Analyzing R

On the Anatomy of Real-World R Code for Static Analysis

At SE ’25

Ulm University | MSR ’24 | **Florian Sihler**, Pietzschmann, Straub, Tichy, Diera, Dahou | February, 2025

Software Engineering
Programming Languages

universität ulm

The R Programming Language

Is Used
in Research[1]

Is Used
for Statistical Computing[2]

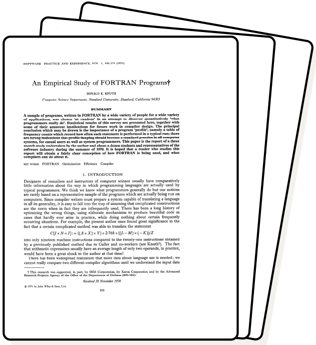
Is Used (mostly)
by Non-Programmers

- ≈70 % of scripts are not reproducible[1]
- Lacks sophisticated static-analysis tools
- *Many* powerful reflective capabilities[3]
- Incomplete language specification[4]

Which features are actually used?

[1] Trisovic et al., “A Large-Scale Study on Research Code Quality and Execution” (*Sci Data ’22*)
[2] https://cran.r-project.org/
[3] Flückiger et al., “R melts brains: an IR for first-class environments and lazy effectful arguments” (*DLS ’19*)
[4] R Core Team, *R Language Definition*

Related Research



SOFTWARE PRACTICE AND EXPERIENCE, VOL. 1, 105-133 (1971)

An Empirical Study of FORTRAN Programs†

DONALD E. KNUTH

Computer Science Department, Stanford University, Stanford, California 94305

SUMMARY

A sample of programs, written in FORTRAN by a wide variety of people for a wide variety of applications, was chosen 'at random' in an attempt to discover quantitatively 'what programmers really do'. Statistical results of this survey are presented here, together with some of their apparent implications for future work in compiler design. The principal conclusion which may be drawn is the importance of a program 'profile', namely a table of frequency counts which record how often each statement is performed in a typical run; there are strong indications that profile-keeping should become a standard practice in all computer systems, for casual users as well as system programmers. This paper is the report of a three month study undertaken by the author and about a dozen students and representatives of the software industry during the summer of 1970. It is hoped that a reader who studies this report will obtain a fairly clear conception of how FORTRAN is being used, and what compilers can do about it.

KEY WORDS FORTRAN Optimization Efficiency Compiler

1. INTRODUCTION

Designers of compilers and instructors of computer science usually have comparatively little information about the way in which programming languages are actually used by typical programmers. We think we know what programmers generally do but our notions are rarely based on a representative sample of the programs which are actually being run on computers. Since compiler writers must prepare a system capable of translating a language in all its generality, it is easy to fall into the trap of assuming that complicated constructions are the norm when in fact they are infrequently used. There has been a long history of optimizing the wrong things, using elaborate mechanisms to produce beautiful code in cases that hardly ever arise in practice, while doing nothing about certain frequently occurring situations. For example, the present author once found great significance in the fact that a certain complicated method was able to translate the statement

$$C[I\times N+J]:=((A+X)\times Y)+2{\cdot}768+((L-M)\times(-K))/Z$$

into only nineteen machine instructions compared to the twenty-one instructions obtained by a previously published method due to Galler and co-workers (see Knuth[1]). The fact that arithmetic expressions usually have an average length of only two operands, in practice, would have been a great shock to the author at that time!

There has been widespread realization that more data about language use is needed; we cannot really compare two different compiler algorithms until we understand the input data

† This research was supported, in part, by IBM Corporation, by Xerox Corporation and by the Advanced Research Projects Agency of the Office of the Department of Defense (SD-183).

Received 20 November 1970

© 1971 by John Wiley & Sons, Ltd.

105

[5] Optimization

Fortran

The Journal of Systems and Software 123 (2017) 160–172

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

ELSEVIER

Understanding the syntactic rule usage in java

CrossMark

Dong Qiu[a], Bixin Li[a,*], Earl T. Barr[b], Zhendong Su[c]

[a] School of Computer Science and Engineering, Southeast University, China
[b] Department of Computer Science, University College London, UK
[c] Department of Computer Science, University of California Davis, USA

ARTICLE INFO

Article history:
Received 1 February 2016
Revised 26 September 2016
Accepted 19 October 2016
Available online 20 October 2016

Keywords:
Language syntax
Empirical study
Practical language usage

ABSTRACT

Context: Syntax is fundamental to any programming language: syntax defines valid programs. In the 1970s, computer scientists rigorously and empirically studied programming languages to guide and inform language design. Since then, language design has been artistic, driven by the aesthetic concerns and intuitions of language architects. Despite recent studies on small sets of selected language features, we lack a comprehensive, quantitative, empirical analysis of how modern, real-world source code exercises the syntax of its programming language.

Objective: This study aims to understand how programming language syntax is employed in actual development and explore their potential applications based on the results of syntax usage analysis.

Method: We present our results on the first such study on Java, a modern, mature, and widely-used programming language. Our corpus contains over 5000 open-source Java projects, totalling 150 million source lines of code (SLoC). We study both independent (*i.e.* applications of a single syntax rule) and dependent (*i.e.* applications of multiple syntax rules) rule usage, and quantify their impact over time and project size.

Results: Our study provides detailed quantitative information and yields insight, particularly (i) confirming the conventional wisdom that the usage of syntax rules is Zipfian; (ii) showing that the adoption of new rules and their impact on the usage of pre-existing rules vary significantly over time; and (iii) showing that rule usage is highly contextual.

Conclusions: Our findings suggest potential applications across language design, code suggestion and completion, automatic syntactic sugaring, and language restriction.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Syntax and semantics define a programming language. Informally, a language has many features. A language's syntactic rules provide the most direct means to measure the use of a language's features. Thousands of programming languages exist; each embodies a different set of possible language features. Language designers usually have limited knowledge on how programmers actually use a language (Knuth, 1971). This leads to many unnatural and rarely used features being introduced, while expected ones not introduced (Strangest language feature, 2016; Your language sucks, 2016). In addition, many language features, especially language syntax, remain a significant barrier to novice programmers (Denny et al., 2011; Stefik and Siebert, 2013).

We tackle the question of how to systematically understand these features and their usage. Rather than ad-hoc characterizations of features, we propose the use of language grammars to precisely and systematically characterize language features. Indeed, most programming language features quite directly map onto syntactic constructs. Therefore, we study how programmers use language features by analyzing their use of the language syntax.

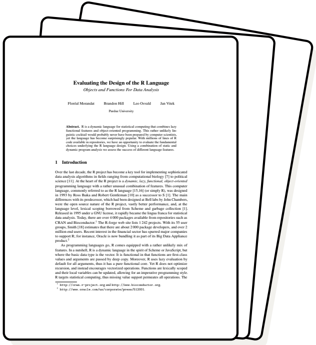
Knuth conducted the first study to understand how programmers use FORTRAN over 40 years ago (Knuth, 1971). Similar studies were subsequently performed on COBOL (Salvadori et al., 1975; Chevance and Heidet, 1978), APL (Saal and Weiss, 1977) and Pascal (Cook and Lee, 1982) between the 1970s and 1980s. In recent decades, there has been little quantitative study demonstrating how a modern programming language is used in practice, especially from the perspective of language syntax. Previous studies have investigated the use of subsets of language features (*e.g.*, Java generics (Parnin et al., 2011) and Java reflection (Livshits et al., 2005)). Although Dyer et al. (Dyer et al., 2014) investigated the use

* Corresponding author. Fax: 86 25 52090879.
E-mail addresses: dongqiu@seu.edu.cn (D. Qiu), bx.li@seu.edu.cn (B. Li), e.barr@ucl.ac.uk (E.T. Barr), su@cs.ucdavis.edu (Z. Su).

http://dx.doi.org/10.1016/j.jss.2016.10.017
0164-1212/© 2016 Elsevier Inc. All rights reserved.

[6] Syntactic Rules

Java



Evaluating the Design of the R Language

Objects and Functions For Data Analysis

Floréal Morandat Brandon Hill Leo Osvald Jan Vitek

Purdue University

Abstract. R is a dynamic language for statistical computing that combines lazy functional features and object-oriented programming. This rather unlikely linguistic cocktail would probably never have been prepared by computer scientists, yet the language has become surprisingly popular. With millions of lines of R code available in repositories, we have an opportunity to evaluate the fundamental choices underlying the R language design. Using a combination of static and dynamic program analysis we assess the success of different language features.

1 Introduction

Over the last decade, the R project has become a key tool for implementing sophisticated data analysis algorithms in fields ranging from computational biology [7] to political science [11]. At the heart of the R project is a *dynamic, lazy, functional, object-oriented* programming language with a rather unusual combination of features. This computer language, commonly referred to as the R language [15,16] (or simply R), was designed in 1993 by Ross Ihaka and Robert Gentleman [10] as a successor to S [1]. The main differences with its predecessor, which had been designed at Bell labs by John Chambers, were the open source nature of the R project, vastly better performance, and, at the language level, lexical scoping borrowed from Scheme and garbage collection [1]. Released in 1995 under a GNU license, it rapidly became the lingua franca for statistical data analysis. Today, there are over 4 000 packages available from repositories such as CRAN and Bioconductor.[1] The R-forge web site lists 1 242 projects. With its 55 user groups, Smith [18] estimates that there are about 2 000 package developers, and over 2 million end users. Recent interest in the financial sector has spurred major companies to support R; for instance, Oracle is now bundling it as part of its Big Data Appliance product.[2]

As programming languages go, R comes equipped with a rather unlikely mix of features. In a nutshell, R is a dynamic language in the spirit of Scheme or JavaScript, but where the basic data type is the vector. It is functional in that functions are first-class values and arguments are passed by deep copy. Moreover, R uses lazy evaluation by default for all arguments, thus it has a pure functional core. Yet R does not optimize recursion, and instead encourages vectorized operations. Functions are lexically scoped and their local variables can be updated, allowing for an imperative programming style. R targets statistical computing, thus missing value support permeates all operations. The

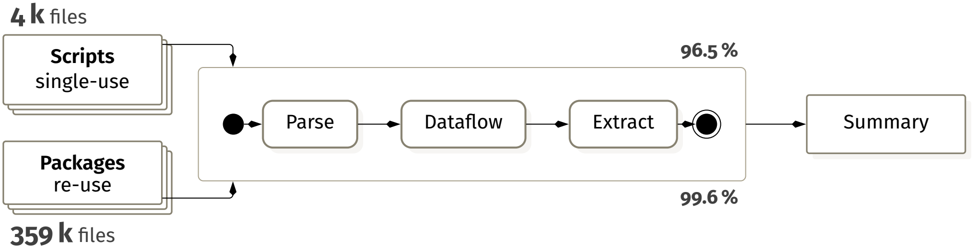
[1] http://cran.r-project.org and http://www.bioconductor.org.
[2] http://www.oracle.com/us/corporate/press/512001.

[7] Dynamic Usage

R

[7] Morandat et al., "Evaluating the design of the R language: Objects and functions for data analysis" (*ECOOP '12*)
[6] Qiu et al., "Understanding the Syntactic Rule Usage in Java" (*JSS '17*)
[5] Knuth, "An Empirical Study of FORTRAN Programs" (*Software: Practice and Experience '71*)

Extraction Workflow



RQ 1: Frequent Features

RQ 2: Differences in Research Scripts and Packages

RQ 3: Insights for Static Analysis

Overview

4.1
Processing Errors

4.2
Metadata

4.3
Assignments and Access

4.4
Conditionals

4.5
Loops

4.6
Function Definitions

4.7
Function Calls

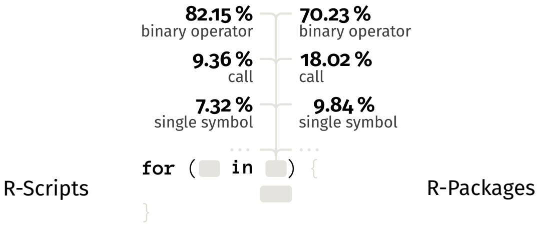
4.8
Packages

Assignments [4.3]

- ←, ↞, =, ↠, →, assign, delayedAssign, ...
- 41 % of scripts mix ← and =
- Assignment functions are rare, but more common in packages

Loops [4.5]

```
for (i in 1:10) {
  print(i)
}
```



R-Scripts

82.15 % binary operator

9.36 % call

7.32 % single symbol

R-Packages

70.23 % binary operator

18.02 % call

9.84 % single symbol

```
for ( in ) {

}
```

- Most loops have a for-i form
- Scripts contain on average 3 times as many loops

Meta-Programming [4.7.1]

Evaluate Strings

```
eval(parse(text=
  paste0("v",1," ← 42")
)) # v1 ← 42
```

Modify Functions

```
f ← function(a, b) a
body(f) ← quote(b)
f(1, 2) # 2
```

Redefine “Keywords”

```
‘for‘ ← \(...) "hi"
for(i in 1:10) x ← i
# "hi"
```

Store/Load Environment

```
save.image(file="env")
# ...
load("env")
```

Meta-Programming [4.7.1]

Evaluate Strings

```
eval(parse(text=
  paste0("v",1," ← 42")
)) # v1 ← 42
```

1 % of scripts

3 % of packages

Modify Functions

```
f ← function(a, b) a
body(f) ← quote(b)
f(1, 2) # 2
```

Effectively unused

Redefine “Keywords”

```
‘for‘ ← \(...) "hi"
for(i in 1:10) x ← i
# "hi"
```

Effectively unused

Store/Load Environment

```
save.image(file="env")
# ...
load("env")
```

12 % of scripts

0.8 % of packages

Study Results

RQ1
Frequent Features

+ Only 2 of all assignment operators
– Reflective functions
– No tests/checks in scripts

RQ2
Differences

- Scripts are longer
- Scripts prefer (for-)loops

RQ3
Insights

- Extensions for `{lintr}`
- No focus on reflective functions required
- Error-tolerant parsing

R has many features

only a few are used frequently

Appendix

References I

[1] Ana Trisovic et al. “A Large-Scale Study on Research Code Quality and Execution”. 2022

[2] *The Comprehensive R Archive Network — cran.r-project.org*. 2024

[3] Olivier Flückiger et al. “R melts brains: an IR for first-class environments and lazy effectful arguments”. 2019

[4] R Core Team. *R Language Definition*. 2024

[5] Donald E. Knuth. “An Empirical Study of FORTRAN Programs”. 1971

[6] Dong Qiu et al. “Understanding the Syntactic Rule Usage in Java”. Jan. 1, 2017

[7] Floréal Morandat et al. “Evaluating the design of the R language: Objects and functions for data analysis”. 2012