# Seminário - CCF 340: Linguagens de Programação

# Artigo

Promises Are Made to Be Broken

Migrating R to Strict Semantics

Promessas são feitas para serem quebradas

Migrando R para semântica estrita

Aviral Goel, Jan Ječmen, Sebastián Krynski, Olivier Flückiger, Jan Vitek

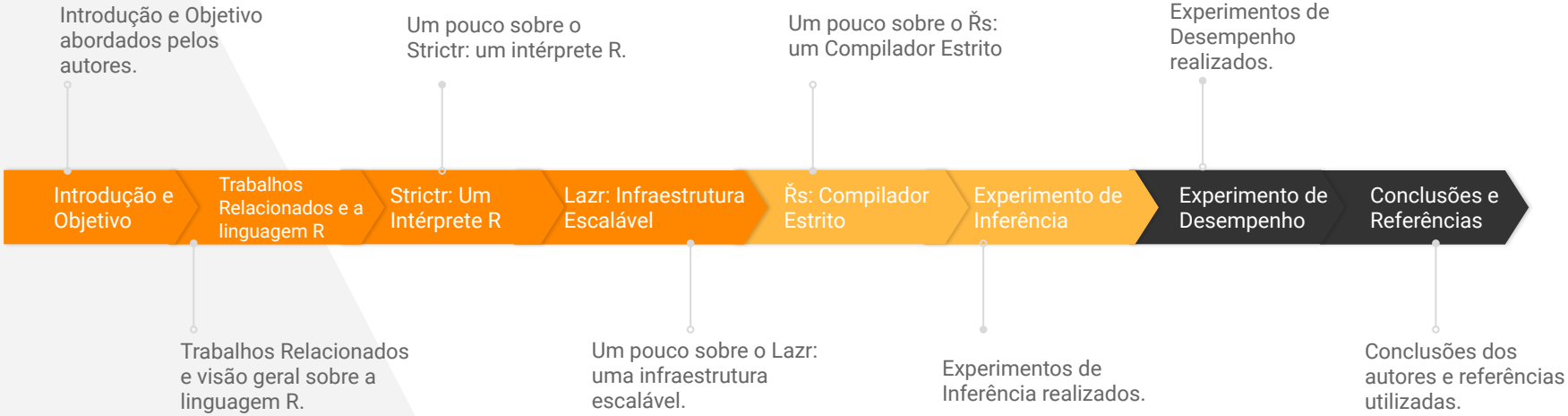
# Grupo

**Igor Lucas**
3865

**Lázaro**
3861

**Pedro Carvalho**
3877

# Sumário


Introdução e Objetivo abordados pelos autores.
Um pouco sobre o Strictr: um intérprete R.
Um pouco sobre o Řs: um Compilador Estrito
Experimentos de Desempenho realizados.
Introdução e Objetivo
Trabalhos Relacionados e a linguagem R
Strictr: Um Intérprete R
Lazr: Infraestrutura Escalável
Řs: Compilador Estrito
Experimento de Inferência
Experimento de Desempenho
Conclusões e Referências
Trabalhos Relacionados e visão geral sobre a linguagem R.
Um pouco sobre o Lazr: uma infraestrutura escalável.
Experimentos de Inferência realizados.
Conclusões dos autores e referências utilizadas.

# 1. Introdução e Objetivo

# 1. Introdução

- A linguagem de programação R é amplamente utilizada em ciência de dados.
- Muitos usuários não sabem que chamadas de função têm semântica lenta
- Goel e Vitek [2019] forneceram um estudo observacional completo de pacotes R.
- Argumenta que a preguiça deve ser a exceção em R.
- Com isso, este trabalho tem como objetivo avaliar os seguintes componentes deste caminho de migração:
- Đ StrictR, Đ LazR:, Đ Řs.

# 2. Trabalhos Relacionados e a linguagem R

# 2. Linguagem R

## 2.1 - Trabalhos relacionados:

- Existem três grupos de resultados:
  - Call by need
  - The R language.
  - Language migration.

# 2. Linguagem R

## 2.2 - Preguiça na linguagem R

- A linguagem R é amplamente utilizada em ciência de dados.
  - Funções
  - Reflection
  - Effects
  - Avaliação atrasado.

# 3. Strictr: Um Intérprete R

# 3. Strictr

- É uma implementação protótipo de R com uma semântica estrita por padrão e assinaturas de rigidez para especificar quais argumentos devem permanecer preguiçosos.
  - Strictness Signatures.
    - sig ::= strict ‘pack :: fun‘ ⟨*i*1,*i*2,*i*3, . . . ⟩
  - Execução
  - Preguiça acidental
  - Ordem de avaliação

# 4. Lazr: Infraestrutura Escalável

# 4. Lazr

- LazR é um pipeline para inferir assinaturas de rigidez para bibliotecas R legados em escala.

```
popular <- function(a,b,c,d,e,f) {
   if (a) b
   print(substitute(c))
   if(!missing(f)) print(f)
   return(e+d)
}
```

*Library*

```
r1<-popular(FALSE,print('Hi'),3,4,5)
r2<-popular(TRUE,1+2,stop(),0,9)
r3<-popular(TRUE,1+2,3,r1<-4,r1+1)
```

*Client code*

# 4. Lazr

Inferência:

- V: Um parâmetro vararg;
- M: Um parâmetro passado para substituir ou PREXPR é metaprogramado
- G: Um parâmetro que não recebeu argumento em nenhuma invocação de função
- U:  Um parâmetro que permanece não avaliado;
- S: Um argumento com efeitos colaterais;
- R: Um argumento que usa reflexão para observar o estado da pilha de chamadas

# 5. Řs: Compilador Estrito

# A escolha do compilador

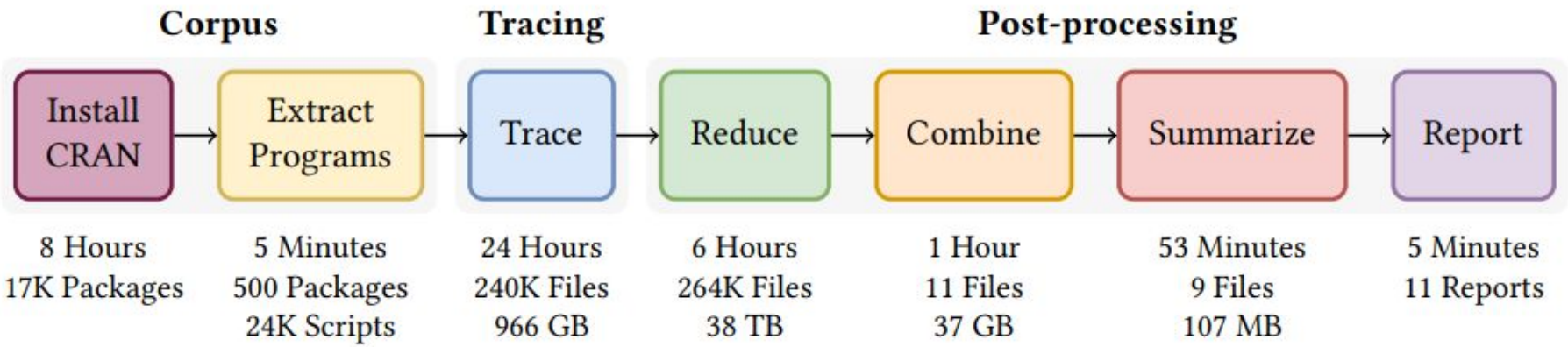
- Abordagens -> Performance
  - Mudança da semântica;
  - Compilador nativo
  - Truffle ou Řs
- Escolha do Řs
- Análise de performance

# 6. Experimento de Inferência

# Objetivo

- Objetivo
  - Preguiça (*laziness*) em código legado
  - Robustez do código de inferência
- LazR
  - Escalável
  - CRAN: quantidade de dados -> TB
  - Estilo simples de mapa de redução

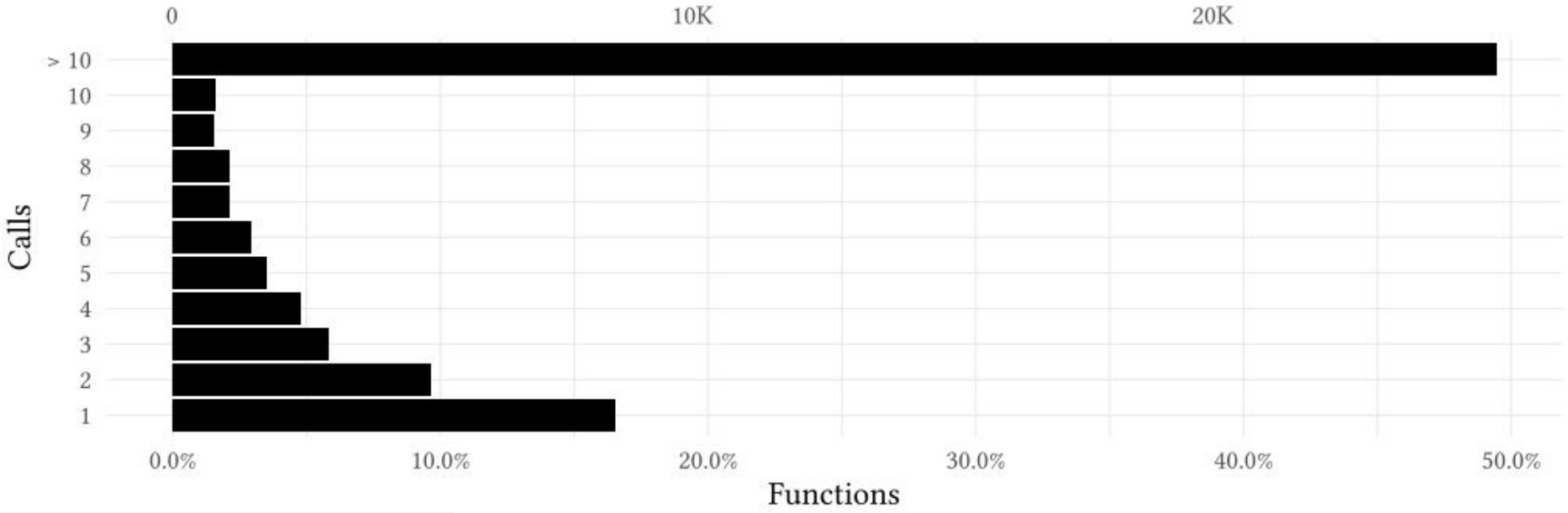
# Análise


Corpus
Tracing
Post-processing
Install CRAN
Extract Programs
Trace
Reduce
Combine
Summarize
Report
8 Hours
17K Packages
5 Minutes
500 Packages
24K Scripts
24 Hours
240K Files
966 GB
6 Hours
264K Files
38 TB
1 Hour
11 Files
37 GB
53 Minutes
9 Files
107 MB
5 Minutes
11 Reports

# Corpus

- Corpo
  - 500 pacotes
  - CRAN
  - *spatstat.geom*
- spatstat.geom
  - 889 funções
  - 130M de chamadas

# Corpus


0
10K
20K
> 10
10
9
8
7
6
5
4
3
2
1
Calls
0.0%
10.0%
20.0%
30.0%
40.0%
50.0%
Functions

# Assinaturas de rigidez inferida

Table 2. Signature Summary

| V | M | G | U | S | R | Parameters | | Functions | | Packages |
|---|---|---|---|---|---|---|---|---|---|---|
| ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | 20.0K | 9.8% | 20.0K | 38.8% | 430 |
| ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | 1.3K | 0.6% | 825 | 1.6% | 118 |
| ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | 3.2K | 1.6% | 1.7K | 3.2% | 240 |
| ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | 148.4K | 72.9% | 49.3K | 95.7% | 489 |
| ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | 529 | 0.3% | 509 | 1% | 119 |
| ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | 1.3K | 0.6% | 950 | 1.8% | 207 |
| ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | 76 | 0% | 74 | 0.1% | 22 |
| ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | 28.5K | 14% | 12.4K | 24.1% | 450 |
| ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | 33 | 0% | 32 | 0.1% | 24 |
| ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | 314 | 0.2% | 226 | 0.4% | 98 |
| ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | 9 | 0% | 9 | 0% | 5 |

# Robustez de Assinaturas Inferidas

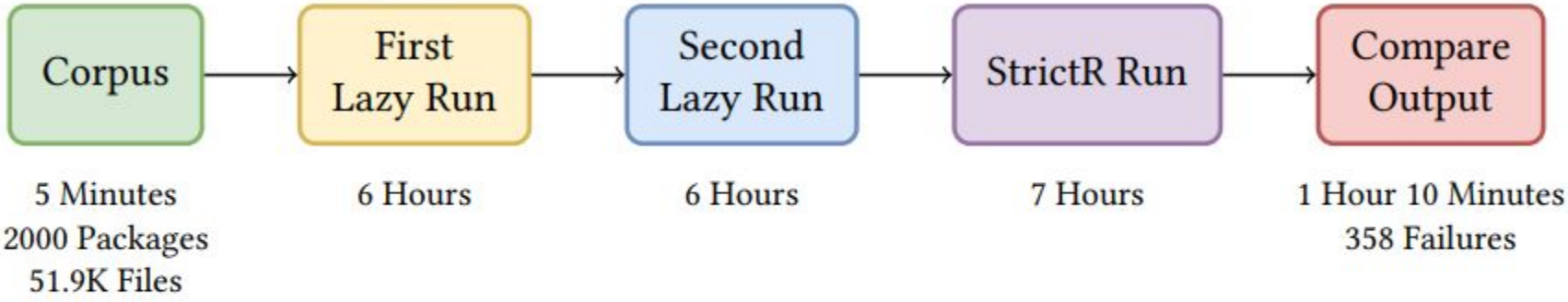



Fig. 5. Validation Pipeline

# 7. Experimentos de Desempenho

# 7. Experimentos de Desempenho

- Hipótese: a semântica estrita leva a programas mais rápidos.
- Compilador just-in-time - é esperado que:
  - A semântica estrita melhore o desempenho da maioria dos benchmarks, para ambos os níveis de Ř;
  - Uma parte do speedup venha da redução da coleta de lixo;
  - Obtenção de uma aceleração adicional devido a otimizações aprimoradas do compilador.
- Metodologia: Foi conduzido um experimento de estudo de limite para avaliar o desempenho mais alto possível, dado o estado atual do compilador.

# 7. Experimentos de Desempenho

- A Tabela apresenta os benchmarks. Foi escolhido um subconjunto do conjunto de benchmarks Ř que inclui um variante de cada benchmark.

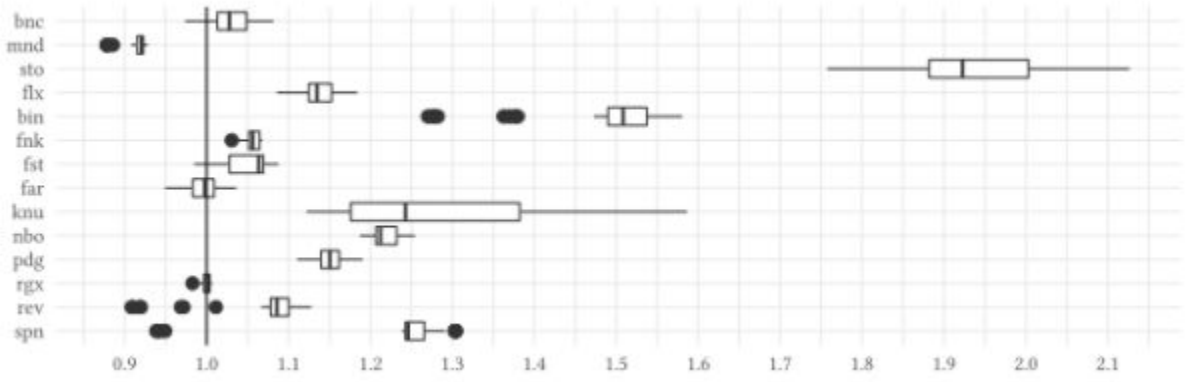
| Id | Benchmark | Suite |
|---|---|---|
| bnc | Bounce | Are we fast |
| mnd | Mandelbrot | Are we fast |
| sto | Storage | Are we fast |
| flx | Flexclust | Real thing |
| bin | Binarytrees | Shootout |
| fst | Fasta | Shootout |
| far | Fastaredux | Shootout |
| fnk | Fannkuchredux | Shootout |
| knu | Knucleotide | Shootout |
| nbo | Nbody | Shootout |
| pdg | Pidigits | Shootout |
| rgx | Regexdna | Shootout |
| rev | Reversecomplement | Shootout |
| spn | Spectralnorm | Shootout |

# 7. Experimentos de Desempenho

- As medições de desempenho são reunidas executando $t_e$ invocações de Ř em cada referência. Dentro de cada invocação medimos o tempo de execução de $t_i$ em processo.
- Como base, foi utilizado Ř que possui aceleração média de 1,89 sobre o GNU R no subconjunto do conjunto de benchmarks, com parâmetros $t_e$ = 1, $t_i$ = 15.
- Experimentos:
  - Aceleração com otimização.
  - Aceleração sem otimização.

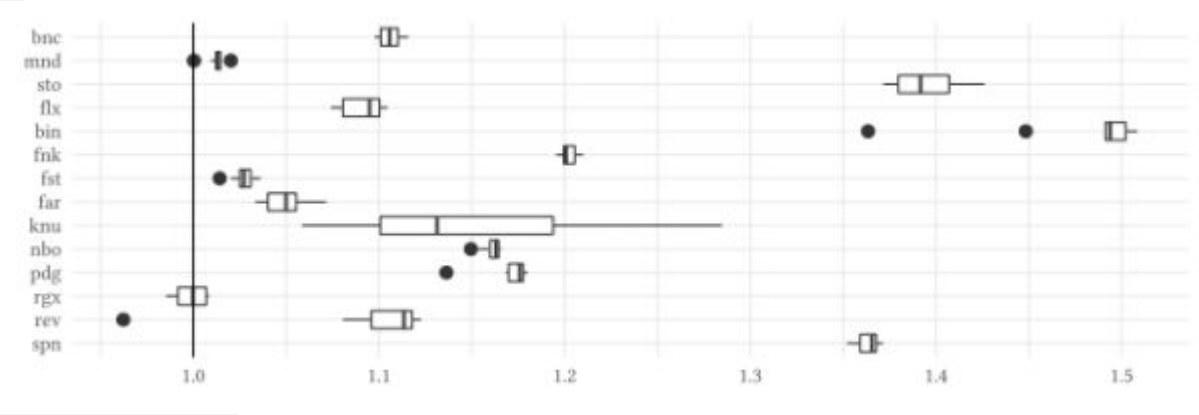
# 7. Aceleração com Otimização

- Primeiro, compara-se Ř com $Ř_s$. Este experimento estima a melhoria de ponta a ponta no desempenho que uma mudança em semântica estrita na linguagem R teria em Ř. A execução dos tempos foram medidos com $t_e$ = 4, $t_i$ = 25.



# 7. Aceleração sem Otimização

- Em seguida, repetiu-se o experimento, mas com o otimizador desativado. Em outras palavras, foram comparadas variantes não otimizadas de $\check{R}_s$ vs. $\check{R}$. Como esta variante executa, de forma geral 0,38× mais lento, optou-se por executar apenas com $t_e$ = 1, $t_i$ = 15.



# 8. Conclusões e Referências

# 8. Conclusões

- Por que R é considerado preguiçoso/lento (lazy)?
  - Isso acontece, pois é permitido que os usuários alterem reflexivamente expressões de argumentos, antes de avaliá-las.
- R se torna atraente para seus usuários, mesmo que eles não percebam que é uma linguagem preguiçosa.
  - No entanto, quando se trata de escrever um código R robusto a linguagem não permanece tão atraente.

# 8. Conclusões

- Proposto e avaliado uma estratégia para evoluir o R para o ecossistema de semântica estrita.
  - Primeiro: foram fornecidas assinaturas de rigor como uma extensão R não invasiva para evitar alterações no código.
  - Segundo: foram realizadas inferências de rigidez robustas para o código do pacote.
- Captura dos dados desejados.
  - Foi possível a execução da maior parte do código do cliente nos experimentos. Apenas 0,79% dos testes de pacotes falharam.

# 8. Conclusões

- Ao transformar R em uma linguagem principalmente estrita, o compilador just-in-time Ř executou 1,17× mais rápido através dos benchmarks mostrados sem mais alterações.
  - Incentivo aos usuários a adotarem nova semântica.
- Mudar o R para uma linguagem estrita seria benéfico de várias maneiras.
  - As implementações seriam mais eficazes de realizar;
  - Compiladores e análises de programas seriam mais fáceis de executar;
  - Os usuários seriam apresentados com uma semântica de chamada mais comumente esperada.

# 8. Referências


- Aviral Goel, Jan Ječmen, Sebastián Krynski, Olivier Flückiger, Jan Vitek - **Promises Are Made to Be Broken: Migrating R to Strict SemanticsIn-Person.** Disponível em: https://dl.acm.org/doi/10.1145/3485478. Acesso em março de 2022.

# Agradecemos a Atenção!