Ř Melts Brains

SCG Seminar 5.11.2019

Olivier Flückiger
Why a Compiler for R

R is...

...useful to many people
<table>
<thead>
<tr>
<th>Date</th>
<th>Package</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-10-13</td>
<td>economiccomplexity</td>
<td>Methods Used in the Economic Complexity Literature</td>
</tr>
<tr>
<td>2019-10-13</td>
<td>flashlight</td>
<td>Shed Light on Black Box Machine Learning Models</td>
</tr>
<tr>
<td>2019-10-13</td>
<td>genvar</td>
<td>An Imperative Library for Data Manipulation</td>
</tr>
<tr>
<td>2019-10-13</td>
<td>metacom</td>
<td>Analysis of the 'Elements of Metacommunity Structure'</td>
</tr>
<tr>
<td>2019-10-13</td>
<td>splannr</td>
<td>Sustainable Transport Planning</td>
</tr>
<tr>
<td>2019-10-13</td>
<td>wyz.code.testthat</td>
<td>Wizardry Code Offensive Programming Test Generation</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>BatchGetSymbols</td>
<td>Downloads and Organizes Financial Data for Multiple Tickers</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>BayesNSGP</td>
<td>Bayesian Analysis of Non-Stationary Gaussian Process Models</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>brglm2</td>
<td>Bias Reduction in Generalized Linear Models</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>CollessLike</td>
<td>Distribution and Percentile of Sackin, Cophenetic and Colless-Like Balance Indices of Phylogenetic Trees</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>EBPRS</td>
<td>Derive Polygenic Risk Score Based on Empirical Bayes Theory</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>fusedest</td>
<td>Block Splitting Algorithm for Estimation with Fused Penalty Functions</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>GMMAT</td>
<td>Generalized Linear Mixed Model Association Tests</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>mafati</td>
<td>Tools for Matrix Algebra, Optimization and Inference</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>MetaUtility</td>
<td>Utility Functions for Conducting and Interpreting Meta-Analyses</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>NACHO</td>
<td>NanoString Quality Control Dashboard</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>phaseR</td>
<td>Phase Plane Analysis of One- And Two-Dimensional Autonomous ODE Systems</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>PNADcIBGE</td>
<td>Downloading, Reading and Analysing PNADc Microdata</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>politeness</td>
<td>Detecting Politeness Features in Text</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>restatapi</td>
<td>Search and Retrieve Data from Eurostat Database</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>shinyglide</td>
<td>Glide Component for Shiny Applications</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>soundgen</td>
<td>Parametric Voice Synthesis</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>spectralGraphTopology</td>
<td>Learning Graphs from Data via Spectral Constraints</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>strucchange</td>
<td>Testing, Monitoring, and Dating Structural Changes</td>
</tr>
<tr>
<td>2019-10-12</td>
<td>WVPlots</td>
<td>Common Plots for Analysis</td>
</tr>
<tr>
<td>2019-10-11</td>
<td>AlphaSimR</td>
<td>Breeding Program Simulations</td>
</tr>
<tr>
<td>2010.10.11</td>
<td>RavesRFK</td>
<td>Bayesian Estimation of Bivariate Volatility Model</td>
</tr>
</tbody>
</table>
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

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

Northeastern University
Czech Technical University

DLS’19
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
Scope in R
Lexical vs. Dynamic Scope

```r
a <- 1

f <- function() {
  a
}

g <- function() {
  a <- 2
  f()
}
```
Variables contained in the environment... 
...depends on the dynamics.

```javascript
function() {
  if (...) {
    b <- 1
    b
  }
}
```
Closures capture environment... ...and can mutate it.

```r
f <- function() {
  a <- a
  a <<- FALSE
}
a <- TRUE
f()
a
```

environments

```
<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>FALSE</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
```
f <- function() {
  a <- TRUE
  f()
  a
}
Scope
Callees...  
...can mutate the environment.

```r
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.

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

BB₀ : e₁ = MkEnv ( : G)
      %2 = ...
    Branch (%2, BB₁, BB₂)
BB₁ : %₄ = LdConst [1] 1
    StVar (x, %₄, e₁)
    Branch BB₃
BB₂ : %₇ = LdConst [1] 2
    StVar (x, %₇, e₁)
    Branch BB₃
BB₃ : %₁₀ = LdVar (x, e₁)
      %₁₁ = Force (%₁₀) e₁
    Return (%₁₁)
Scope Resolution: 1. Find Reaching Stores

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

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

- \( BB_0: \ e_1 = \text{MkEnv}(\ : G) \)
  - \( %2 = \ldots \) Branch \( (%2, \ BB_1, \ BB_2) \)
- \( BB_1: \ %4 = \text{LdConst}[1] 1 \)
  - \( \text{StVar}(x, %4, e_1) \) Branch \( BB_3 \)
- \( BB_2: \ %7 = \text{LdConst}[1] 2 \)
  - \( \text{StVar}(x, %7, e_1) \) Branch \( BB_3 \)
- \( BB_3: \ %10 = \text{LdVar}(x, e_1) \)
  - \( %11 = \text{Force}(%10) \ e_1 \)
  - \( \text{Return}(%11) \)
Scope Resolution: 2. Replace Loads

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

BB0: e1 = MkEnv ( : G)
    %2 = ...
    Branch (%2, BB1, BB2)

BB1: %4 = LdConst [1] 1
    StVar (x, %4, e1)
    Branch BB3

BB2: %7 = LdConst [1] 2
    StVar (x, %7, e1)
    Branch BB3

BB3: %10 = Phi (BB1: %4, BB2: %7)
    Return (%10)
This looks suspiciously easy
Scope Resolution : Stub Environments

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

BB0: e1 = (MkEnv)( : G)
    %2 = ...
    Branch (%2, BB1, BB2)

BB1: %4 = LdConst [1] 1
    StVar (x, %4, e1)

Call

BB2: %7 = LdConst [1] 2
    StVar (x, %7, e1)
    Branch BB3

BB3: %10 = LdVar (x, e1)
    %11 = Force (%10) e1
    Return (%11)
Scope Resolution: Stub Environments

function () {
if (...) 
    x = %4
else 
    x
}

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

BB0: e1 = (MkEnv)( : G)

BB1: %4 = LdConst [1] 1
     StVar (x, %4, e1)
     Call
     Branch BB3

BB2: %7 = LdConst [1] 2
     StVar (x, %7, e1)
     Branch BB3

BB3: %10 = LdVar (x, e1)

%11 = Force (%10) e1
Return (%11)
Inlining of Closures and Promises
Promise Inlining

\[ f \leftarrow \text{function}(b) \ b \]
\[ f(x) \]

\[
\begin{align*}
%1 &= \text{MkClosure} \ (f, \ G) \\
%2 &= \text{MkArg} \ (pr0, \ G) \\
%3 &= \text{Call} \ %1 \ (%2) \ G
\end{align*}
\]

\[
\begin{align*}
%6 &= \text{LdArg} \ (0) \\
e7 &= \text{MkEnv} \ (b = %6 : G) \\
%8 &= \text{Force} \ (%6) \ e7 \\
\text{Return} \ (%8)
\end{align*}
\]
Promise Inlining : 1. Normal Inlining

\[
f \leftarrow \text{function}(b) \ bn(x)
\]

inlinee retains environment

\[
f \leftarrow \text{function}(b) \{\ 
\text{print}(\text{ls}()) \quad # \text{->} \text{"b"} 
\}
\]
Promise Inlining : 2. Promise Inlining

\[ f \leftarrow \text{function(b) b} \]
\[ f(x) \]

**dominating force instruction**

\[
\begin{align*}
%2 &= \text{MkArg (pr0, G)} \\
\text{# inlinee} \\
e7 &= \text{MkEnv (b = %2 : G)} \\
%8 &= \text{Force (%2) e7}
\end{align*}
\]

pr0

\[
\begin{align*}
%4 &= \text{LdVar (x, G)} \\
%5 &= \text{Force (%4) G} \\
\text{Return (%5)}
\end{align*}
\]
Promise Inlining : 2. Promise Inlining

\[
f \leftarrow \text{function}(b) \ b
\]

\[
f(x)
\]

\[
\begin{align*}
\%2 &= \text{MkArg} \ (pr0, \ G) \\
&\quad \# \ \text{inlinnee} \\
\text{e6} &= \text{MkEnv} \ (b = \%2 : G) \\
&\quad \# \ \text{inlined promise} \\
\%4 &= \text{LdVar} \ (x, \ G) \\
\%5 &= \text{Force} \ (%4, \ \text{e6})
\end{align*}
\]
Promise Inlining : 2. Promise Inlining

\[
f \gets \text{function}(b) \ b\ \ f(x)
\]

\[
\begin{align*}
%2 & = \text{MkArg} (\text{pr0}, \ G) \\
\# & \text{ inlinnee} \\
e6 & = \text{MkEnv} (b = \%2 : G) \\
\# & \text{ inlined promise} \\
%4 & = \text{LdVar} (x, \ G) \\
%5 & = \text{Force} (%4) \ e6
\end{align*}
\]

Assume `x` does not do reflection
Promise Inlining: 3. Is the promise safe?

Inter-procedural analysis

Speculation + Deoptimization

Version Dispatch

\[ f(x) \]

- x-maybe reflective \rightarrow \text{defensive } f

- x-not reflective \rightarrow \text{more optimized } f
Results
...sometimes it is possible to 
statically resolve R scopes

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

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