A few words about profiling and debugging in R

Przemyslaw.Biecek@gmail.com, MIM University of Warsaw

Outline

1. system.time() - the simplest profiler,
2. Rprof() - default R profiler,
3. Rprofmem() - memory profiling in R,
4. profr() - graphical visualisation of Rprof results,
5. dump.frames() - post-mortem debugging,
6. traceback() - call stack,
7. debug() - debugging a function,
8. browser() - inspection of current environment,
9. recover() - browsing after an error.

system.time() - return CPU time that expr used

Very usefull tool for comparision of run-times for different expressions.

```
system.time(expr, gcFirst = TRUE)
```

Timings of evaluations of the same expression can vary considerably depending on whether the evaluation triggers a garbage collection. Thus the option **gcFirst** is usefull to produce more consistent timings.

```
> system.time( x=NULL; x = runif(10^5) )
   user  system elapsed
   0.02    0.00    0.01
```

Rprof() - the default R profiler

```
Rprof(filename = "Rprof.out", append = FALSE, interval = 0.02,
       memory.profiling=FALSE)
```

Rprof works by writing out the call stack every interval seconds, to the specified file.

Then the summaryRprof() function might be used to process the output file to produce a summary of the usage.

Functions will only be recorded in the profile log if they put a context on the call stack.

```
> Rprof("profiler.out", interval = 0.01, memory.profiling=TRUE)
> for (i in 1:10)
+     cat(1:10^6); runif(10^6); rexp(10^6); rnorm(10^6)
> Rprof(NULL)
>
> summaryRprof("profiler.out",memory="both")
```

Rprof() - the default R profiler

```
> summaryRprof("profiler.out",memory="both")
$by.self
         self.time self.pct total.time total.pct mem.total
cat          12.59     85.5      12.60      85.5       4.8
rnorm         1.12      7.6       1.12       7.6       8.6
rexp          0.60      4.1       0.60       4.1       9.5
runif         0.40      2.7       0.40       2.7       8.6
:             0.01      0.1       0.01       0.1       0.5
Rprof         0.01      0.1       0.01       0.1       0.0
printe        0.00      0.0      12.60      85.5       4.8
generate      0.00      0.0       2.12      14.4      26.7

$by.total
         total.time total.pct mem.total self.time self.pct
cat           12.60      85.5       4.8     12.59     85.5
rnorm          1.12       7.6       8.6      1.12      7.6
rexp           0.60       4.1       9.5      0.60      4.1
runif          0.40       2.7       8.6      0.40      2.7
:              0.01       0.1       0.5      0.01      0.1
Rprof          0.01       0.1       0.0      0.01      0.1

$sampling.time
[1] 14.73
```

profr() - for graphical visualisation of Rprof results

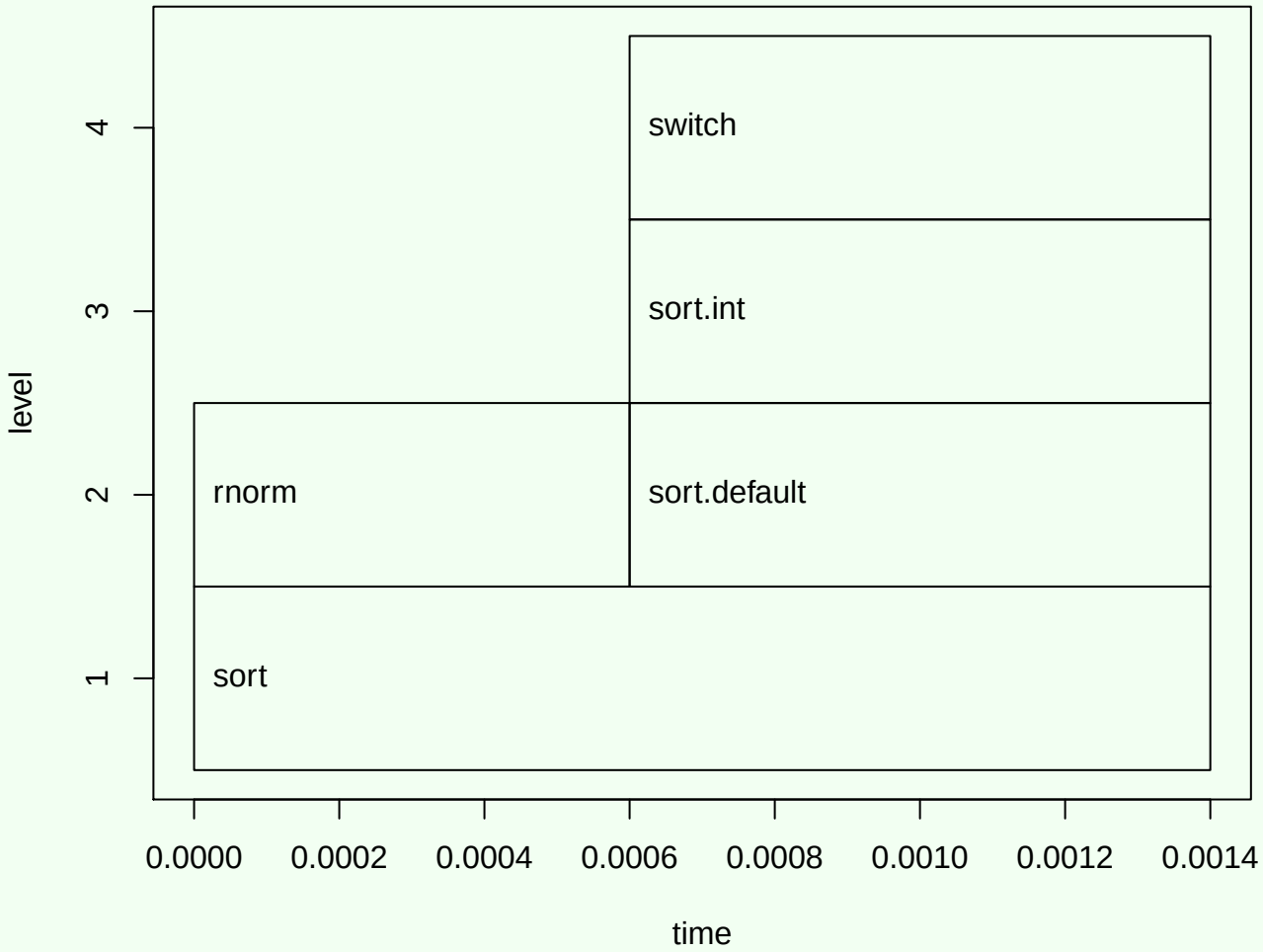
```
profr(expr, interval = 0.02, quiet = TRUE)
plot.profr(x, ..., minlabel = 0.1, angle = 0)
```

Function profr() is a wrapper around Rprof that provides results in an alternative data structure - data.frame.

This data.frame can be visualised with polymorphic functions: ggplot() or plot().

```
> library(profr)
> sortex <- profr(sort(rnorm(10^6)), interval = 0.0001)
Read 15 items
> head(sortex)
              f level   time start    end  leaf source
8          sort     1 0.0014 0e+00 0.0014 FALSE   base
9         rnorm     2 0.0006 0e+00 0.0006  TRUE  stats
10 sort.default     2 0.0008 6e-04 0.0014 FALSE   base
11     sort.int     3 0.0008 6e-04 0.0014 FALSE   base
12       switch     4 0.0008 6e-04 0.0014  TRUE   base
> plot(sortex, minlabel = 0.02)
> ggplot(sortex, minlabel = 0.02)
```

profr() - for graphical visualisation of Rprof results



dump.frames() - post-mortem debugging

```
dump.frames(dumpto = "last.dump", to.file = FALSE)
debugger(dump = last.dump)
```

These functions might be used to dump the evaluation environments (frames) and to examine dumped frames.

```
>  options(error=quote(dump.frames("errorDump", TRUE)))
>  fun <- function(x)
+    log(x)
+
>  fun("string")
Error in log(x) : Non-numeric argument to mathematical function
> list.files(pattern=".rda")
[1] "errorDump.rda"
```

dump.frames() - post-mortem debugging

```
> load("errorDump.rda")

> debugger(errorDump)
Execution halted
Available environments had calls:
1: fun("string")
2: log(x)
3: .execute(.Primitive("log"), x, envir = sys.parent(1))
Enter an environment number, or 0 to exit Selection: 1
Browsing in the environment with call:
fun("string")

Called from: debugger.look(ind)

Browse[1]> x
[1] "string"

Browse[1]> n
...
```

traceback() - Print Call Stacks

```
traceback(x = NULL, max.lines = getOption("deparse.max.lines"))
```

The function traceback() prints the call stack of the last uncaught error, i.e., the sequence of calls that lead to the error.

It is very useful when an error occurs with an unidentifiable error message.

```
> fun2 = function(x,y) fun(x); fun(y)
> fun2(1,"two")
Error in log(x) : Non-numeric argument to mathematical function
> traceback()
2: fun(y)
1: fun2(1, "two")
```

debug() - debug a function

Functions debug() and undebug() set and unset the debugging flag on a function.

When a function flagged for debugging is entered, normal execution is suspended and the body of function is executed one statement at a time.

Then new browser context is initiated for every step.

```
> debug(fun2)
> fun2(1,"two")
debugging in: fun2(1, "two")
debug:
fun(x)
fun(y)

Browse[1]> x
[1] 1
Browse[1]> n
debug: fun(x)
Browse[1]> y
[1] "two"
Browse[1]> n
```

browser() - inspection of current environment

```
browser()
```

This funtion interrupts the execution of an expression and allow the inspection of the environment where browser was called from.

A call to browser can be included in the body of a function, then when **browser()** is reached the R interpreter is avaliable.

recover() - browsing after an error

```
options(error = recover)
```

This is very usefull way to start the R interpreter after every error.

Recover prints the list of current calls, and prompts the user to select one of them.

When finished browsing in this call, type c to return to recover from the browser.

Type another frame number to browse some more, or type 0 to exit recover.