Ř Melts Brains

SCG Seminar 5.11.2019

Olivier Flückiger

Why a Compiler for R

R is...

...useful to many people

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Available CRAN Packages By Date of Publication

Date	Package	Title
2019-10-13	economiccomplexity	Methods Used in the Economic Complexity Literature
2019-10-13	flashlight	Shed Light on Black Box Machine Learning Models
2019-10-13	genvar	An Imperative Library for Data Manipulation
2019-10-13	metacom	Analysis of the 'Elements of Metacommunity Structure'
2019-10-13	stplanr	Sustainable Transport Planning
2019-10-13	wyz.code.metaTesting	Wizardry Code Meta Testing
2019-10-13	wyz.code.testthat	Wizardry Code Offensive Programming Test Generation
2019-10-12	BatchGetSymbols	Downloads and Organizes Financial Data for Multiple Tickers
2019-10-12	BayesNSGP	Bayesian Analysis of Non-Stationary Gaussian Process Models
2019-10-12	brglm2	Bias Reduction in Generalized Linear Models
2019-10-12	CollessLike	Distribution and Percentile of Sackin, Cophenetic and Colless-Like Balance Indices of Phylogenetic Trees
2019-10-12	EBPRS	Derive Polygenic Risk Score Based on Emprical Bayes Theory
2019-10-12	fusedest	Block Splitting Algorithm for Estimation with Fused Penalty Functions
2019-10-12	GMMAT	Generalized Linear Mixed Model Association Tests
2019-10-12	maotai	Tools for Matrix Algebra, Optimization and Inference
2019-10-12	MetaUtility	Utility Functions for Conducting and Interpreting Meta-Analyses
2019-10-12	NACHO	NanoString Quality Control Dashboard
2019-10-12	phaseR	Phase Plane Analysis of One- And Two-Dimensional Autonomous ODE Systems
2019-10-12	PNADcIBGE	Downloading, Reading and Analysing PNADc Microdata
2019-10-12	politeness	Detecting Politeness Features in Text
2019-10-12	restatapi	Search and Retrieve Data from Eurostat Database
2019-10-12	shinyglide	Glide Component for Shiny Applications
2019-10-12	soundgen	Parametric Voice Synthesis
2019-10-12	spectralGraphTopology	Learning Graphs from Data via Spectral Constraints
2019-10-12	strucchange	Testing, Monitoring, and Dating Structural Changes
2019-10-12	WVPlots	Common Plots for Analysis
2019-10-11	AlphaSimR	Breeding Program Simulations
2019-10-11	BayesBEKK	Bayesian Estimation of Bivariate Volatility Model

Why a Compiler for R

R is…

…useful to many people

…slow

…very hard to optimize

Why a Compiler for R

Because writing compilers is fun

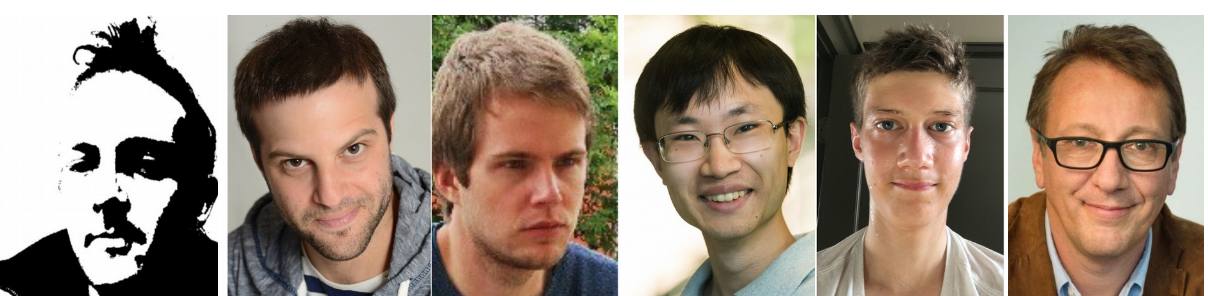
You **will** find interesting problems

R Melts Brains

An IR for First-Class Environments and Lazy Effectful Promises

DLS'19

Olivier Flückiger, Guido Chari, Jan Ječmen,
Ming-Ho Yee, Jakob Hain, Jan Vitek

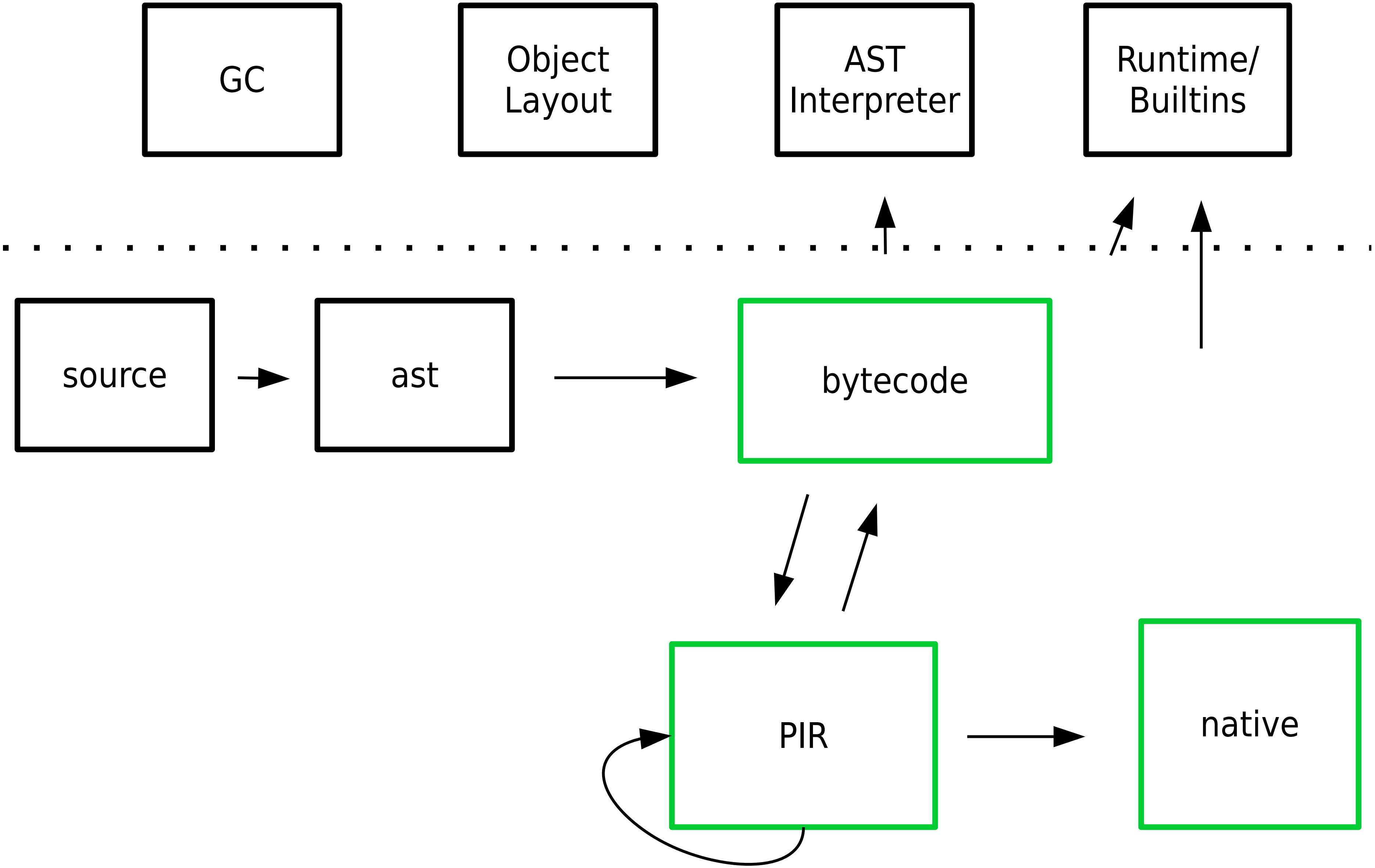


Northeastern University
Czech Technical University

R’s mutable variable scopes heavily interfere with compiler optimizations

PIR – an IR to explicitly model, analyze and lower R variables

How to design a compiler for R


GC
Object Layout
AST Interpreter
Runtime/ Builtins
source
ast
bytecode
PIR
native

Scope in R

Lexical vs. Dynamic Scope

litmus test

```
a <- 1

f <- function() {
  a
}

g <- function() {
  a <- 2
  f()
}
```

Variables contained in the environment…
…depends on the dynamics.

```
function() {
  if (...)
    b <- 1
  b
}
```

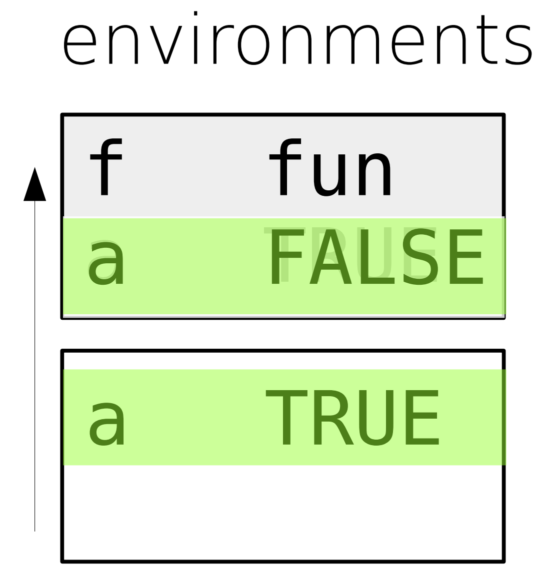
Closures capture environment…

…and can mutate it.

```
f <- function() {

   a <-  a
   a <<- FALSE

}
a <- TRUE
f()
a
```



```
f <- function() {

}
a <- TRUE
f()
a
```

Callees…

…can mutate the environment.

```
f <- function() {

    e <- parent.frame()
    rm(`a`, envir=e)

}
g <- function() {
  a <- TRUE
  f()
  a   # → object `a` not found
}
```

Arguments…

…are promises.

…can mutate the environment.

```
f <- function(q) {
  e <<- environment()
  a <- TRUE
  q
  a
}
f(
  rm(`a`, envir=e)
)
```

Local variables…

…stored in first-class environment

…accessible to callees

…accessible to promises

How to resolve variables and scopes

environments change dynamically

where are promises evaluated?

which function gets called?

An IR will Save Us

PIR design

- Single Static Assignment (SSA)
- Explicit environments and promises

Scope Resolution

a flow-sensitive analysis for reaching stores

Scope Resolution

```
function () {
  if (...) x <- 1
  else     x <- 2
  x
}
```

$$
\begin{array}{llcl}
BB_0: & \texttt{e1} & = & \texttt{MkEnv}\ (\ :G) \\
 & \%2 & = & \ldots \\
 & & & \texttt{Branch}\ (\%2,\ BB_1,\ BB_2) \\
BB_1: & \%4 & = & \texttt{LdConst}\ [1]\ 1 \\
 & & & \texttt{StVar}\ (\texttt{x},\ \%4,\ \texttt{e1}) \\
 & & & \texttt{Branch}\ BB_3 \\
BB_2: & \%7 & = & \texttt{LdConst}\ [1]\ 2 \\
 & & & \texttt{StVar}\ (\texttt{x},\ \%7,\ \texttt{e1}) \\
 & & & \texttt{Branch}\ BB_3 \\
BB_3: & \%10 & = & \texttt{LdVar}\ (\texttt{x},\ \texttt{e1}) \\
 & \%11 & = & \texttt{Force}\ (\%10)\ \texttt{e1} \\
 & & & \texttt{Return}\ (\%11)
\end{array}
$$

Scope Resolution : 1. Find Reaching Stores

```
function () {
  if (...) x <- 1
  else     x <- 2
  x
}
```

BB1
x = %4
BB2
x = %7
BB3
x = %4 | %7

BB_0 :	e1	=	MkEnv (: G)
	%2	=	…
			Branch (%2, BB_1, BB_2)
BB_1 :	%4	=	LdConst [1] 1
			StVar (x, %4, e1)
			Branch BB_3
BB_2 :	%7	=	LdConst [1] 2
			StVar (x, %7, e1)
			Branch BB_3
BB_3 :	%10	=	LdVar (x, e1)
	%11	=	Force (%10) e1
			Return (%11)

Scope Resolution : 2. Replace Loads

```
function () {
  if (...) x <- 1
  else     x <- 2
  x
}
```

BB1
  x = %4
BB2
  x = %7
BB3
  x = %4 | %7

BB_0 :	e1	=	MkEnv (: G)
	%2	=	...
			Branch (%2, BB_1, BB_2)
BB_1 :	%4	=	LdConst [1] 1
			StVar (x, %4, e1)
			Branch BB_3
BB_2 :	%7	=	LdConst [1] 2
			StVar (x, %7, e1)
			Branch BB_3
BB_3 :	%10	=	Phi (BB_1 : %4, BB_2 : %7)
			Return (%10)

This looks suspiciously easy

Scope Resolution : Stub Environments

```
function () {
  if (...){x <- 1 ;f()}
  else      x <- 2
  x
}
```

BB1
  x = ?
BB2
  x = %7
BB3
  x = ? | %7

$$
\begin{array}{llcl}
BB_0: & \texttt{e1} & = & (\texttt{MkEnv})(\ : G) \\
 & \%2 & = & \ldots \\
 & & & \texttt{Branch}\ (\%2,\ BB_1,\ BB_2) \\
BB_1: & \%4 & = & \texttt{LdConst}\ [1]\ 1 \\
 & & & \texttt{StVar}\ (\texttt{x},\ \%4,\ \texttt{e1}) \\
 & & & \texttt{Call} \\
 & & & \texttt{Branch}\ BB_3 \\
BB_2: & \%7 & = & \texttt{LdConst}\ [1]\ 2 \\
 & & & \texttt{StVar}\ (\texttt{x},\ \%7,\ \texttt{e1}) \\
 & & & \texttt{Branch}\ BB_3 \\
BB_3: & \%10 & = & \texttt{LdVar}\ (\texttt{x},\ \texttt{e1}) \\
 & \%11 & = & \texttt{Force}\ (\%10)\ \texttt{e1} \\
 & & & \texttt{Return}\ (\%11)
\end{array}
$$

Scope Resolution : Stub Environments

```
function () {
  if (...)
  else
  x
}
```

BB1
x = ?
BB2
x = %7
BB3
x = ? | %7

$BB_0 :$ e1 = (MkEnv)(: G)

Static analysis if possible...
...speculation if needed

$BB_1, BB_2)$

$BB_1 :$ %4 = LdConst [1] 1
StVar (x, %4, e1)
Call → deopt
Branch BB_3

$BB_2 :$ %7 = LdConst [1] 2
StVar (x, %7, e1)
Branch BB_3

$BB_3 :$ %10 = LdVar (x, e1)
%11 = Force (%10) e1
Return (%11)

Inlining of Closures and Promises

Promise Inlining

```
f <- function(b) b
f(x)
```

```
%1  =  MkClosure (f, G)
%2  =  MkArg (pr0, G)
%3  =  Call %1 (%2) G

f
 %6  =  LdArg (0)
 e7  =  MkEnv (b = %6 : G)
 %8  =  Force (%6) e7
        Return (%8)
```

Promise Inlining : 1. Normal Inlining

```
f <- function(b) b
f(x)
```

```
%1   =   MkClosure (f,  G)
%2   =   MkArg (pr0,  G)
# inlinee
e7   =   MkEnv (b = %2  : G)
%8   =   Force (%2) e7

pr0
  %4   =   LdVar (x,  G)
  %5   =   Force (%4) G
           Return (%5)

g (0)
v (b = %6  : G)
e (%6) e7
rn (%8)
```

inlinee retains environment

```
f <- function(b) {
  print(ls())       # -> “b”
  b
}
```

Promise Inlining : 2. Promise Inlining

```
f <- function(b) b
f(x)
```

$\%2 \;=\;$ `MkArg` (pr0, G)
\# inlinee
e7 $=$ `MkEnv` ($b = \%2$: G)
$\%8 \;=\;$ `Force` ($\%2$) e7

pr0
$\%4 \;=\;$ `LdVar` (x, G)
$\%5 \;=\;$ `Force` ($\%4$) G
`Return` ($\%5$)

dominating force instruction

Promise Inlining : 2. Promise Inlining

```
f <- function(b) b
f(x)
```

%2 = MkArg (pr0, G)
# inlinee
e6 = MkEnv (b = %2 : G)
# inlined promise
%4 = LdVar (x, G)
%5 = Force (%4) e6

Promise Inlining : 2. Promise Inlining

```
f <- function(b) b
f(x)
```

$\%2 \;=\; \mathtt{MkArg}\ (\mathrm{pr0},\ G)$

\# inlinee

$e6 \;=\; \mathtt{MkEnv}\ (b = \%2\ : G)$

\# inlined promise

$\%4 \;=\; \mathtt{LdVar}\ (x,\ G)$

$\%5 \;=\; \mathtt{Force}\ (\%4)\ \cancel{e6}$

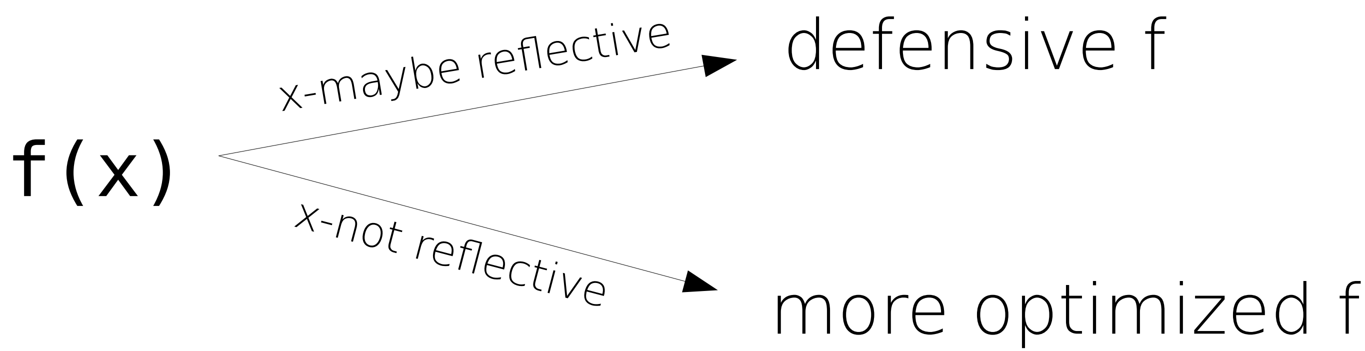
Assume `x` does not do reflection

Promise Inlining : 3. Is the promise safe?

Inter-procedural analysis

Speculation + Deoptimization

Version Dispatch



Results

(12%-65% of closures,
28%-87% of invocations)

…sometimes it is possible to statically resolve R scopes

…disabling either scope resolution or promise inlining significantly defeats the other optimizations

Ř

R is a bizarre language.

Static reasoning for some R programs is possible.

Explicitly model what is hard to reason.

https://github.com/reactorlabs/rir
https://o1o.ch/about/brains