# Towards Automated Analysis Of R Programs

Anton Xue<sup>1</sup>, Ross Mawhorter<sup>2</sup>, Gian Pietro Farina<sup>3</sup>, Stephen Chong<sup>4</sup>

1 Yale University, <sup>2</sup>Harvey Mudd, <sup>3</sup>University at Buffalo, <sup>4</sup>Harvard University

### **Motivation**

- ★ The R statistical and graphical language is popular
  - ★ 200k+ public GitHub repositories
- \* R supports many potentially **hazardous** features
  - \* Lazy side effects, first-order environments, dynamic typing
- \* Many users lack computer science background
  - \* May not understand features and write **erroneous** code
- \* R has little formal methods support

# **Objectives**

- ★ Formalize a subset of R (Simple-R)
  - ★ Leverage previous efforts: Morandat et al. [ECOOP12]
- ★ Evaluate Simple-R can capture real-world programs
  - ⋆ Use a corpus of code from Harvard's Dataverse Repository
- ⋆ Develop analysis tooling as a proof of concept
  - \* Symbolic execution as a first step

# The R Language And Its Nuances

### R's Language Features

- \* Interpreted language with dynamic typing
- ★ Basic data types are vectors, environments, and functions
- \* Imperative, functional, object-oriented
  - \* Function arguments are lazy
  - \* 3 types of object-orientation: S3, S4, reference classes
- \* Allows manipulation of environments as first-class objects
- \* Allows metaprogramming through eval and parse

### R Language Interpreter Implementation

- \* R is an open source: <a href="https://github.com/wch/r-source">https://github.com/wch/r-source</a>
- ★ Complicated grammar but simple internal core language
- \* All data eventually lives on the heap
- ★ Very little optimization in the interpreter

# Sample R Programs

\* Many strange programs can be written in R!

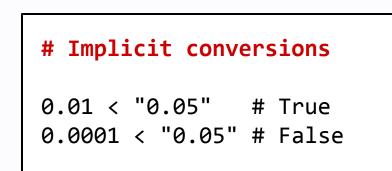
```
# `rm` deletes variables

x <- 5
rm(x)
print(x) # Error</pre>
```

```
# Super assign changes parent scope

x <- 5
foo <- function () { x <<- 6 }
foo()
print(x) # Prints 6 instead of 5</pre>
```

```
# `eval` allows for metaprogramming
eval(parse(text="x<-5"));
print(x) # Prints 5!</pre>
```



- ★ Such programs are hard to analyze
  - ⋆ Can statically detect some of these features

# A Simple(-R) Formalization Of R

#### Simple-R Language Syntax

★ Reduced version of R's syntax

c	:=	num   bool   str   NA		
p	:=	$x \mid x = e \mid \mathtt{VarParam}$		
a	:=	$e \mid x = e \mid \mathtt{VarArg}$		
e	:=	$x \mid c \mid \lambda ar{p}$ . $e \mid e \ (ar{a}) \mid$ Return $e$		
		$\bar{e} \mid x \leftarrow e \mid \texttt{if} \ (e_1) \ \texttt{then} \ e_2 \ \texttt{else} \ e_3$		
		$e[[\bar{a}]] \mid e[[\bar{a}]] \leftarrow e$		
		$\mathtt{attr}(e,x) \mid \mathtt{attr}(e,x) \leftarrow e$		
		while $(e_1)$ do $e_2$   Break   Next		

- ⋆ Differences with R:
  - ⋆ Some de-sugaring (e.g. no for-loops)
  - ⋆ No "super assignments" (direct environment manipulation)
  - ⋆ Permits only S3-style object orientation

### **Simple-R Execution Semantics**

★ Can be modeled as CEK-style lambda calculus interpreters

```
S := (R, K, H)

R := e \mid m

K := \emptyset \mid C : K

H := \{m \mapsto B\}

B := \overline{c} \mid \{x \mapsto m\} \mid (\lambda \overline{x}.e, m)
```

#### Execution reduction rules:

- 1. Reduce each expression (redex R) to normal form
- 2. Allocate object (B) on heap (H).
- 3. Yield its pointer / memory (m) to the redex a. Objects may be a:
  - i. Vector (array of constants or pointers)
  - ii. Environment (map variable to pointer)
- iii. Closure (lambda with environment pointer)
  4. Check next continuation (C) on stack (K)
- 5. Repeat until continuation (c) on Stack (k)
- ⋆ Differences with R:
  - \* Restrictions on primitive functions
    - \* No environment manipulation or metaprogramming
  - ★ Strict evaluation of function arguments instead of lazy
    - ⋆ Detecting when doing this preserves semantics is hard!

### **Evaluation**

- ⋆ Files scraped from Harvard Dataverse Repository
  - \* R code used in actual research

Feature Usage	124 Files
Super assignments	0 (0%)
Non-S3 object orientation	0 (0%)
Uses environments	1 (0.81%)
Makes a rm call	59 (47.58%)
Out of place rm call	16 (12.90%)
Uses eval call	1 (0.81%)
Uses external library	94 (75.81%)

- ★ Most of the dangerous features were not used
- \* Many programs make rm calls (deletes variable definition)
  - ★ But 72.88% of these appear as the first or last expression
- ⋆ Most programs do use external libraries

# **Towards Symbolic Execution**

- ★ Write custom parser to parse R programs into Simple-R
- \* Preprocessor to lint, reject, or transform Simple-R programs
- \* Run Simple-R execution semantics
  - \* Augmented with semantics to handle symbolic variables
- ★ Call solvers when branching on symbolic variables
- \* Prototype: <a href="https://github.com/aremath/core-r">https://github.com/aremath/core-r</a>

### **Conclusion And Future Work**

- \* Formalized syntax and semantics of a subset of R (Simple-R)
- \* Evaluation of how Simple-R captures real-world R programs
- Preliminary work towards analysis tools for R
- \* TODO: develop symbolic execution, gradual typing system