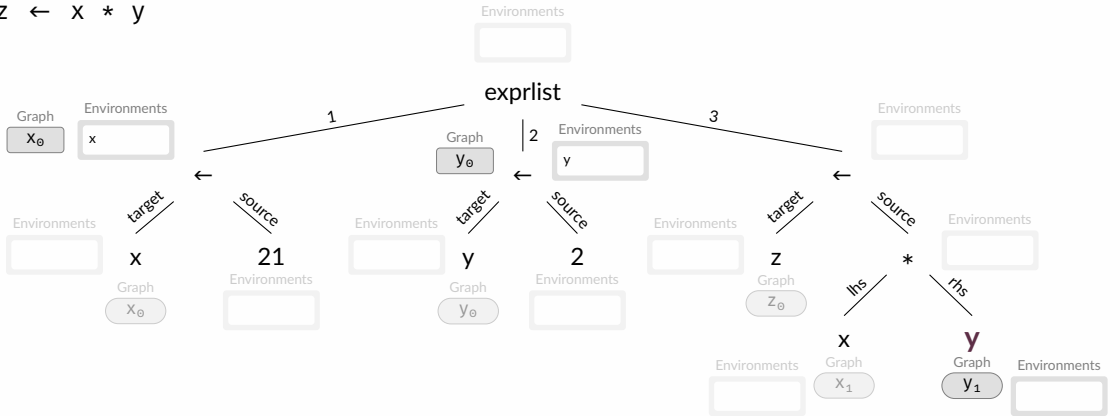
```

x ← 21
y ← 2
z ← x * y
    
```



RQ1: Data-Flow

```
X ← 21
y ← 2
Z ← X * y
```



RQ1: Data-Flow

```
x ← 21
y ← 2
z ← x * y
```



Parsing



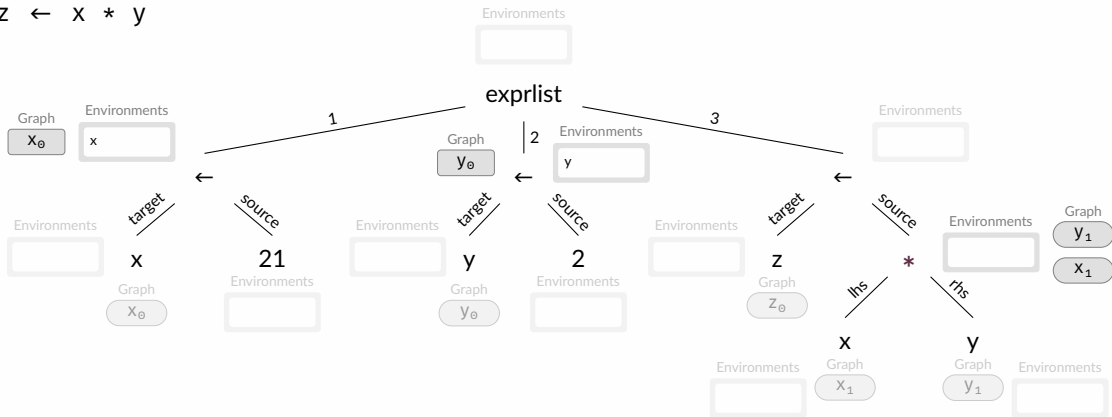
Normalization



Data-Flow

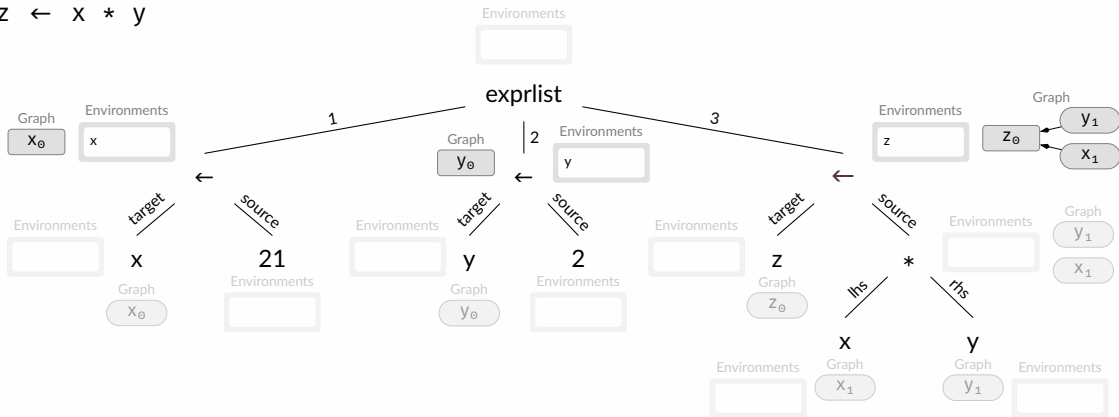
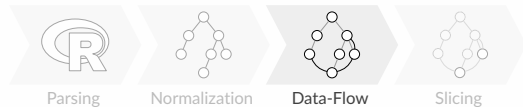


Slicing



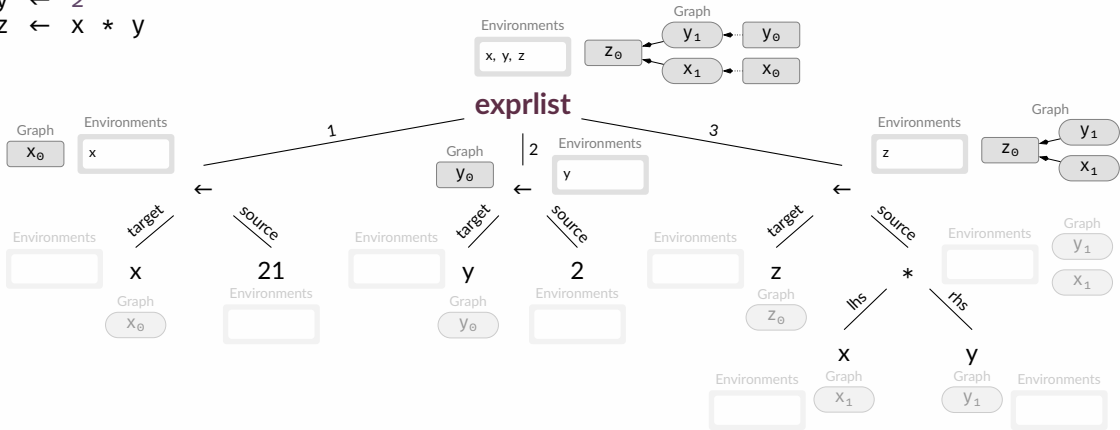
RQ1: Data-Flow

```
x ← 21
y ← 2
z ← x * y
```

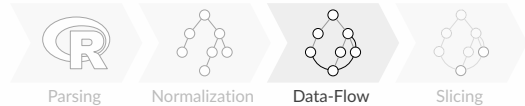


RQ1: Data-Flow

```
x ← 21
y ← 2
z ← x * y
```



RQ2: Features



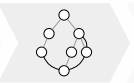
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

- There are many ways to modify data in R, like:

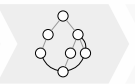
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

➤ There are many ways to modify data in R, like:

- $a \leftarrow 1$, $a \llcorner 1$, $a = 1$, $1 \rightarrow a$, $1 \twoheadrightarrow a$

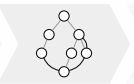
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

› There are many ways to modify data in R, like:

- $a \leftarrow 1$, $a \llcorner 1$, $a = 1$, $1 \rightarrow a$, $1 \twoheadrightarrow a$
- **assign**("a", 1), $b \leftarrow \text{"a"}$; **assign**(b, 1)

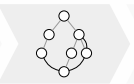
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

› There are many ways to modify data in R, like:

- $a \leftarrow 1$, $a \llcorner 1$, $a = 1$, $1 \rightarrow a$, $1 \twoheadrightarrow a$
- **assign**("a", 1), $b \leftarrow \text{"a"}$; **assign**(b, 1)
- **setGeneric**("props", **function**(object) object)

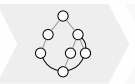
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

- There are many ways to modify data in R, like:
 - $a \leftarrow 1$, $a \llcorner 1$, $a = 1$, $1 \rightarrow a$, $1 \twoheadrightarrow a$
 - **assign**("a", 1), $b \leftarrow \text{"a"};$ **assign**(b, 1)
 - **setGeneric**("props", **function**(object) object)
- Environments can be changed manually

RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

- 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)

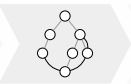
RQ2: Features



Parsing



Normalization



Data-Flow



Slicing

- 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

RQ2: Features, II

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

RQ2: Features, II

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

UserRs

published scripts in social science
4230 files

RQ2: Features, II

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

UserRs

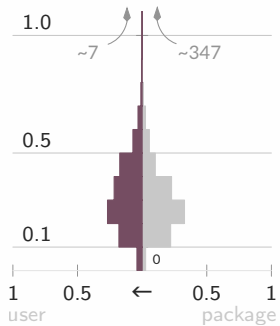
published scripts in social science
4230 files

Package Authors

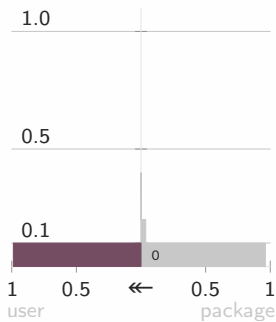
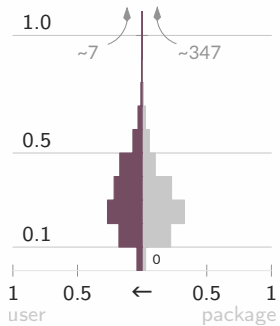
top 500 CRAN packages
25 691 files

RQ2: Features, III

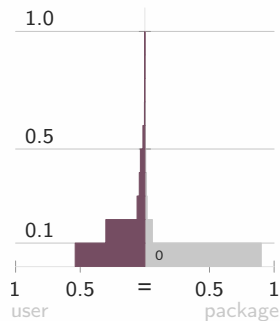
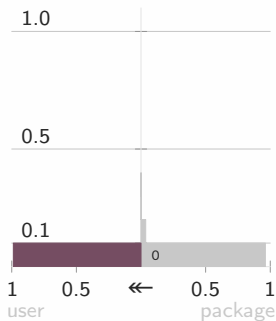
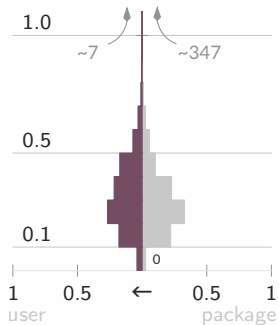
RQ2: Features, III



RQ2: Features, III



RQ2: Features, III



RQ2: Features, IV

RQ2: Features, IV

- › Most used packages allow to prioritize special support

RQ2: Features, IV

- › Most used packages allow to prioritize special support
- › `setGeneric`, `assign`,... used very often in package code

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.

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

RQ2: Features, IV

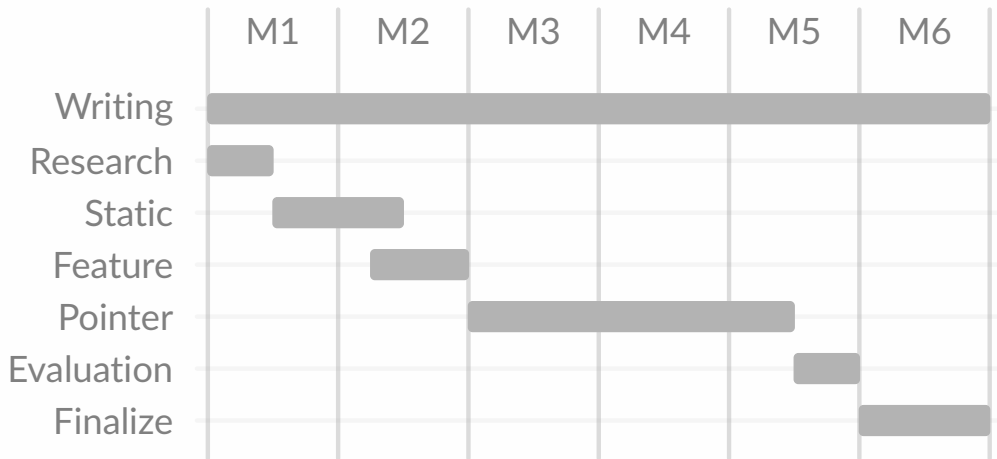
- › Most used packages allow to prioritize special support
- › `setGeneric`, `assign`,... used very often in package code
 - Namespaces are very rarely manipulated, we do not plan supporting that.
- › Data types are mostly accessed by name which allows for pointer analysis
- › `.C` and `.Fortran` are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

The Plan

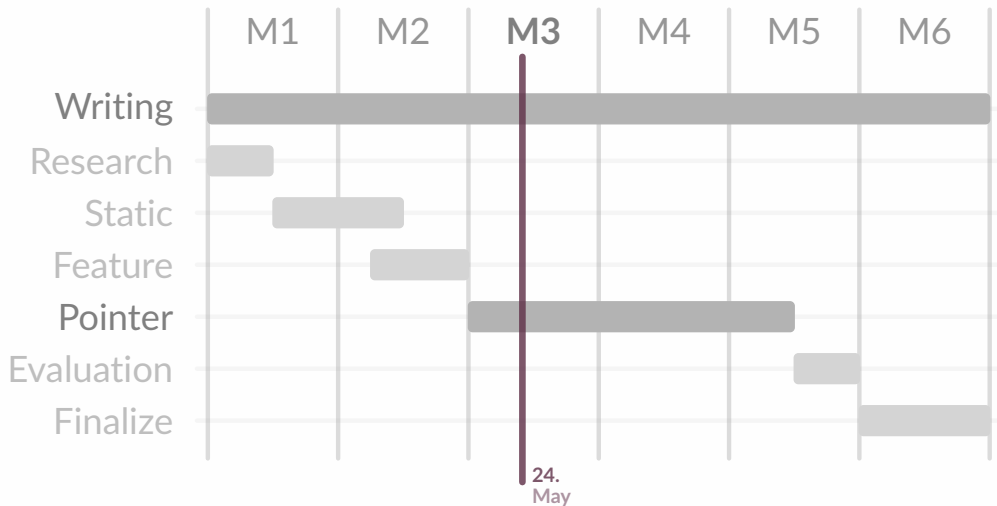
The Plan

	M1	M2	M3	M4	M5	M6
Writing						
Research						
Static						
Feature						
Pointer						
Evaluation						
Finalize						

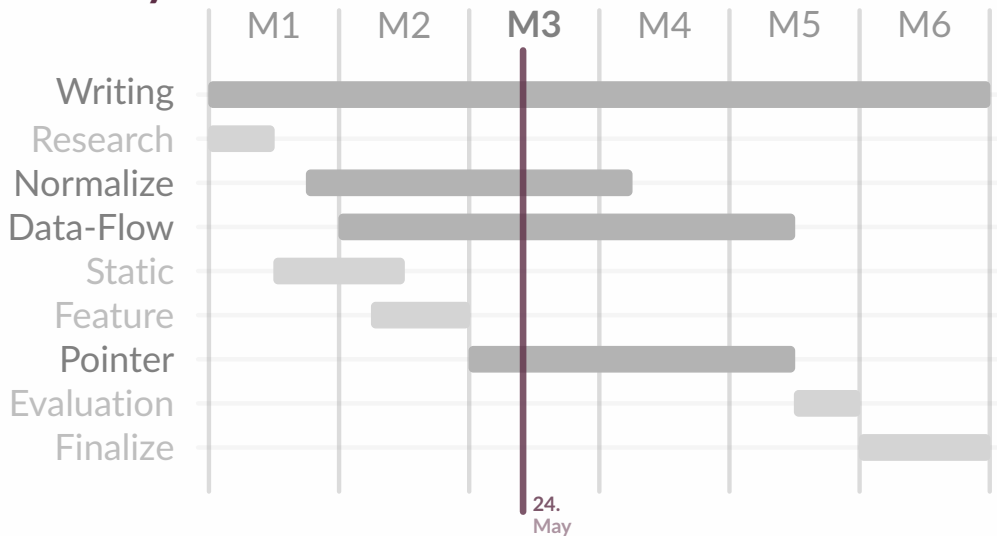
The Plan



The Plan



The Reality



Overview

Overview

<pre>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")</pre>	<p>slice(10, sum)</p> <p>→</p>	<pre>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")</pre>
--	--------------------------------	--

Goal

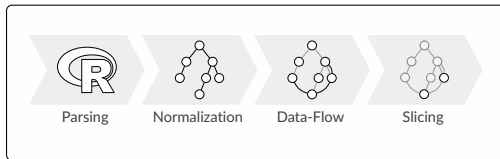
Overview

```
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

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")
```

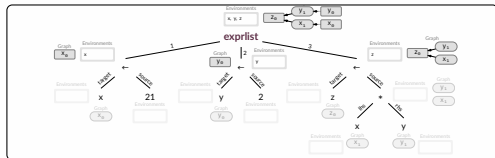
→ slice(10, sum) →

```
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



Data-Flow



Parsing



Normalization



Data-Flow



Slicing

Program

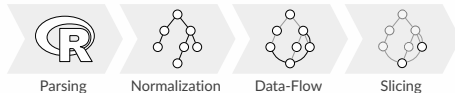
Overview

```
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")
```

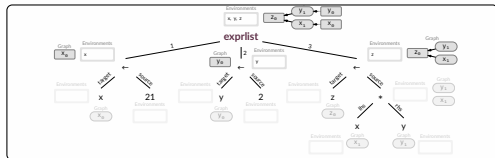
slice(10, sum) →

```
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")
```

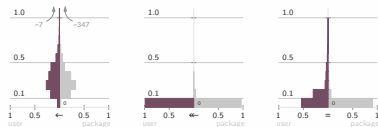
Goal



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.

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:

In package code:

Used Packages

In UseR scripts:

In package code:

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:

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 statistical functions
2. utils basic programming functions
3. rlang working with types
4. withr temporarily 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

Example Dataflow

```
a ← 3  
a ← x * m  
  
if(m > 3) {  
  a ← 5  
}  
  
b ← a + c
```

Example Dataflow

```
a0 ← 3  
a1 ← x0 * m0  
  
if(m1 > 3) {  
    a2 ← 5  
}  
  
b0 ← a3 + c0
```

Example Dataflow

a_0

```
>  $a_0 \leftarrow 3$   
    $a_1 \leftarrow x_0 * m_0$ 
```

```
if( $m_1 > 3$ ) {  
     $a_2 \leftarrow 5$   
}
```

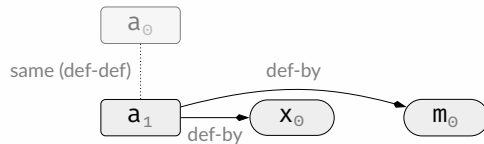
```
 $b_0 \leftarrow a_3 + c_0$ 
```


Example Dataflow

```
a0 ← 3  
> a1 ← x0 * m0
```

```
if(m1 > 3) {  
    a2 ← 5  
}
```

```
b0 ← a3 + c0
```

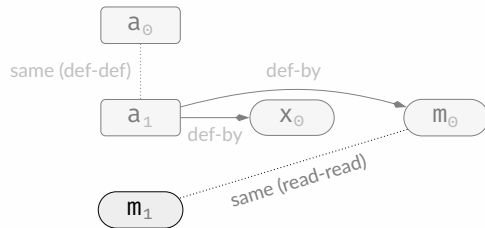


Example Dataflow

```
a0 ← 3  
a1 ← x0 * m0
```

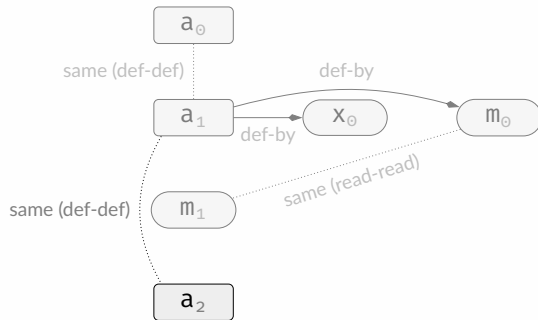
```
> if(m1 > 3) {  
    a2 ← 5  
}
```

```
b0 ← a3 + c0
```



Example Dataflow

```
a0 ← 3  
a1 ← x0 * m0  
  
if(m1 > 3) {  
>   a2 ← 5  
}  
  
b0 ← a3 + c0
```

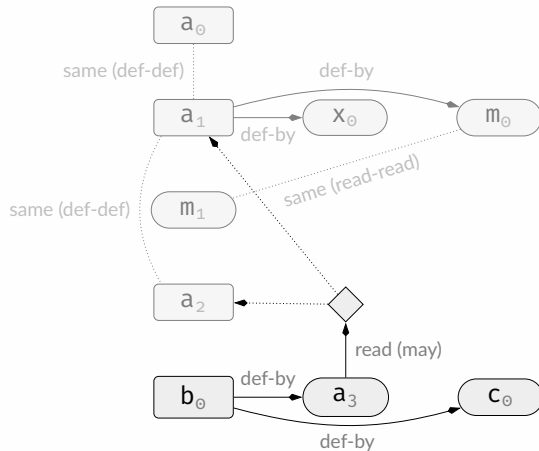


Example Dataflow

```
a0 ← 3  
a1 ← x0 * m0
```

```
if(m1 > 3) {  
  a2 ← 5  
}
```

```
> b0 ← a3 + c0
```

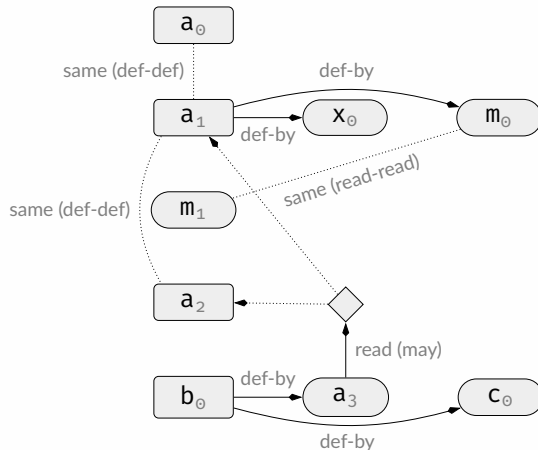


Example Dataflow

```
a0 ← 3  
a1 ← x0 * m0
```

```
if(m1 > 3) {  
  a2 ← 5  
}
```

```
b0 ← a3 + c0
```



Modifying Environments and Functions

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

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

Modifying Assignments

Modifying Assignments

```
f ← function(x) { body(f)[[2]] ← 3 }  
f(2)
```


Modifying Assignments

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

Modifying Assignments

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

Modifying Assignments

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

```
f ← function(x) a + b  
f(2)
```

Modifying Assignments

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

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

Modifying Assignments

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

```
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