R PROFILING AND OPTIMISATION

GÜNTHER SAWITZKI

Pending changes

Warning: this is under construction.

This vignette contains experimental which may sink down to the package implementation, or vanish.

Known issues:

- Control information may be included as special stack in raw format.
- A list of profiles may become default. Only one profiling interval value per profile.
- Nodes may be implemented as *`factor`*.

Contents

Date: May 2013. *Revised:* July 2013
Typeset, with minor revisions: July 18, 2013 from SVN *Revision* : 175 2013-07-18.
Key words and phrases. R programming, profiling, optimisation, R program language.
An R *vignette for package sprof.*
URL: `http://sintro.r-forge.r-project.org/`
Private Version

.

Profiling facilities in R

R provides the basic instruments for profiling, both for time based samplers as for event based instrumentation. However this source of information seems to be rarely used.

Maybe the supporting tools are not adequate. The summaries provided by R reduce the information beyond necessity. Additional packages are available, but these are not sufficiently action oriented.

ToDo: add reference to R profiling info

With package ***sprof*** we want to give a data representation that keeps the full profile information. Tools to answer common questions are provided. The data structure should make it easy to extend the tools as required.

The package is currently distributed at r-forge as part of the ***sintro*** material.

To install this package directly within R, type
install.packages("sprof",repos="http://r-forge.r-project.org")

To install the recent package from source directly within R, type
install.packages("sprof",repos="http://r-forge.r-project.org",type="source")

LaTeX layout tools and R settings

You may want to skip this section, unless you want to modify the vignette for your own purposes, or look at the internals.

This is the main library we are going to use.

Input
```
library(sprof)
```

We want immediate warnings, if necessary. Set to level 2 to handle warnings as error.

Input
```
message("switching  options(warn=1) -- immediate warning on")
options(warn=1)
```

We want a second chance on errors.

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

Print parameters used here:

Input
```
options(width = 72)
options(digits = 6)
```

For larger tables and data frames, we use a kludge to avoid long outputs.

ToDo: add keep3 to keep header, some middle, tail

Input
```
xcutdata.frame <- function(df,  cut, margin){
#! keep3, to add: margin top - random center - margin bottom
        if (!is.data.frame(df)) return(df)
        nrow <- nrow(df)
        # cut a range if it is not empty.
        # Quiet noop else.
        # Does not cut single lines.
         cutrng <- function(cutfrom,cutto){
                 if (cutfrom<cutto){
                df[cutfrom,] <- NA
                if (!is.null(rownames(df))) rownames(df)[cutfrom] <- "< cut >"
                if (!is.null(df$name)) df$name[cutfrom] <- ""

                cutfrom <- cutfrom+1
                df[-(cutfrom:cutto),]
                }#if
        }
        if (!missing(cut))  {df <- cutrng(cut[1],cut[2]); return(df)}
        if (!missing(margin))  {
                if (length(margin)==1) margin <-  c(margin,margin)
        cut <- c(margin[1]+1,nrow-margin[2])
        df <-cutrng(cut[1],cut[2]);
        return(df)}
#        if (!missing(keep3))  { cut <- c(keep3[1]+1, keep3[1]+1,
#                nrow-keep3[3]-1,nrow-keep3[3]-1)
#        if (cut[3]-cut[4] > keep3[2]+2){delta<-(cut[3]-cut[2]) div 2
#        cut[3]<-0
#        browser()
#        } else df <- cutrng(cut[1],cut[4])
#        cutrng(cut[1],cut[4]) return(df)}
}
```

ToDo: remove text vdots from string/-name columns. Use empty string.

We use the R function *xtable()* for output and LaTeX *longtable*. A convenient wrapper to use this in out *Sweave* source is:

Input
```
library(xtable)
prxt <- function (x, digits=2,        caption=NULL,
        label=NULL, ...)  {
        margin <- 10
        if (nrow(x)> 2*margin+3) x <-xcutdata.frame(x, margin=margin)
        print(
                xtable(x, digits=digits, caption=caption,
                                label=label, ...),
                floating=FALSE,
                tabular.environment="longtable",
                caption.placement="top",
                NA.string="\\vdots")
        }
```

This is to be used with `<<print=FALSE, results =tex, label=tab:prxx>>=`

1. Profiling

The basic information provided by all profilers in is a protocol of sampled stacks. For each recorded event, the protocol has one record, such as a line with a text string showing the sampled stack.

We use profiles to provide hints on the dynamic behaviour of programs. Most often, this is used to improve or even optimise programs. Sometimes, it is even used to understand some algorithm.

Profiles represent the program flow, which is considered to be laid out by the control structure of a program. The control structure is represented by the control graph, and this leads to the common approach to (re)construct the control graph, map the profile to this graph, and used graph based methods for further analysis. The prime example for this strategy is the GNU profiler *gprof* (see `http://sourceware.org/binutils/docs/gprof/`) which is used as master plan for many common profilers.

It is only half of the truth that the control graph can serve as a base for the profiled stacks. In R, we have some peculiarities.

lazy evaluation: Arguments to functions can passed as promises. These are only evaluated when needed, which may be at a later time, and may then lead to insertions in the stack. So we may have information resulting from the data flow, interspersed with the control flow.

memory management: Allocation of memory, and garbage collection, may interfere and leave their traces in the stack. While allocation is closely related to the visible control flow, garbage collection is a collective effect largely out of control of the code to execute.

primitives: Internal functions may escape the usual stack conventions and execute without leaving any identifiable trace on the stack.

control structures: In R, many control structures are implemented as function. Most notably, the *apply()* family appear as function calls and lead to cliques in the graph representation that do not correspond to relevant structures. Since these functions are well know, they can have a special treatment.

So while the stack follows an overall well known dynamics, in R there are exceptions from regularity. The general approach, by *summaryRprof()* and others, is to reduce the profile to node information, or two consider single transitions.

We take a different approach. We take the stacks, as recorded in the profiles as our basic information unit. From this, we ask: what are the actions we need to answer our questions? Representation in graphs may come later, if they can help.

If the stacks would come from the control flow only, we could make use of the sequential nature of stacks. But since we have to live with the R specific interferences, we stay with the raw stacks.

ToDo: rearrange stacks? detect order?

In this presentation, we will use a small list of examples Since *Rprof* is not implemented on all systems, and since the profiles tend to get very large, we use some prepared examples that are frozen in this vignette and not included in the distribution, but all the code to generate the examples is provided.

1.1. Simple regression example.

Input

```
n <- 10000
x <- runif(n)
err <- rnorm(n)
y <- 2+ 3 * x + err
reg0data <- data.frame(x=x, y=y, err=err)
rm(x,y,err)
```

We will use this example to illustrate the basics. Of course the immediate questions are the variance between varying samples, and the influence of the sample size n. We keep everything fixed, so the only issue for now is the computational performance under strict iid conditions.

Still we have parameters to choose. We can determine the profiling granularity by setting the timing interval, and we can use repeated measurements to increase precision below the timing interval.

The timing interval should depend on the clock speed. Using for example 1ms amounts to some 1000 steps on a current CPU, per kernel.

ToDo: Can we calibrate times to CPU rate? Introduce cpu clock cycle as a time base

If we use repeated samples, the usual rules of statistics applies. So taking 100 runs and taking the mean reduces the standard deviation by a factor 1/10.

Following the usal R conventions, seconds are used as time base for parameters. However report will use ms as a time base.

Here is an example how to take a profile, using basic R. See section 1.1.2 on page 9 how to use *sampleRprof* in package *sprof* for an easier solution.

Input
```
profinterval <- 0.001
simruns <- 100
Rprof(filename="RprofsRegressionExpl.out", interval = profinterval)
        for (i in 1:simruns) xxx<- summary(lm(y~x, data=reg0data))
Rprof(NULL)
```

We now have the profile data in a file *RprofsRegressionExpl.out*. For this vignette, we use a frozen version *RprofsRegressionExpl01.out*.

1.1.1. *R basic.* The basic R functions invite us to get a summary.

Input
```
sumRprofRegressionExpl <- summaryRprof("RprofsRegressionExpl01.out")
str(sumRprofRegressionExpl, vec.len=3)
```

Output
```
List of 4
 $ by.self        :'data.frame':        41 obs. of  4 variables:
  ..$ self.time : num [1:41] 0.087 0.057 0.051 0.043 0.042 0.04 0.032 0.026 ...
  ..$ self.pct  : num [1:41] 16.67 10.92 9.77 8.24 ...
  ..$ total.time: num [1:41] 0.113 0.099 0.069 0.043 0.474 0.045 0.033 0.114 ...
  ..$ total.pct : num [1:41] 21.65 18.97 13.22 8.24 ...
 $ by.total       :'data.frame':        62 obs. of  4 variables:
  ..$ total.time: num [1:62] 0.522 0.522 0.521 0.521 0.521 0.521 0.521 0.521 ...
  ..$ total.pct : num [1:62] 100 100 99.8 99.8 ...
  ..$ self.time : num [1:62] 0.006 0 0.001 0 0 0 0 0 ...
  ..$ self.pct  : num [1:62] 1.15 0 0.19 0 0 0 0 0 ...
 $ sample.interval: num 0.001
 $ sampling.time  : num 0.522
```

The summary reduces the information contained in the profile to marginal statistics per node. This is provided in two data frames giving the same information, only in different order.

The file contains several spurious recordings: nodes that have been recorded only few times. It is worth noting these, but then they better be discarded. We use a time limit of 4ms, which given our sampling interval of 1ms means we require more than four observations.

Input
```
prxt(sumRprofRegressionExpl$by.self,
        caption="summaryRprof result: by.self as final stack entry, all records",
        label="tab:prSRREbs")
```

Table 1: summaryRprof result: by.self as final stack entry, all records

	self.time	self.pct	total.time	total.pct

”lm.fit”	0.09	16.67	0.11	21.65
”[.data.frame”	0.06	10.92	0.10	18.97
”model.matrix.default”	0.05	9.77	0.07	13.22
”as.character”	0.04	8.24	0.04	8.24
”lm”	0.04	8.05	0.47	90.80
”summary.lm”	0.04	7.66	0.04	8.62
”structure”	0.03	6.13	0.03	6.32
”na.omit.data.frame”	0.03	4.98	0.11	21.84
”anyDuplicated.default”	0.02	4.21	0.02	4.21
”as.list.data.frame”	0.02	4.21	0.02	4.21
< cut >	⋮	⋮	⋮	⋮
”FUN”	0.00	0.19	0.01	1.34
”%in%”	0.00	0.19	0.00	0.77
”deparse”	0.00	0.19	0.00	0.38
”$”	0.00	0.19	0.00	0.19
”as.list.default”	0.00	0.19	0.00	0.19
”as.name”	0.00	0.19	0.00	0.19
”coef”	0.00	0.19	0.00	0.19
”file”	0.00	0.19	0.00	0.19
”NCOL”	0.00	0.19	0.00	0.19
”terms.formula”	0.00	0.19	0.00	0.19

Input
```
prxt(sumRprofRegressionExpl$by.total[sumRprofRegressionExpl$by.total$total.time>0.004,],
        caption="summaryRprof result: by.total, total time > 4ms",
        label="tab:prSRREbt")
```

Table 2: summaryRprof result: by.total, total time > 4ms

	total.time	total.pct	self.time	self.pct
”<Anonymous>”	0.52	100.00	0.01	1.15
”Sweave”	0.52	100.00	0.00	0.00
”eval”	0.52	99.81	0.00	0.19
”doTryCatch”	0.52	99.81	0.00	0.00
”evalFunc”	0.52	99.81	0.00	0.00
”try”	0.52	99.81	0.00	0.00
”tryCatch”	0.52	99.81	0.00	0.00
”tryCatchList”	0.52	99.81	0.00	0.00
”tryCatchOne”	0.52	99.81	0.00	0.00
”withVisible”	0.52	99.81	0.00	0.00
< cut >	⋮	⋮	⋮	⋮
”as.list”	0.02	4.41	0.00	0.00
”anyDuplicated.default”	0.02	4.21	0.02	4.21
”as.list.data.frame”	0.02	4.21	0.02	4.21
”sapply”	0.01	2.68	0.00	0.19
”match”	0.01	2.11	0.00	0.19

"[[.data.frame"	0.01	1.53	0.00	0.19
"[["	0.01	1.53	0.00	0.00
"rep.int"	0.01	1.34	0.01	1.34
"FUN"	0.01	1.34	0.00	0.19
"list"	0.01	0.96	0.01	0.96

1.1.2. *Package sprof.* In contrast to the common R packages, in our implementation we take a two step approach. First we read in the profile file to an internal representation. Analysis is done in later steps.

Input
```
sprof01<- readRprof("RprofsRegressionExpl01.out")
```

We keep this example and use the copy ***sprof01*** of it extensively for illustration.

Input
```
save(sprof01, file="sprof01lm.RData")
```

To run the vignette with a different profile, replace ***sprof01*** by your example. You still have the file for reference.

Package ***sprof*** provides a function ***sampleRprof()*** to take a sample and create a profile on the fly, as in

Input
```
sprof01temp <- sampleRprof(runif(10000), runs=100)
```

The basic data structure consists of four data frames. The ***info*** section collects global information from the input file, such as an identification strings and various global matrix. The ***nodes*** section initially gives the same information marginal information as ***summaryRprof***. The ***stacks*** section puts the node information into their calling context as found in the input profile file. The ***profiles*** section gives the temporal context. It is implemented as a list, but conceptually it is a data frame. Implementing it as a list allows run length encoding of variables, which unfortunately is not allowed by R in data frames.

ToDo: add sampling.interval, sampling.time for backward compability

Input
```
str(sprof01, max.level=2, vec.len=3,nchar.max=40)
```

Output
```
List of 4
 $ info    :'data.frame':        1 obs. of  8 variables:
  ..$ id       : Factor w/ 1 level "\"RprofsRegressionExpl01.out\" 2013-06-"| __truncated__: 1
  ..$ date     : POSIXct[1:1], format: "2013-07-18 22:06:05"
  ..$ nrnodes  : int 62
  ..$ nrstacks : int 50
  ..$ nrrecords: int 522
  ..$ firstline: Factor w/ 1 level "sample.interval=1000": 1
  ..$ ctllines : Factor w/ 1 level "sample.interval=1000": 1
  ..$ ctllinenr: num 1
 $ nodes   :'data.frame':        62 obs. of  5 variables:
  ..$ name      : Factor w/ 62 levels "!","..getNamespace",..: 1 2 3 4 5 6 7 8 ...
  ..$ self.time : num [1:62] 2 0 2 0 0 57 0 1 ...
  ..$ self.pct  : num [1:62] 0.38 0 0.38 0 ...
  ..$ total.time: num [1:62] 2 1 4 26 99 99 8 8 ...
  ..$ total.pct : num [1:62] 0.03 0.01 0.05 0.34 1.29 1.29 0.1 0.1 ...
 $ stacks  :'data.frame':        50 obs. of  7 variables:
  ..$ nodes         :List of 50
  ..$ shortname     : Factor w/ 50 levels "S<A>eFttCtCLtCOdTCwVeesleem.m..n.n...[["| __truncated__,.
```

```
  ..$ refcount      : num [1:50] 1 5 26 55 13 43 51 87 ...
  ..$ stacklength   : int [1:50] 19 20 19 21 14 15 15 14 ...
  ..$ stackheadnodes: int [1:50] 52 52 52 52 52 52 52 52 ...
  ..$ stackleafnodes: int [1:50] 27 28 41 6 39 14 38 30 ...
  ..$ stackssrc     : Factor w/ 50 levels "! [.data.frame [ na.omit.data.frame na."| __truncated__,..
 $ profiles:List of 4
  ..$ data    : int [1:522] 1 2 2 3 4 4 5 5 ...
  ..$ mem     : NULL
  ..$ malloc  : NULL
  ..$ timesRLE:List of 2
  .. ..- attr(*, "class")= chr "rle"
 - attr(*, "class")= chr [1:2] "sprof" "list"
```

The nodes do not come in a specific order. Access via a permutation vector is preferred. This allows different views on the same data set. For example, table 4 on the next page uses a permutation by total time, and a selection (compare to table 2 on page 8). The only difference is that we work on a ms base internally, whereas R uses seconds as a base.

Input
```
nodes <- sprof01$nodes[order(sprof01$nodes$self.time, decreasing=TRUE),]
prxt(nodes[nodes$self.time>4,],
caption="splot result: by.self, self time > 4ms",
        label="tab:prspbtself")
```

Table 3: splot result: by.self, self time > 4ms

	name	self.time	self.pct	total.time	total.pct
30	lm.fit	87.00	16.67	113.00	1.47
6	[.data.frame	57.00	10.92	99.00	1.29
38	model.matrix.default	51.00	9.77	69.00	0.90
14	as.character	43.00	8.24	43.00	0.56
29	lm	42.00	8.05	474.00	6.16
51	summary.lm	40.00	7.66	45.00	0.59
49	structure	32.00	6.13	33.00	0.43
41	na.omit.data.frame	26.00	4.98	114.00	1.48
13	anyDuplicated.default	22.00	4.21	22.00	0.29
16	as.list.data.frame	22.00	4.21	22.00	0.29
40	na.omit	20.00	3.83	134.00	1.74
39	model.response	13.00	2.49	56.00	0.73
36	model.frame.default	12.00	2.30	168.00	2.18
46	rep.int	7.00	1.34	7.00	0.09
10	<Anonymous>	6.00	1.15	522.00	6.79
28	list	5.00	0.96	5.00	0.07

At this level, it is helpful to note the expectations, and only then inspect the timing results. Since we a running a linear model, we are nor surprised to see functions related to linear models on the top of the list. we may however be surpised to see functions related to data acces and to character conversion very high on the list. The sizeable amount of time spent on NA handling is another aspect that is surprising.

Input
```
nodes <- sprof01$nodes[order(sprof01$nodes$total.time, decreasing=TRUE),]
prxt(nodes[nodes$total.time>4,],
caption="splot result: by.total, total time > 4ms",
        label="tab:prspbt")
```

Table 4: splot result: by.total, total time > 4ms

	name	self.time	self.pct	total.time	total.pct
10	<Anonymous>	6.00	1.15	522.00	6.79
52	Sweave	0.00	0.00	522.00	6.79
21	doTryCatch	0.00	0.00	521.00	6.78
22	eval	1.00	0.19	521.00	6.78
23	evalFunc	0.00	0.00	521.00	6.78
55	try	0.00	0.00	521.00	6.78
56	tryCatch	0.00	0.00	521.00	6.78
57	tryCatchList	0.00	0.00	521.00	6.78
58	tryCatchOne	0.00	0.00	521.00	6.78
62	withVisible	0.00	0.00	521.00	6.78
$<$ cut $>$	\vdots	$\vdots$	$\vdots$	$\vdots$	$\vdots$
61	vapply	3.00	0.57	23.00	0.30
13	anyDuplicated.default	22.00	4.21	22.00	0.29
16	as.list.data.frame	22.00	4.21	22.00	0.29
47	sapply	1.00	0.19	14.00	0.18
31	match	1.00	0.19	11.00	0.14
7	[[	0.00	0.00	8.00	0.10
8	[[.data.frame	1.00	0.19	8.00	0.10
25	FUN	1.00	0.19	7.00	0.09
46	rep.int	7.00	1.34	7.00	0.09
28	list	5.00	0.96	5.00	0.07

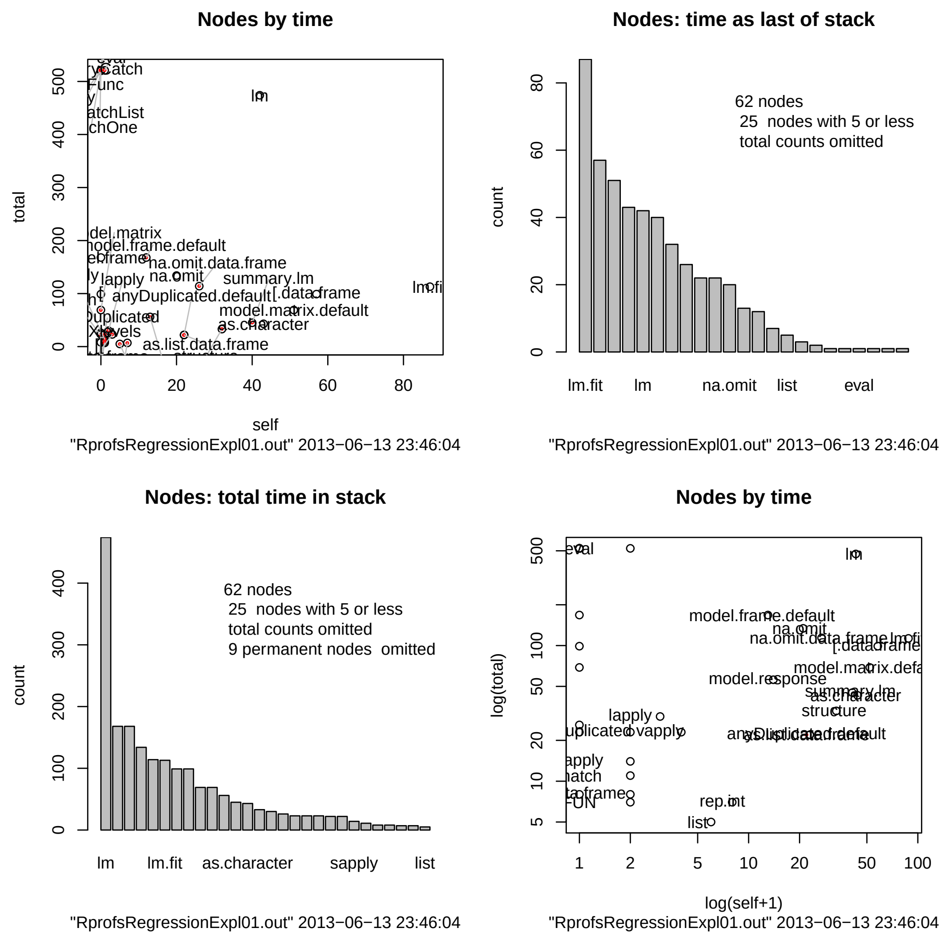
ToDo: remove text vdots from string/-name columns

Given the sampling structure of the profiles, two aspect are common. The sampling picks up scafffoling functions with a hicht, nearly constant requency. And the sampling will pick up rare recordings that are near to detection range. The display functions hide these effects by default. In our example, about half of the nodes are cleared by this garbage collector.

Common rearrangements as by total time and by self time are supplied by the display functions.

Plot, for example, currently gives a choice of four displays for nodes.

Input
```
oldpar <- par(mfrow=c(2,2))
plot_nodes(sprof01)
par(oldpar)
```



We can add colour. To illustrate this, we encode the frequency of the nodes as colour. As a palette, we choose a heat map here.

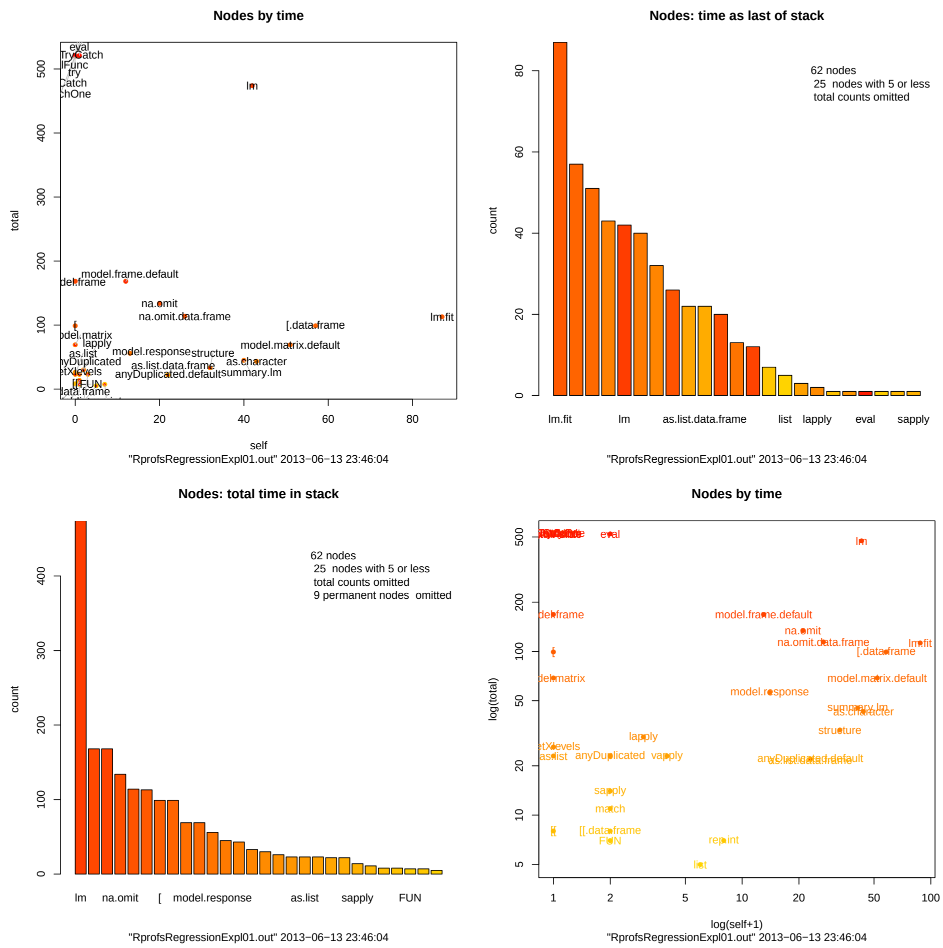
ToDo: apply colour to selction?
ToDo: spread colour on displayed part

Input
```
freqrank01 <- rank(-sprof01$nodes$total.time, ties.method="random")
freqrankcol01 <- heat.colors(length(freqrank01))
```

ToDo: colour by class – redo. Bundle colour index with colour?

Here is the node view using these choices:

Input
```
sprof01$nodes$icol <- freqrank01
oldpar <- par(mfrow=c(2,2))
plot_nodes(sprof01, col=freqrankcol01)
par(oldpar)
```



Colour is considered a volatile attribute. So you may need to pay some attention to keep colour indices (and colour palettes) aligned to your context.

ToDo: support colour in a structure

1.1.3. *Node classes.* We can add attributes to the plots. But we can also attributes to the nodes, and use these in the plots. In principle, this has been alway available. We are now making explicit use of this possibility.

The attribute `icol` is a special case which we used above. If present, it will be interpreted as an index to a colour table. For example, we can collect special well known functions in groups.

The node information is to some part arbitrary. You may achive the same functionality by different functions, and you will see different load in the profiles. Grouping nodes may be a mean to clarify the picture.

Grouping may also help you to focus your attention. “HOT” and “cold” may be ver helpful tags. These can be used in a flexible way.

ToDo: Move class attributes to package code

ToDo: add class by keyword

ToDo: add class by package

ToDo: Supply colour tables

Input

```
x_apply <- c("apply", "lapply", "vapply", "sapply")
x_as <- c("as.list", "as.data.frame", "as.list.data.frame",
        "as.character", "as.list.default","as.name")
```

(Extend as you need it) and then us as for example:

Input
```
nodeclass <- rep("x_nn", sprof01$info$nrnodes)
nodeclass[sprof01$nodes$name %in% x_apply] <- "x_apply"
nodeclass[sprof01$nodes$name %in% x_as] <- "x_as"
```

or use assignments on the fly

Input
```
nodeclass[sprof01$nodes$name %in%
        c("eval",  "evalFunc",
                "try", "tryCatch", "tryCatchList", "tryCatchOne",
                "doTryCatch")
                ] <- "x_eval"
nodeclass[sprof01$nodes$name %in%
        c("model.frame", "model.matrix.default","model.frame.default",
         " model.response", "model.matrix", "model.response")
                ] <- "x_model"
nodeclass[sprof01$nodes$name %in%
        c("lm", "lm.fit", "summary.lm")
                ] <- "x_lm"
nodeclass[sprof01$nodes$name =="<Anonymous>"] <- "x_Anon"
```

Input
```
sprof01$nodes$icol <-as.factor(nodeclass)
```

adds a sticky color attribute. To interpret, you should choose your preferred color palette, for example

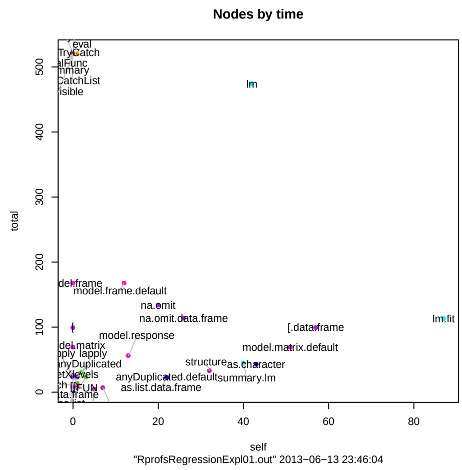
ToDo: add a reference to colorbrewer

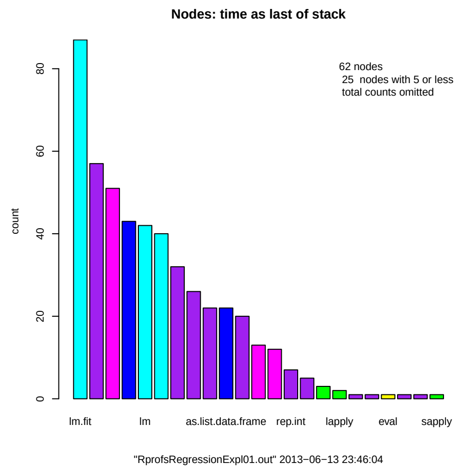
Input
```
classcol=c("red", "green", "blue", "yellow", "cyan", "magenta", "purple")
```

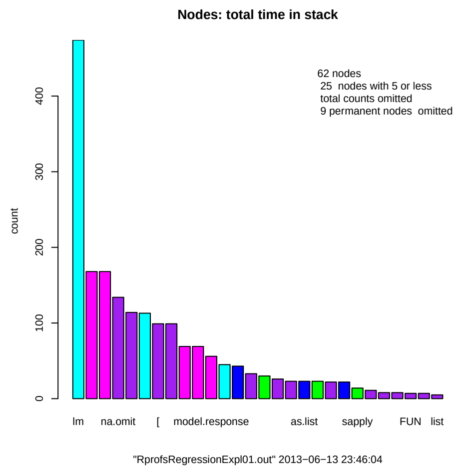
ToDo: plot_nodes: make col explicite

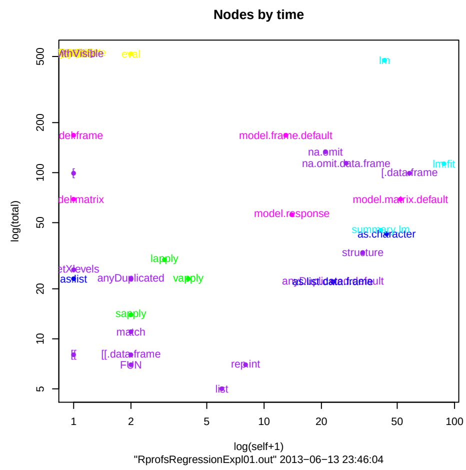
ToDo: Defaults by class

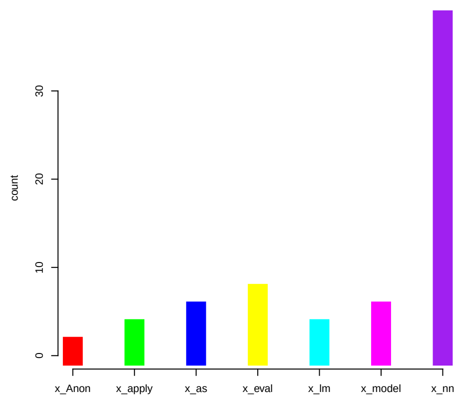
Input
```
oldpar <- par(mfrow=c(3,2))
plot_nodes(sprof01, col=classcol)
plot(table(sprof01$nodes$icol),
        type="h", lwd=20,col=classcol, lend="square", ylab="count")
nodescloud(sprof01, min.freq=5,
                col=classcol)
par(oldpar)
```

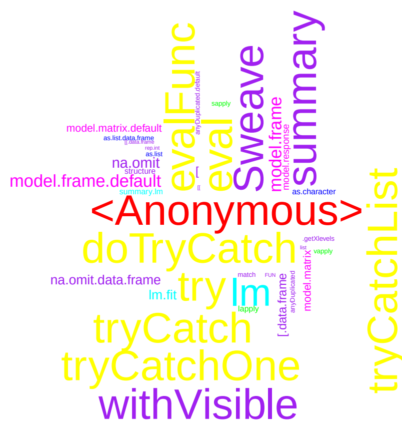












You can break down the frequency by the classes you have define. But beware of Simpson's paradox. The information you think you see may be strongly affected by your choices - what you see are reflections of conditional distributions. These may very different from the global picture.

If package ***wordcloud*** is installed, a different view is possible. This is added in the plots above.

ToDo: make more flexible and add to plot_node

2. A better grip on profile information

The basic information provided by all profilers in R is a protocol of sampled stacks. The conventional approach is to break the information down to nodes and edges. The stacks provide more information than this. One way to access it is to use linking to pass information. This has already been used on the node level in section 1.1.2 on page 9

ToDo: add attributes to stacks, and discuss scope

ToDo: sorting/arranging stacks

2.1. **The internal details.** For each recorded event, the protocol records one line with a text string showing the sampled stack (in reverse order: most recent first). The stack lines may be preceded by header lines with event specific information. The protocol may be interspersed with control information, such as information about the timing interval used.

We know that the structural information, static information as well as dynamic information, can be represented with the help of a graph. For a static analysis, the graph representation may be the first choice. For a dynamic analysis, the stack information is our first information. A stack is a connected path in the program graph. If we start with nodes and edges, we loose information which is readily available in record of stacks.

As we know that we are working with stacks, we know that they have their peculiarities. Stacks tend to grow and shrink. Subsequent events will have extensions and shrinkages of stacks (if the recording is on a fine scale), or stack sharing common stumps (if the recording is on a coarser scale).

There have always been interrupts, and these show up in profiles. In R, this is related problem (GC)

The graph is a second instance that is (re)constructed from the stack recording.

Here is the way we represent the profile information:

The profile log file is sanitised:

- Control lines are extracted and recorded in a separate list.
- Head parts, if present, ere extracted and recorded in a matrix that is kept line-aligned with the remainder
- Line content is standardised, for example by removing stray quotation marks etc.

After this, the sanitised lines are encoded as a vector of stacks, and references to this.

If necessary, these steps are done by chunks to reduce memory load.

From the vector of stacks, a vector of nodes (or rather node names) is derived.

The stacks are now encoded by references to the nodes table. For convenience, we keep the (sanitised) textual representation of the stacks.

So far, texts are in reverse order. For each stack, we record the trailing leaf, and then we reverse order. The top of stack is now on first position.

Several statistics can be accumulated easily as a side effect.

Conceptually, the data structure consist of three tables (the implementation may differ, and is subject to change).

The profiles table is the representation of the input file. Control lines are are collected in a special table. With the control lines removed, the rest is a table, one row per input line. The body of the line, the stack, is encoded as a reference to a stacks table (obligatory) and header information (optional).

The stacks table contains the collected stacks, each stack encoded as a list of references to the node table. This is obligatory. This list is kept in reverse order (root at position 1). A source line representing the stack information may be kept (optional).

The nodes table keeps the names at the nodes.

Sometimes, it is more convenient to use a simple representation, such as a matrix. Several extraction routines are provided for this, and the display routines make heavy use of this. See table 5.

ToDo: complete matrix conversion

TABLE 5. Extraction and conversion routines

profiles_matrix()	incidence matrix: nodes by event
stacks_matrix()	incidence matrix: nodes by stack
list.as.matrix()	fill list to equal length and convert to matrix
stackstoadj()	stacks to (correspondence) adjacency matrix
adjacency()	sprof to (correspondence) adjacency matrix

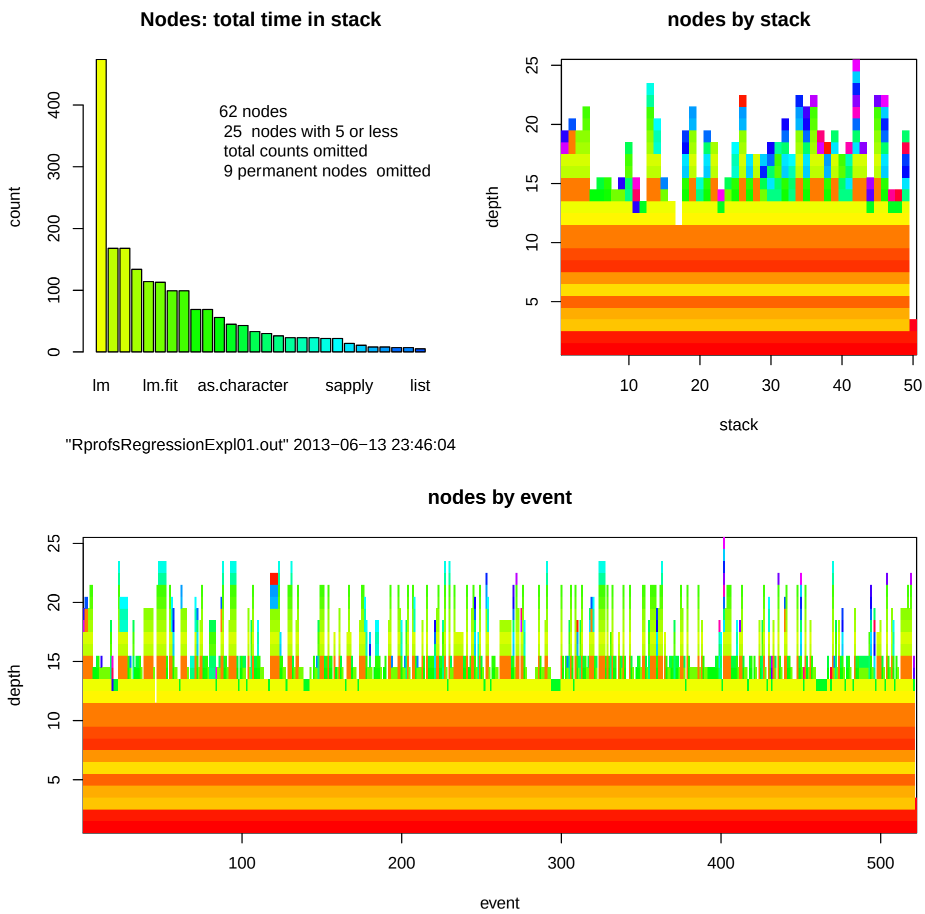
We now can go beyond node level.

This is what we get for free from the node information on our three levels: node, stack, and profile.

ToDo: check and stabilize color linking

Input

```
#8 rainbow
sprof01$nodes$icol <- freqrank01
shownodes(sprof01, col=rainbow(62))
```



The obvious message is that if seen by stack level, there are different structures. Profiling usually takes place in a framework. So at the base of the stacks, we find entries that are (almost) persistent. Then usually we have some few steps where the algorithm splits, and then we have the finer details. These can be identified using information on the stack level, but of course they are not visible on the node or edge level.

Not so often, but a frequent phenomenon is to have some "burn in" or "fade out". To identify this, we need to look at the profile level.

ToDo: example

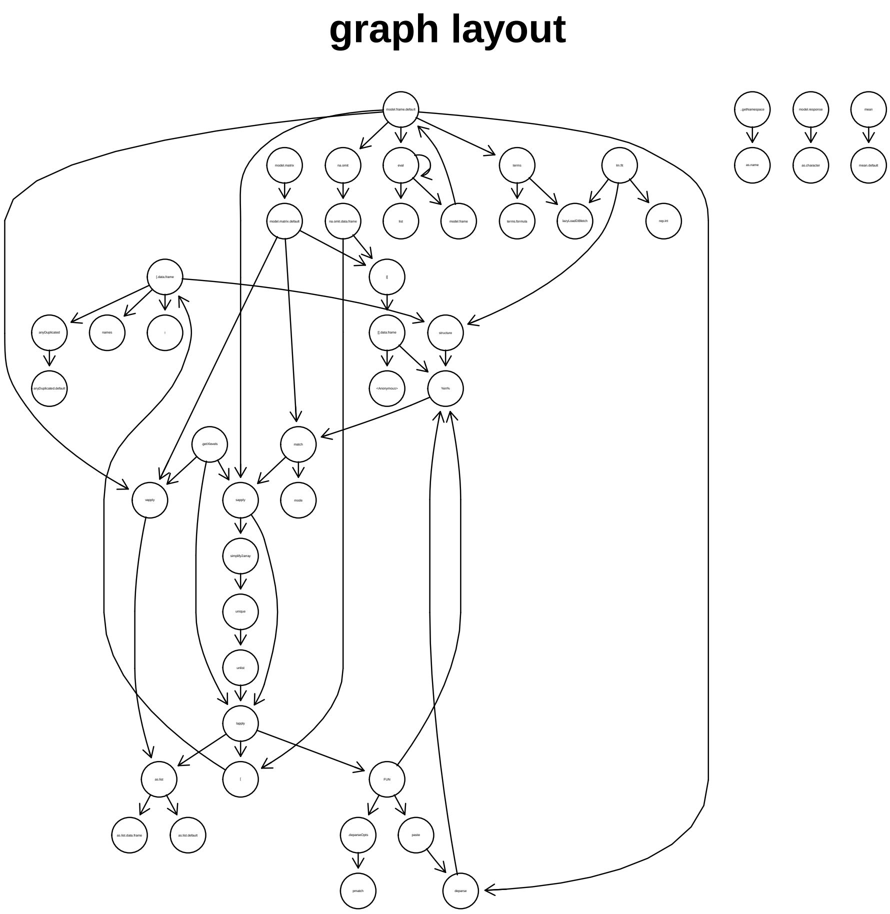
At a closer look, we may find stack patterns (maybe marked by specific nodes) that indicate administrative intervention and rather should be handled as separators between distinct profiles rather than as part of the general dynamics. Stable framework effect can be detected automatically. "burn in" or "fade out' may need a closer look, and special stacks need and individual inspection on low frequency stacks.

ToDo: colours. re-colour. Propagate colour to graph.

2.2. **The free lunch.** What you have seen so far is what you get for free when using package *sprof*. If your want to wrap up the information and look at it from a graph point of view, here is just one example. More are in section 5 on page 65. But before changing to the graph perspective, we recommend to see the next sections, not to skip them. The preview, at his point, taking package ***graph*** as an example:

Input

```
#8
library(graph)
sprof01adjNEL <- as(adjacency(sprof01),"graphNEL")
plot(sprof01adjNEL,  main="graph layout example", cex.main=2)
rm(sprof01adjNEL);
# detach("package:graph")
#! sorry. still needed by Rgraphviz -- clean up
```



2.3. **Cheap thrills.** Before starting additional inspection, the data better be trimmed.

Input

```
sprof02 <- sprof01
basetrim <- 13
sprof02$stacks$nodes <- sapply(sprof02$stacks$nodes,
```

```
function (x){if (length(x)> basetrim) x[-(1:basetrim)] })
```

ToDo: handle empty stacks and zero counts gracefully

ToDo: add ?? function

At this point, it is a decision whether to adapt the timing information, or keep the original information. Since this decision does affect the structural information, it is not critical. But analysis is easier if unused nodes are eliminated.

Input
```
summary(sprof02)
```

Output
```
$id
[1] "Profile Summary Thu Jul 18 22:06:07 2013"

$len
[1] 522

$uniquestacks
[1] 50

$nr_runs
[1] 396

$nrstacks
[1] 50

$stacklength
[1]  3 25

$nrnodesperlevel
 [1] 10 11  9  9 15  8  7  5  7  2  1  1

                      shortname root leaf self.time  self.pct
!                             !    - LEAF         2  0.383142
..getNamespace             ..gN    -    -         0  0.000000
.deparseOpts               .dpO    - LEAF         2  0.383142
.getXlevels                .gtX    -    -         0  0.000000
[                             [    -    -         0  0.000000
[.data.frame               [.d.    - LEAF        57 10.919540
[[                           [[    -    -         0  0.000000
[[.data.frame              [[..    - LEAF         1  0.191571
%in%                       %in%    - LEAF         1  0.191571
<Anonymous>                <An>    - LEAF         6  1.149425
$                             $    - LEAF         1  0.191571
anyDuplicated              anyD    - LEAF         1  0.191571
anyDuplicated.default      anD.    - LEAF        22  4.214559
as.character               as.c    - LEAF        43  8.237548
as.list                    as.l    -    -         0  0.000000
as.list.data.frame         a...    - LEAF        22  4.214559
as.list.default            as..    - LEAF         1  0.191571
as.name                    as.n    - LEAF         1  0.191571
coef                       coef    - LEAF         1  0.191571
```

```
deparse                    dprs    - LEAF          1  0.191571
doTryCatch                 dTrC    -    -          0  0.000000
eval                       eval    - LEAF          1  0.191571
evalFunc                   evlF    -    -          0  0.000000
file                       file    - LEAF          1  0.191571
FUN                         FUN    - LEAF          1  0.191571
lapply                     lppl    - LEAF          2  0.383142
lazyLoadDBfetch            lLDB    - LEAF          2  0.383142
list                       list    - LEAF          5  0.957854
lm                           lm    - LEAF         42  8.045977
lm.fit                     lm.f    - LEAF         87 16.666667
match                      mtch    - LEAF          1  0.191571
mean                       mean    -    -          0  0.000000
mean.default               mn.d    - LEAF          2  0.383142
mode                       mode    - LEAF          2  0.383142
model.frame               mdl.f    -    -          0  0.000000
model.frame.default      mdl.f.    - LEAF         12  2.298851
model.matrix              mdl.m    -    -          0  0.000000
model.matrix.default     mdl.m.    - LEAF         51  9.770115
model.response            mdl.r    - LEAF         13  2.490421
na.omit                    n.mt    - LEAF         20  3.831418
na.omit.data.frame         n...    - LEAF         26  4.980843
names                      nams    - LEAF          2  0.383142
NCOL                       NCOL    - LEAF          1  0.191571
paste                      past    -    -          0  0.000000
pmatch                     pmtc    - LEAF          2  0.383142
rep.int                    rp.n    - LEAF          7  1.340996
sapply                     sppl    - LEAF          1  0.191571
simplify2array             smp2    -    -          0  0.000000
structure                  strc    - LEAF         32  6.130268
summary                    smmr    -    -          0  0.000000
summary.lm                 smm.    - LEAF         40  7.662835
Sweave                     Swev ROOT    -          0  0.000000
terms                      trms    -    -          0  0.000000
terms.formula              trm.    - LEAF          1  0.191571
try                         try    -    -          0  0.000000
tryCatch                   tryC    -    -          0  0.000000
tryCatchList               trCL    -    -          0  0.000000
tryCatchOne                trCO    -    -          0  0.000000
unique                     uniq    - LEAF          3  0.574713
unlist                     unls    -    -          0  0.000000
vapply                     vppl    - LEAF          3  0.574713
withVisible                wthV    -    -          0  0.000000
                      total.time total.pct
!                              2  0.383142
..getNamespace                 1  0.191571
.deparseOpts                   4  0.766284
.getXlevels                   26  4.980843
[                             99 18.965517
[.data.frame                  99 18.965517
[[                             8  1.532567
[[.data.frame                  8  1.532567
%in%                           4  0.766284
```

```
<Anonymous>                  6  1.149425
$                            1  0.191571
anyDuplicated               23  4.406130
anyDuplicated.default       22  4.214559
as.character                43  8.237548
as.list                     23  4.406130
as.list.data.frame          22  4.214559
as.list.default              1  0.191571
as.name                      1  0.191571
coef                         1  0.191571
deparse                      2  0.383142
doTryCatch                   0  0.000000
eval                       168 32.183908
evalFunc                     0  0.000000
file                         0  0.000000
FUN                          7  1.340996
lapply                      30  5.747126
lazyLoadDBfetch              2  0.383142
list                         5  0.957854
lm                           0  0.000000
lm.fit                     113 21.647510
match                       11  2.107280
mean                         2  0.383142
mean.default                 2  0.383142
mode                         2  0.383142
model.frame                168 32.183908
model.frame.default        168 32.183908
model.matrix                69 13.218391
model.matrix.default        69 13.218391
model.response              56 10.727969
na.omit                    134 25.670498
na.omit.data.frame         114 21.839080
names                        2  0.383142
NCOL                         1  0.191571
paste                        1  0.191571
pmatch                       2  0.383142
rep.int                      7  1.340996
sapply                      14  2.681992
simplify2array               4  0.766284
structure                   33  6.321839
summary                      0  0.000000
summary.lm                   0  0.000000
Sweave                       0  0.000000
terms                        2  0.383142
terms.formula                1  0.191571
try                          0  0.000000
tryCatch                     0  0.000000
tryCatchList                 0  0.000000
tryCatchOne                  0  0.000000
unique                       4  0.766284
unlist                       1  0.191571
vapply                      23  4.406130
withVisible                  0  0.000000
```

Input
```r
sprof02 <- updateRprof(sprof02)
summary(sprof02)
```

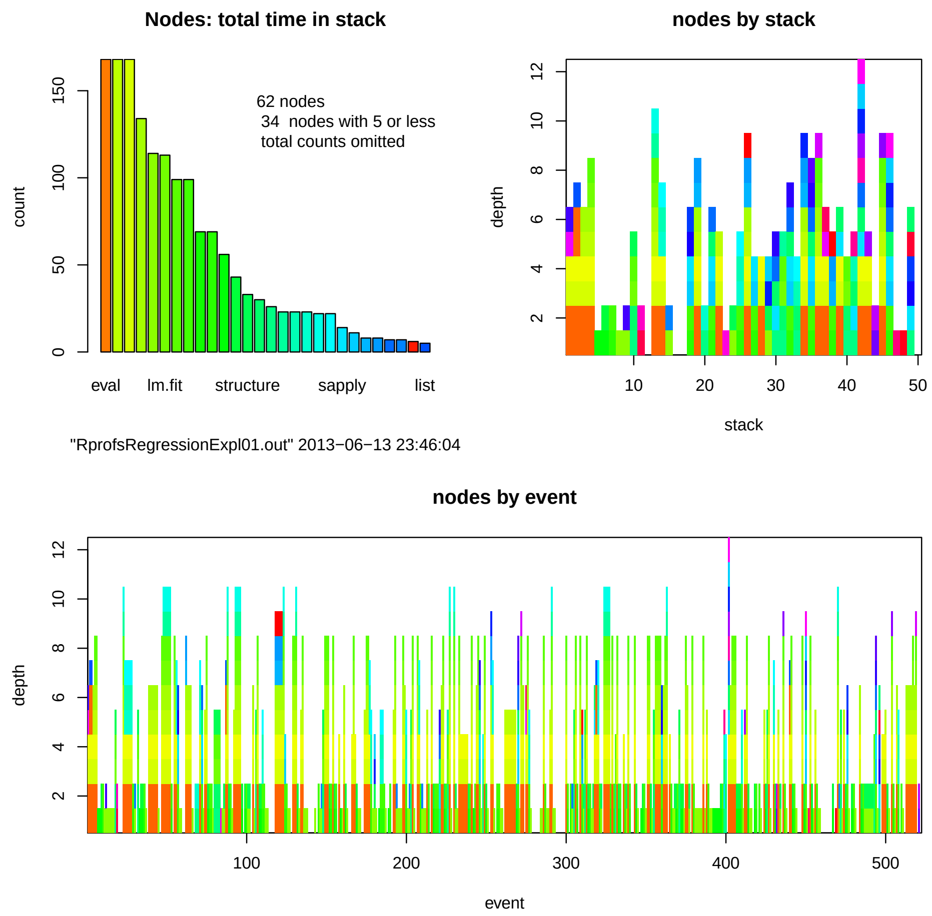
Output
```
$id
[1] "Profile Summary Thu Jul 18 22:06:07 2013"

$len
[1] 522

$uniquestacks
[1] 50

$nr_runs
[1] 396

$nrstacks
[1] 50

$stacklength
[1]  0 12

$nrnodesperlevel
 [1] 10 11  9  9 15  8  7  5  7  2  1  1

                      shortname root leaf self.time   self.pct
!                             !    - LEAF         2   0.383142
..getNamespace              ..gN ROOT    -        0   0.000000
.deparseOpts                .dpO    - LEAF        2   0.383142
.getXlevels                 .gtX ROOT    -        0   0.000000
[                              [    -    -        0   0.000000
[.data.frame                [.d.    - LEAF       57  10.919540
[[                            [[    -    -        0   0.000000
[[.data.frame               [[..    - LEAF        1   0.191571
%in%                        %in%    - LEAF        1   0.191571
<Anonymous>                 <An>    - LEAF        6   1.149425
$                              $ ROOT LEAF        1   0.191571
anyDuplicated               anyD    - LEAF        1   0.191571
anyDuplicated.default       anD.    - LEAF       22   4.214559
as.character                as.c    - LEAF       43   8.237548
as.list                     as.l    -    -        0   0.000000
as.list.data.frame          a...    - LEAF       22   4.214559
as.list.default             as..    - LEAF        1   0.191571
as.name                     as.n    - LEAF        1   0.191571
coef                        coef ROOT LEAF        1   0.191571
deparse                     dprs    - LEAF        1   0.191571
doTryCatch                  dTrC    -    -        0   0.000000
eval                        eval ROOT    -        0   0.000000
evalFunc                    evlF    -    -        0   0.000000
file                        file    -    -        0   0.000000
FUN                          FUN    - LEAF        1   0.191571
lapply                      lppl    - LEAF        2   0.383142
```

```
lazyLoadDBfetch           lLDB    - LEAF        2  0.383142
list                      list    - LEAF        5  0.957854
lm                          lm    -    -        0  0.000000
lm.fit                    lm.f ROOT LEAF       87 16.666667
match                     mtch    - LEAF        1  0.191571
mean                      mean ROOT    -        0  0.000000
mean.default              mn.d    - LEAF        2  0.383142
mode                      mode    - LEAF        2  0.383142
model.frame              mdl.f    -    -        0  0.000000
model.frame.default     mdl.f.    - LEAF       12  2.298851
model.matrix             mdl.m ROOT    -        0  0.000000
model.matrix.default    mdl.m.    - LEAF       51  9.770115
model.response           mdl.r ROOT LEAF       13  2.490421
na.omit                   n.mt    - LEAF       20  3.831418
na.omit.data.frame        n...    - LEAF       26  4.980843
names                     nams    - LEAF        2  0.383142
NCOL                      NCOL ROOT LEAF        1  0.191571
paste                     past    -    -        0  0.000000
pmatch                    pmtc    - LEAF        2  0.383142
rep.int                   rp.n    - LEAF        7  1.340996
sapply                    sppl    - LEAF        1  0.191571
simplify2array            smp2    -    -        0  0.000000
structure                 strc    - LEAF       32  6.130268
summary                   smmr    -    -        0  0.000000
summary.lm                smm.    -    -        0  0.000000
Sweave                    Swev    -    -        0  0.000000
terms                     trms    -    -        0  0.000000
terms.formula             trm.    - LEAF        1  0.191571
try                        try    -    -        0  0.000000
tryCatch                  tryC    -    -        0  0.000000
tryCatchList              trCL    -    -        0  0.000000
tryCatchOne               trCO    -    -        0  0.000000
unique                    uniq    - LEAF        3  0.574713
unlist                    unls    -    -        0  0.000000
vapply                    vppl    - LEAF        3  0.574713
withVisible               wthV    -    -        0  0.000000
                      total.time total.pct
!                              2  0.383142
..getNamespace                 1  0.191571
.deparseOpts                   4  0.766284
.getXlevels                   26  4.980843
[                             99 18.965517
[.data.frame                  99 18.965517
[[                             8  1.532567
[[.data.frame                  8  1.532567
%in%                           4  0.766284
<Anonymous>                    6  1.149425
$                              1  0.191571
anyDuplicated                 23  4.406130
anyDuplicated.default         22  4.214559
as.character                  43  8.237548
as.list                       23  4.406130
as.list.data.frame            22  4.214559
```

```
as.list.default              1  0.191571
as.name                      1  0.191571
coef                         1  0.191571
deparse                      2  0.383142
doTryCatch                   0  0.000000
eval                       168 32.183908
evalFunc                     0  0.000000
file                         0  0.000000
FUN                          7  1.340996
lapply                      30  5.747126
lazyLoadDBfetch              2  0.383142
list                         5  0.957854
lm                           0  0.000000
lm.fit                     113 21.647510
match                       11  2.107280
mean                         2  0.383142
mean.default                 2  0.383142
mode                         2  0.383142
model.frame                168 32.183908
model.frame.default        168 32.183908
model.matrix                69 13.218391
model.matrix.default        69 13.218391
model.response              56 10.727969
na.omit                    134 25.670498
na.omit.data.frame         114 21.839080
names                        2  0.383142
NCOL                         1  0.191571
paste                        1  0.191571
pmatch                       2  0.383142
rep.int                      7  1.340996
sapply                      14  2.681992
simplify2array               4  0.766284
structure                   33  6.321839
summary                      0  0.000000
summary.lm                   0  0.000000
Sweave                       0  0.000000
terms                        2  0.383142
terms.formula                1  0.191571
try                          0  0.000000
tryCatch                     0  0.000000
tryCatchList                 0  0.000000
tryCatchOne                  0  0.000000
unique                       4  0.766284
unlist                       1  0.191571
vapply                      23  4.406130
withVisible                  0  0.000000
```

Input

```
#8
shownodes(sprof02)
```



ToDo: trimexample

2.3.1. *Trimming.*

Input
```
trimstacks <- function(sprof, level){
lapply(sprof$stacks$nodes, function(x) {x[-(1:level)]})
}
```

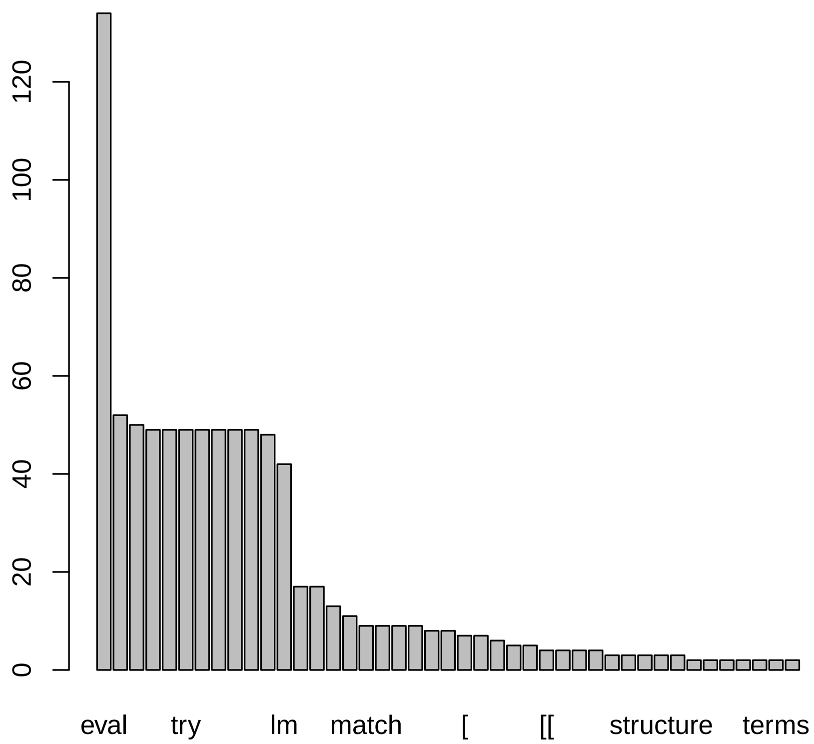
Input
```
sprof01Tr <- trimstacks(sprof01, 11)
#profile_nodesTr <- profiles_matrix(sprof01Tr)
#image(x=1:ncol(profile_nodesTr),y=1:nrow(profile_nodesTr), t(profile_nodesTr),xlab="event", ylab='
```

Input
```
nodefreq <- rep(0,length(sprof01$nodes$name))
for (i in (1:length(sprof01$stacks$nodes))){
        nodefreq <- nodefreq +
                table( factor(sprof01$stacks$nodes[[i]],
```

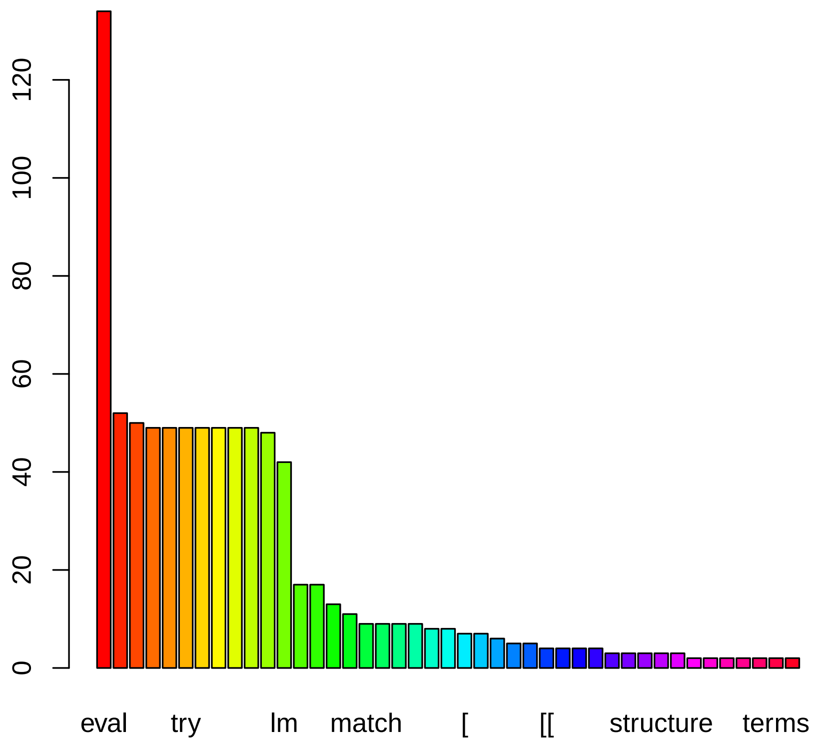
```
                        levels <- 1:length(sprof01$nodes$name),
                        ordered=FALSE))
        }
names(nodefreq) <- sprof01$nodes$name
```

Top frequent nodes.

Input
```
ndf <- nodefreq[nodefreq>1]
ondf <- order(ndf,decreasing=TRUE)
barplot(ndf[ondf])
```

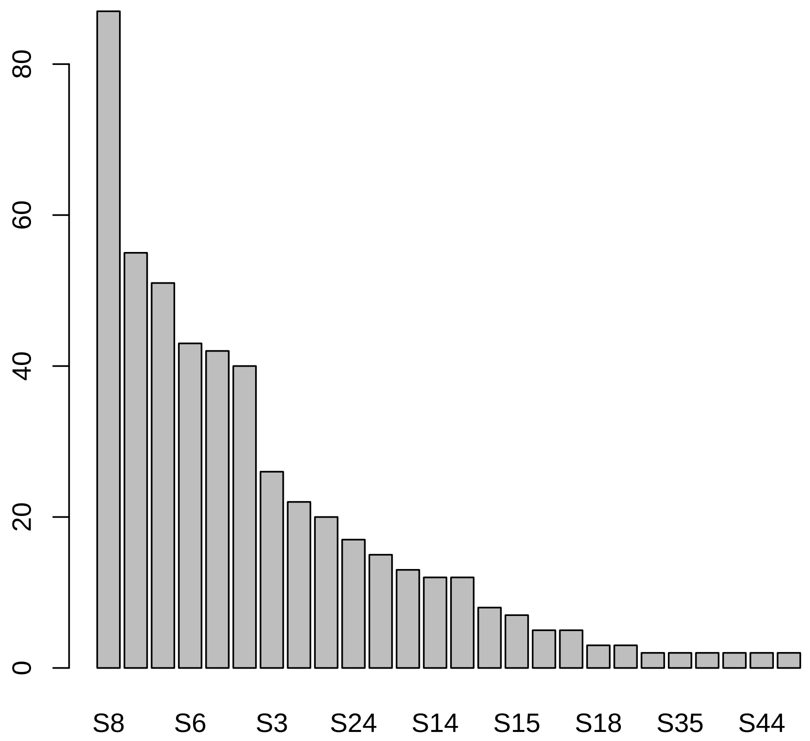


Input
```
barplot(ndf[ondf], col=rainbow(length(ondf)))
```

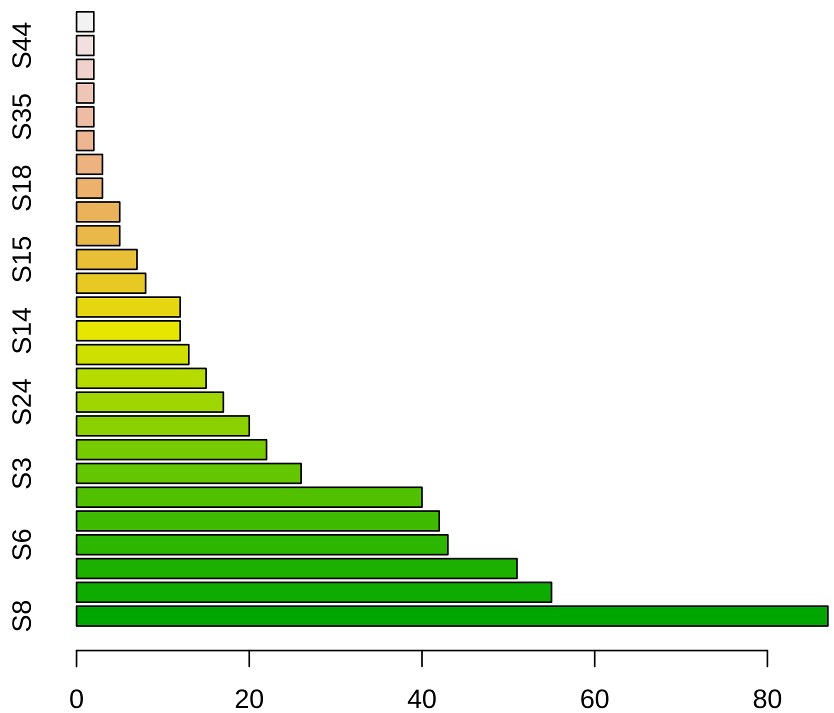


Top frequent stacks.

Input
```
x <- sprof01
xsrc <- as.matrix(x$stacks$refcount)
rownames(xsrc) <- rownames(xsrc, do.NULL=FALSE, prefix="S")
#stf <- x$stacks$refcount[x$stacks$refcount>1]
#names(stf) <-  x$stacks$shortname[x$stacks$refcount>1]
stf <- xsrc[xsrc>1]
names(stf) <- rownames(xsrc)[xsrc>1]
ostf <- order(stf,decreasing=TRUE)
barplot(stf[ostf])
```

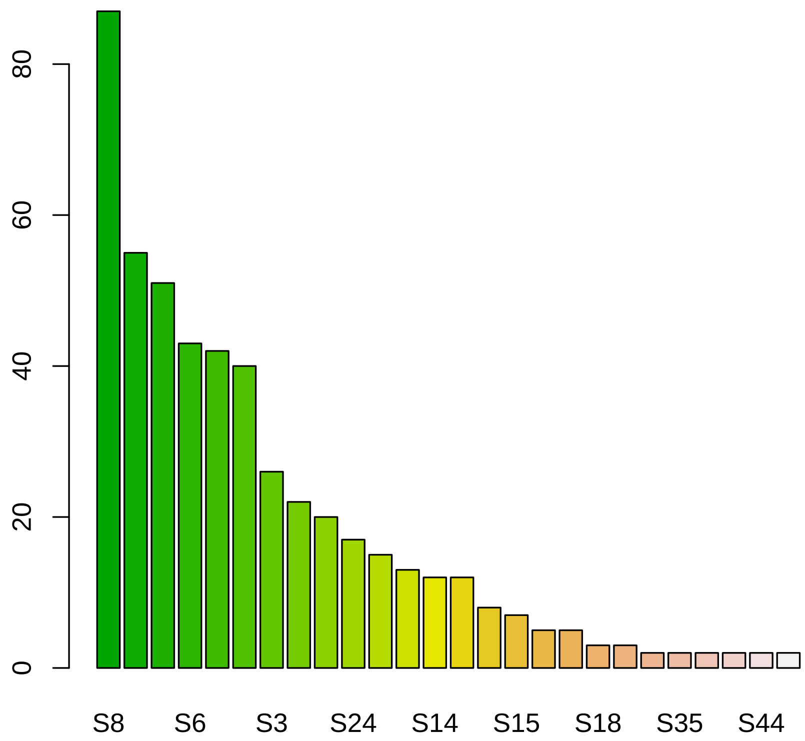


Input
```
barplot(stf[ostf], col=terrain.colors(length(ostf)), horiz=TRUE)
```



Input

```
barplot(stf[ostf], col=terrain.colors(length(ostf)))
```



There is no statistics on profiles. Profiling are our elementary data. However we can link to our derived data to get a more informative display. For example, going one step back we can encode stacks and use these color codes in the display of a profile.

Or going two steps back, we can encode nodes in color, giving colored stacks, and use these in the display of profile data.

ToDo: function addnode to be added

2.3.2. *Surgery.* Looking at nodes gives you a point-wise horizon. Looking at edges gives you a one step horizon. The stacks give a wider horizon, typically a step size of 10 or more. The stacks we get from R have peculiarities, and we can handle with this broader perspective. These are not relevant if we look point-wise, but may become dominating if we try to get a global picture. We take a look ahead (details to come in section 5 on page 65 and nave a preview how our example is represented as a graph. Left is the original graph as recovered from the edge information, right the graph after we have cut off the scaffold effects. Control structures may be represented in R as function, and these may lead to concentration points. Using information from the stacks, we can avoid these by introducing substitute nodes on the stack level. For example, `"[" "lapply" ".getXlevels" -> "<.getXlevels_[>"`

ToDo: cut next level

ToDo: Implement. Currently handled on source=text level

ToDo: collapse. cut tails

"as.list" "vapply" "model.frame.default" -> "<model_as.list>"

ToDo: implement

"as.list" "vapply" "model.matrix.default" -> "<model_matrix_as.list>"

Input
```
newchopnode <- function(nodenames, chop) {
tmpname <- paste("<",as.character(nodenames[chop]),">")
# chec for existing.
# add if necessary
tmpname
}
chopstack <- function(x , chop, replacement)
{
# is chop in x`
# y: cut x.
# merge x <- head + replacement + tiail
return(x)
}
```

ToDo: needs serious revision

2.4. **Run length.** For a visual inspection, runs of the same node and level in the profile are easily perceived. For an analytical inspection, we have to reconstruct the runs from the data. In stacks, runs are organized hierarchically. On the root level, runs are just ordinary runs. On the next levels, runs have to be defined given (within) the previous runs. So we need a recursive version of *rle*, applied to the profile information. This gives a detailed information about the presence time of each node, by stack level.

Input
```
profile_nodes <- profiles_matrix(sprof02)
profile_nodes_rle<- rrle(profile_nodes)
str(profile_nodes_rle)
```

Output
```
List of 12
 $ :List of 2
  ..$ lengths: int [1:361] 6 3 1 7 1 1 1 1 1 6 ...
  ..$ values : int [1:361] 22 39 37 30 4 2 NA NA NA 22 ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:407] 6 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:407] 22 NA NA 14 38 NA 27 NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:427] 6 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:427] 35 NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:427] 6 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:427] 36 NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:450] 1 2 3 1 1 1 1 1 1 1 ...
```

```
  ..$ values : int [1:450] 53 22 40 NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:466] 1 2 3 1 1 1 1 1 1 1 ...
  ..$ values : int [1:466] 27 22 41 NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:489] 1 2 1 2 1 1 1 1 1 1 ...
  ..$ values : int [1:489] NA 28 NA 5 NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:494] 1 1 1 1 2 1 1 1 1 1 ...
  ..$ values : int [1:494] NA NA NA NA 6 NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:508] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:508] NA NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:512] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:512] NA NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:522] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:522] NA NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
 $ :List of 2
  ..$ lengths: int [1:522] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ values : int [1:522] NA NA NA NA NA NA NA NA NA NA ...
  ..- attr(*, "class")= chr "rle"
```

On a given stack level, the run length is the best information on the time used per call, and the run count of a node is the best information on the number of calls. So this is a prime starting point for in-depth analysis.

ToDo: keep as factor. This is a saprse cube with margins node, stack level, run length. Nodes are mostly concentrated on few levels. **ToDo:** Warning: data structure still under discussion

Input

```
# side effect: NAs are removed
profile_nodes_rlet <- lapply(profile_nodes_rle,
        function(x) table(x,dnn=c("run length","node")) )
invisible(lapply(profile_nodes_rlet,
function(x) print.table(x,zero.print = ".") ))
```

Output

```
          node
run length  2  4 11 19 22 30 32 37 39 43
         1  1 17  1  1 40 46  2 55 35  1
         2  .  1  .  . 17 18  .  4  3  .
         3  .  1  .  .  6  3  .  2  3  .
         4  .  1  .  .  4  1  .  .  .  .
         5  .  .  .  .  2  1  .  .  .  .
         6  .  .  .  .  6  1  .  .  1  .
         7  .  .  .  .  2  1  .  .  .  .
          node
```

```
run length 14 18 22 26 27 33 38 46 47 49 61
         1 34  1 40 16  1  2 55  7  3 10  .
         2  3  . 17  .  .  .  4  .  .  1  .
         3  1  .  6  .  .  .  2  .  .  .  1
         4  .  .  4  1  .  .  .  .  .  .  .
         5  .  .  2  .  .  .  .  .  .  .  .
         6  .  .  6  .  .  .  .  .  .  1  .
         7  .  .  2  .  .  .  .  .  .  .  .
          node
run length  5  7  9 15 26 31 35 48 61
         1 14  1  1  2  2  9 40  1  6
         2  .  .  .  .  .  . 17  .  1
         3  .  .  .  .  .  .  6  .  .
         4  1  .  .  .  .  .  4  .  .
         5  .  .  .  .  .  .  2  .  .
         6  .  .  .  .  .  .  6  .  .
         7  .  .  .  .  .  .  2  .  .
          node
run length  6  8 15 16 25 31 36 47 59
         1 14  1  7  2  1  1 40  9  1
         2  .  .  1  .  .  . 17  .  .
         3  .  .  .  .  .  .  6  .  .
         4  1  .  .  .  .  .  4  .  .
         5  .  .  .  .  .  .  2  .  .
         6  .  .  .  .  .  .  6  .  .
         7  .  .  .  .  .  .  2  .  .
          node
run length  3 10 12 16 17 20 22 26 40 47 48 49 53 60 61
         1  1  1  1  6  1  1  3  5 46  2  3 11  2  1  7
         2  .  .  .  1  .  .  1  . 10  .  .  .  .  .  .
         3  .  .  .  .  .  .  .  .  5  .  .  .  .  .  .
         4  .  .  .  .  .  .  .  .  4  .  .  1  .  .  .
         5  .  .  .  .  .  .  .  .  1  .  .  .  .  .  1
         6  .  .  .  .  .  .  .  .  3  .  .  .  .  .  .
         7  .  .  .  .  .  .  .  .  2  .  .  .  .  .  .
          node
run length 15 22 25 26 27 41 54 59
         1  7  3  5  3  1 43  1  3
         2  .  1  .  .  .  7  .  .
         3  .  .  .  .  .  4  .  .
         4  .  .  .  .  .  5  .  .
         5  1  .  .  .  .  .  .  .
         6  .  .  .  .  .  3  .  .
         7  .  .  .  .  .  1  .  .
          node
run length  3  5  7  9 16 25 28
         1  3 46  2  1  7  1  3
         2  .  4  .  .  .  .  1
         3  .  3  .  .  .  .  .
         4  .  3  .  .  .  .  .
         5  .  .  1  .  1  .  .
         6  .  1  .  .  .  .  .
          node
```

```
run length  6  8 31 44 45
         1 46  2  1  1  2
         2  4  .  .  .  .
         3  3  .  .  .  .
         4  3  .  .  .  .
         5  .  1  .  .  .
         6  1  .  .  .  .
          node
run length 1 9 10 12 20 34 42
         1 2 1  .  9  1  1  2
         4 . .  .  2  .  .  .
         5 . .  1  1  .  .  .
          node
run length 9 13
         1 1  9
         4 .  2
         5 .  1
          node
run length 31
         1  1
          node
run length 34
         1  1
```

This is a poor first attempt to tame *profile_nodes_rlet*.

ToDo: some mess in the code below – not working

ToDo: replace by decent vector/array based implementation

ToDo: add names for node dimension

Input

```
maxnode <-0
maxlen <-0
maxlevel <-length(profile_nodes_rle)
for (lev in (1:maxlevel) ) {
        proflev <- profile_nodes_rle[[lev]]
        if (!is.null(proflev)) {
                maxn <- max(proflev$values, na.rm=TRUE)
                if (maxn>maxnode) maxnode <- maxn
                maxl <- max(proflev$lengths, na.rm=TRUE)
                if (maxl>maxlen) maxlen <- maxl
                # cat("Level ",lev,maxn," Length:",maxl,"\n")
        }
}
## collapse profile_nodes_rle to 3d array. Allocate memory first.
profile_nodes_rlearray <- array(0,
        dim=c(maxnode,length(profile_nodes_rle), maxlen),
        dimnames= list("node"=sprof02$nodes$name[1:maxnode], "level"=1:length(profile_nodes_rle), '
str(profile_nodes_rlearray)
```

Output

```
num [1:61, 1:12, 1:7] 0 0 0 0 0 0 0 0 0 0 ...
- attr(*, "dimnames")=List of 3
 ..$ node      : chr [1:61] "!" "..getNamespace" ".deparseOpts" ".getXlevels" ...
 ..$ level     : chr [1:12] "1" "2" "3" "4" ...
 ..$ run_length: chr [1:7] "1" "2" "3" "4" ...
```

Input
```
for (lev in (1:maxlevel) ) {
        proflev <- profile_nodes_rle[[lev]]
        if (!is.null(proflev)) {
                for (j in (1: length(proflev$lengths))){
                        if (!is.na(proflev$values[j])){
                                profile_nodes_rlearray[proflev$values[j],lev,proflev$lengths[j]] <-
                                profile_nodes_rlearray[proflev$values[j],lev,proflev$lengths[j]] +1
                        #cat(lev,j,":",proflev$values[j],lev,proflev$lengths[j],"\n")
                        }#if (!is.na
                }#for j
        }
}
```

ToDo: add marginals and conditionals. Provide function node_summary.

This allows us to extract marginal from `provlev[ node, level, run length]`.

Input
```
node=41
cat(sprof02$nodes$name[node],"\n")
```

Output
```
41
```

Input
```
nn <- profile_nodes_rlearray[node, , ]
#dimnames(nn)<- list( "level", "run_length")
nn
```

Output
```
     run_length
level  1 2 3 4 5 6 7
   1   0 0 0 0 0 0 0
   2   0 0 0 0 0 0 0
   3   0 0 0 0 0 0 0
   4   0 0 0 0 0 0 0
   5   0 0 0 0 0 0 0
   6  43 7 4 5 0 3 1
   7   0 0 0 0 0 0 0
   8   0 0 0 0 0 0 0
   9   0 0 0 0 0 0 0
   10  0 0 0 0 0 0 0
   11  0 0 0 0 0 0 0
   12  0 0 0 0 0 0 0
```

Input
```
print.table(addmargins(nn), zero.print = ".")
```

Output
```
     run_length
level  1  2  3  4  5  6  7 Sum
  1    .  .  .  .  .  .  .   .
  2    .  .  .  .  .  .  .   .
  3    .  .  .  .  .  .  .   .
```

```
4    .  .  .  .  .  .  .   .
5    .  .  .  .  .  .  .   .
6   43  7  4  5  .  3  1  63
7    .  .  .  .  .  .  .   .
8    .  .  .  .  .  .  .   .
9    .  .  .  .  .  .  .   .
10   .  .  .  .  .  .  .   .
11   .  .  .  .  .  .  .   .
12   .  .  .  .  .  .  .   .
Sum 43  7  4  5  .  3  1  63
```

ToDo: rescale to application scale

ToDo: replace sum by weighted sum

ToDo: allow sorting, e.g. by marginals

Input

```
mt <- margin.table(profile_nodes_rlearray, margin = c(1,3))
amt <- addmargins(mt)
amt <- amt[amt[,"Sum"]>0,]
print.table(amt, zero.print = ".")
```

Output

```
                       run_length
node                     1    2    3    4    5    6    7  Sum
  !                      2    .    .    .    .    .    .    2
  ..getNamespace         1    .    .    .    .    .    .    1
  .deparseOpts           4    .    .    .    .    .    .    4
  .getXlevels           17    1    1    1    .    .    .   20
  [                     60    4    3    4    .    1    .   72
  [.data.frame          60    4    3    4    .    1    .   72
  [[                     3    .    .    .    1    .    .    4
  [[.data.frame          3    .    .    .    1    .    .    4
  %in%                   4    .    .    .    .    .    .    4
  <Anonymous>            1    .    .    .    1    .    .    2
  $                      1    .    .    .    .    .    .    1
  anyDuplicated         10    .    .    2    1    .    .   13
  anyDuplicated.default  9    .    .    2    1    .    .   12
  as.character          34    3    1    .    .    .    .   38
  as.list               16    1    .    .    1    .    .   18
  as.list.data.frame    15    1    .    .    1    .    .   17
  as.list.default        1    .    .    .    .    .    .    1
  as.name                1    .    .    .    .    .    .    1
  coef                   1    .    .    .    .    .    .    1
  deparse                2    .    .    .    .    .    .    2
  eval                  86   36   12    8    4   12    4  162
  FUN                    7    .    .    .    .    .    .    7
  lapply                26    .    .    1    .    .    .   27
  lazyLoadDBfetch        2    .    .    .    .    .    .    2
  list                   3    1    .    .    .    .    .    4
  lm.fit                46   18    3    1    1    1    1   71
  match                 12    .    .    .    .    .    .   12
  mean                   2    .    .    .    .    .    .    2
  mean.default           2    .    .    .    .    .    .    2
  mode                   2    .    .    .    .    .    .    2
  model.frame           40   17    6    4    2    6    2   77
  model.frame.default   40   17    6    4    2    6    2   77
  model.matrix          55    4    2    .    .    .    .   61
```

```
  model.matrix.default     55    4    2    .    .    .    .   61
  model.response           35    3    3    .    .    1    .   42
  na.omit                  46   10    5    4    1    3    2   71
  na.omit.data.frame       43    7    4    5    .    3    1   63
  names                     2    .    .    .    .    .    .    2
  NCOL                      1    .    .    .    .    .    .    1
  paste                     1    .    .    .    .    .    .    1
  pmatch                    2    .    .    .    .    .    .    2
  rep.int                   7    .    .    .    .    .    .    7
  sapply                   14    .    .    .    .    .    .   14
  simplify2array            4    .    .    .    .    .    .    4
  structure                21    1    .    1    .    1    .   24
  terms                     2    .    .    .    .    .    .    2
  terms.formula             1    .    .    .    .    .    .    1
  unique                    4    .    .    .    .    .    .    4
  unlist                    1    .    .    .    .    .    .    1
  vapply                   13    1    1    .    1    .    .   16
  Sum                     820  133   52   41   18   35   12 1111
```

These are some attempts to recover the factor structures.

Input

```
 xfi <- levels(sprof02$nodes$name)
 profile_nodes_rlefac <- lapply(profile_nodes_rle, function(xl) {xl$values <- factor(xl$values, leve
 profile_nodes_rletfac <- lapply(profile_nodes_rle,
         function(x) table(x,dnn=c("run length","node")) ) #factors lost again
 colnames(profile_nodes_rletfac[[1]]) <- sprof02$nodes$name[ as.integer(colnames(profile_nodes_rletf
 profile_nodes_rletfac1 <- lapply(profile_nodes_rletfac,
         function(xl) {colnames(xl) <- sprof02$nodes$name[ as.integer(colnames(xl))];
         xl} )
 invisible(lapply(profile_nodes_rletfac1,
 function(x) print.table(t(x),zero.print = ".") ))
```

Output

```
      run length
node    1  2  3  4  5  6  7
  <NA>  1  .  .  .  .  .  .
  <NA> 17  1  1  1  .  .  .
  <NA>  1  .  .  .  .  .  .
  <NA>  1  .  .  .  .  .  .
  <NA> 40 17  6  4  2  6  2
  <NA> 46 18  3  1  1  1  1
  <NA>  2  .  .  .  .  .  .
  <NA> 55  4  2  .  .  .  .
  <NA> 35  3  3  .  .  1  .
  <NA>  1  .  .  .  .  .  .
                       run length
node                     1  2  3  4  5  6  7
  as.character          34  3  1  .  .  .  .
  as.name                1  .  .  .  .  .  .
  eval                  40 17  6  4  2  6  2
  lapply                16  .  .  1  .  .  .
  lazyLoadDBfetch        1  .  .  .  .  .  .
```

```
  mean.default          2  .  .  .  .  .  .  .
  model.matrix.default 55  4  2  .  .  .  .  .
  rep.int               7  .  .  .  .  .  .  .
  sapply                3  .  .  .  .  .  .  .
  structure            10  1  .  .  .  1  .
  vapply                .  .  1  .  .  .  .
                run length
node              1  2  3  4  5  6  7
  [              14  .  .  1  .  .  .
  [[              1  .  .  .  .  .  .
  %in%            1  .  .  .  .  .  .
  as.list         2  .  .  .  .  .  .
  lapply          2  .  .  .  .  .  .
  match           9  .  .  .  .  .  .
  model.frame    40 17  6  4  2  6  2
  simplify2array  1  .  .  .  .  .  .
  vapply          6  1  .  .  .  .  .
                     run length
node                   1  2  3  4  5  6  7
  [.data.frame        14  .  .  1  .  .  .
  [[.data.frame        1  .  .  .  .  .  .
  as.list              7  1  .  .  .  .  .
  as.list.data.frame   2  .  .  .  .  .  .
  FUN                  1  .  .  .  .  .  .
  match                1  .  .  .  .  .  .
  model.frame.default 40 17  6  4  2  6  2
  sapply               9  .  .  .  .  .  .
  unique               1  .  .  .  .  .  .
                    run length
node                  1  2  3  4  5  6  7
  .deparseOpts        1  .  .  .  .  .  .
  <Anonymous>         1  .  .  .  .  .  .
  anyDuplicated       1  .  .  .  .  .  .
  as.list.data.frame  6  1  .  .  .  .  .
  as.list.default     1  .  .  .  .  .  .
  deparse             1  .  .  .  .  .  .
  eval                3  1  .  .  .  .  .
  lapply              5  .  .  .  .  .  .
  na.omit            46 10  5  4  1  3  2
  sapply              2  .  .  .  .  .  .
  simplify2array      3  .  .  .  .  .  .
  structure          11  .  .  1  .  .  .
  terms               2  .  .  .  .  .  .
  unlist              1  .  .  .  .  .  .
  vapply              7  .  .  .  1  .  .
                    run length
node                  1  2  3  4  5  6  7
  as.list             7  .  .  .  1  .  .
  eval                3  1  .  .  .  .  .
  FUN                 5  .  .  .  .  .  .
  lapply              3  .  .  .  .  .  .
  lazyLoadDBfetch     1  .  .  .  .  .  .
  na.omit.data.frame 43  7  4  5  .  3  1
```

```
  terms.formula       1  .  .  .  .  .  .
  unique              3  .  .  .  .  .  .
                    run length
node                  1  2  3  4  5  6
  .deparseOpts        3  .  .  .  .  .
  [                  46  4  3  3  .  1
  [[                  2  .  .  .  1  .
  %in%                1  .  .  .  .  .
  as.list.data.frame  7  .  .  .  1  .
  FUN                 1  .  .  .  .  .
  list                3  1  .  .  .  .
               run length
node              1  2  3  4  5  6
  [.data.frame   46  4  3  3  .  1
  [[.data.frame   2  .  .  .  1  .
  match           1  .  .  .  .  .
  paste           1  .  .  .  .  .
  pmatch          2  .  .  .  .  .
               run length
node             1 4 5
  !              2 . .
  %in%           1 . .
  <Anonymous>    . . 1
  anyDuplicated 9 2 1
  deparse        1 . .
  mode           1 . .
  names          2 . .
                       run length
node                    1 4 5
  %in%                  1 . .
  anyDuplicated.default 9 2 1
       run length
node    1
  match 1
      run length
node   1
  mode 1
```

Input

ToDo: add current level
ToDo: generate a coplot representation
ToDo: add time per call information: add marginals statistics run time by node

3. XXX – LOST & FOUND

Input

```
# xtable cannot handle posix
str(sprof01$info,
        caption="splot node info",
        label="tab:prSREinfo0")
```

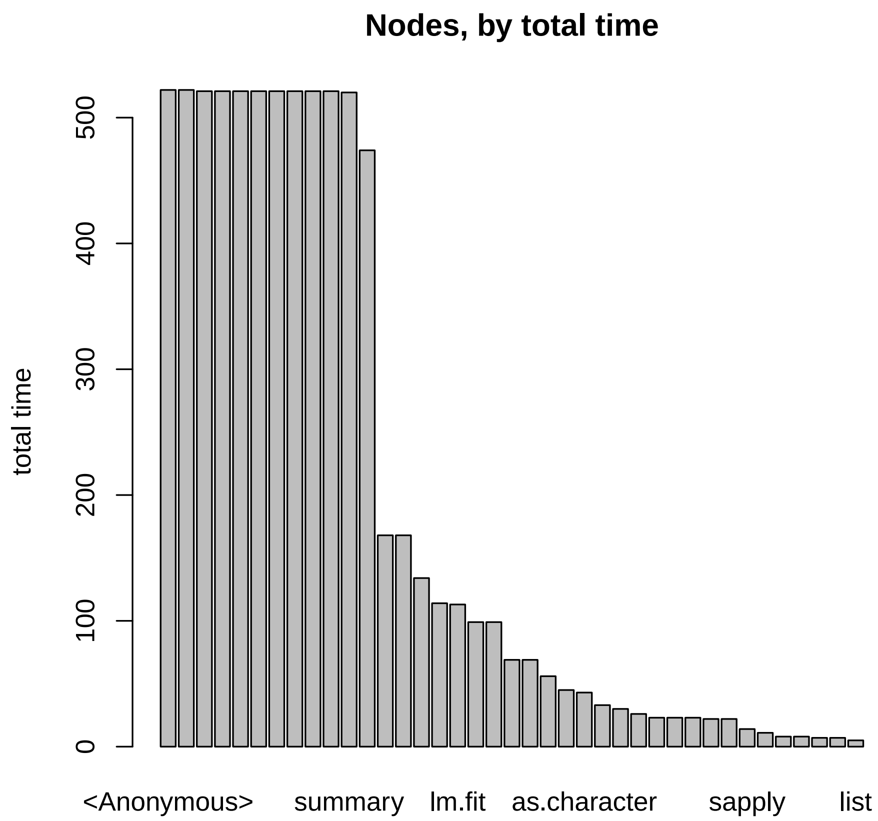
Output

```
'data.frame':        1 obs. of  8 variables:
 $ id       : Factor w/ 1 level "\"RprofsRegressionExpl01.out\" 2013-06-13 23:46:04": 1
```

```
$ date      : POSIXct, format: "2013-07-18 22:06:05"
$ nrnodes   : int 62
$ nrstacks  : int 50
$ nrrecords: int 522
$ firstline: Factor w/ 1 level "sample.interval=1000": 1
$ ctllines  : Factor w/ 1 level "sample.interval=1000": 1
$ ctllinenr: num 1
```

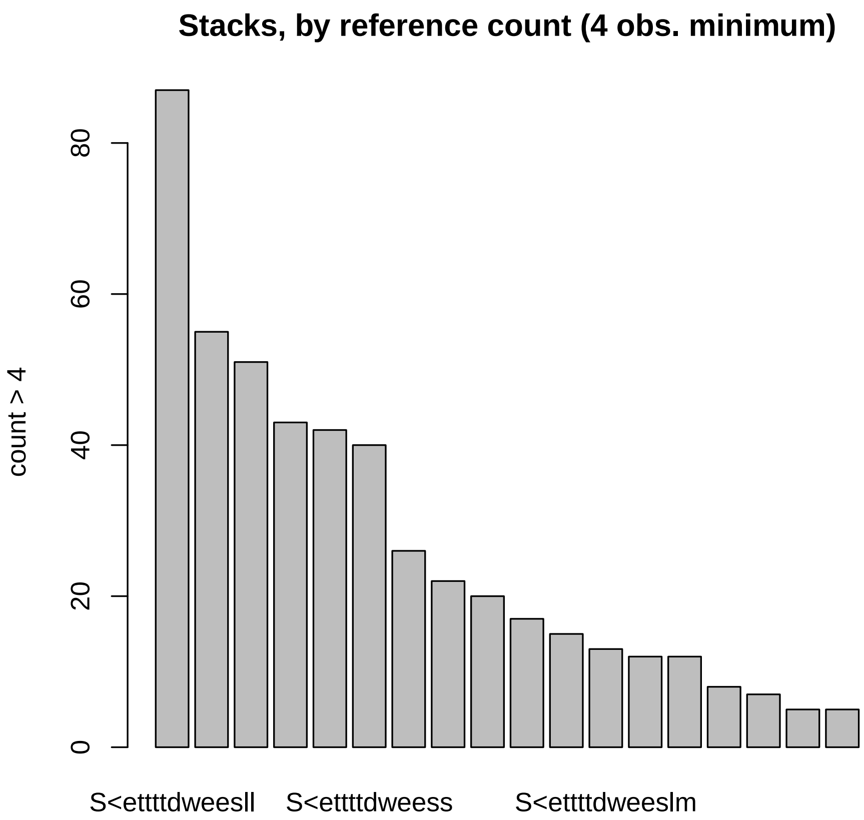
Selections are recorded as selection vectors, with reference to the original order. This needs some caution to align them with the order choices.

Input
```
rownames(sprof01$nodes) <- sprof01$nodes$names
nodesperm <- order(sprof01$nodes$total.time,decreasing=TRUE)
nodesnrobsok <- sprof01$nodes$total.time > 4
sp <- sprof01$nodes$total.time[nodesperm][nodesnrobsok[nodesperm]]
names(sp) <- sprof01$nodes$name[nodesperm][nodesnrobsok[nodesperm]]
barplot(sp,
 main="Nodes, by total time", ylab="total time")
```



ToDo: move to plot_stacks

Input
```
stacksperm <- order(sprof01$stacks$refcount,decreasing=TRUE)
stacksnrobsok <- sprof01$stacks$refcount > 4
sp4 <- sprof01$stacks$refcount[stacksperm][stacksnrobsok[stacksperm]]
names(sp4) <- sprof01$stacks$shortname[stacksperm][stacksnrobsok[stacksperm]]
barplot(sp4,
 main="Stacks, by reference count (4 obs. minimum)", ylab="count > 4")
```

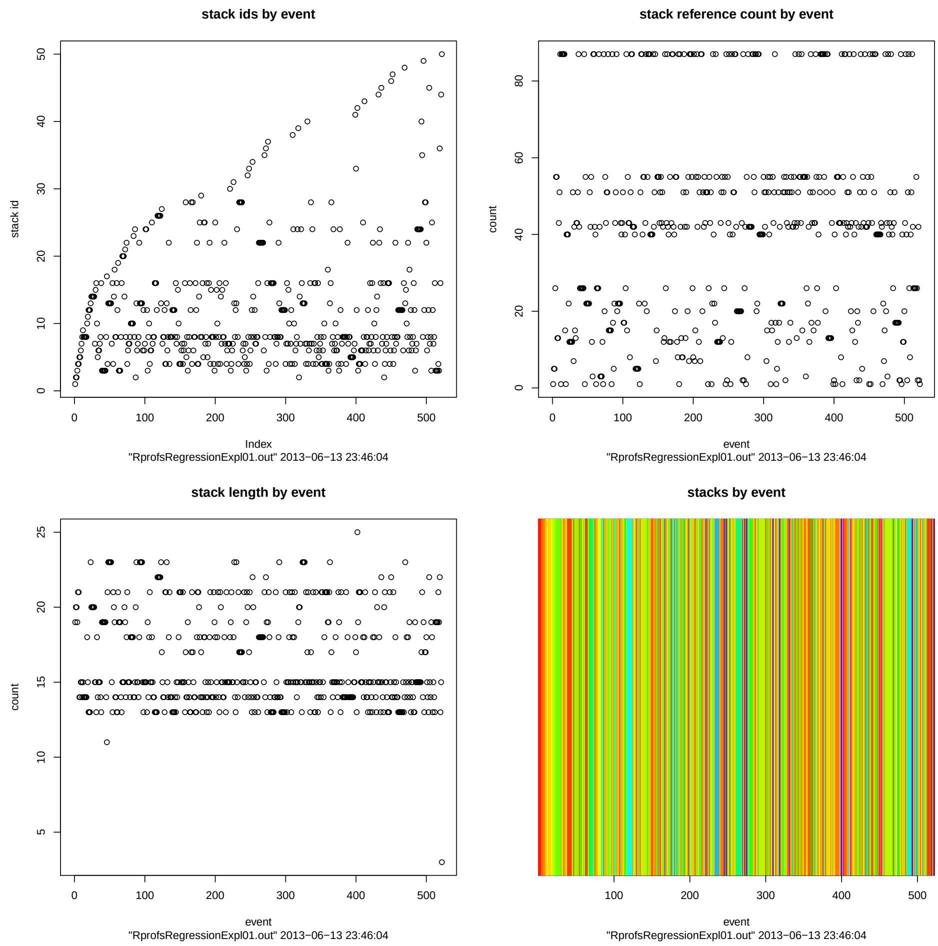


On the first look, information on the profile level is not informative. Profile records are just recordings of some step, taken at regular intervals. We get a minimal information, if we encode the stacks in colour.

ToDo: use stack colours

Input

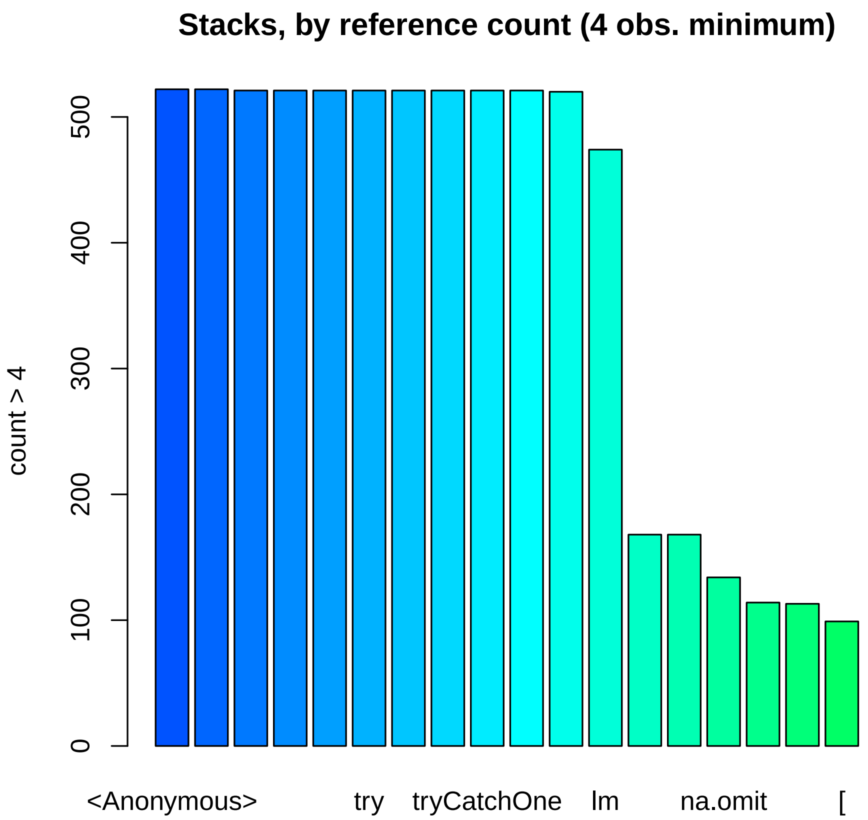
```
oldpar<-par(mfrow=c(2,2))
plot_profiles(sprof01)
par(oldpar)
```



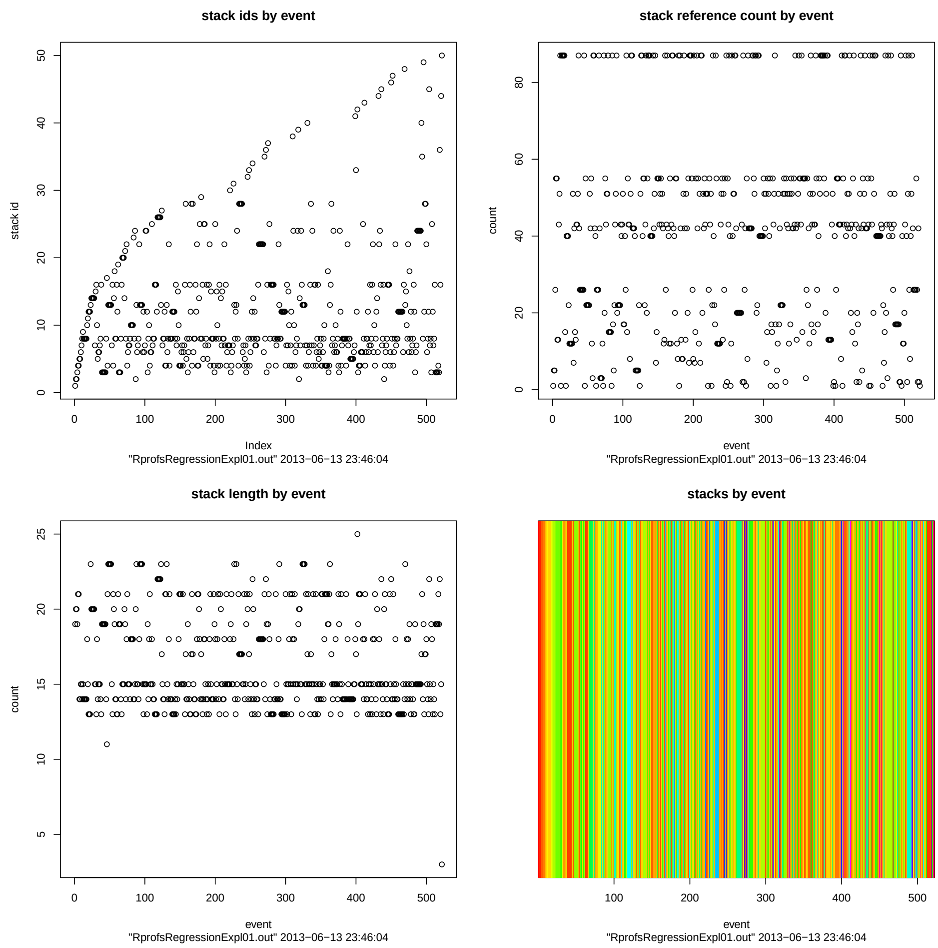
We now do a step down analysis. Aggregating the information from the profiling events, we have the frequency of stack references. On the stack level, we encode the frequency in color, and linking propagates this to the profile level.

Input

```
stackfreqscore <- rank(sprof01$stacks$refcount,ties.method="random")
stackfreqscore4<- stackfreqscore[stacksperm][stacksnrobsok[stacksperm]]
barplot(sp[stacksnrobsok[stacksperm]], main="Stacks, by reference count (4 obs. minimum)", ylab="co
col=rainbow(80)[stackfreqscore4])
```



Input
```
oldpar<- par(mfrow=c(2,2))
plot_profiles(sprof01)
par(oldpar)
```


stack ids by event
stack id
Index
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
stack reference count by event
count
event
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
stack length by event
count
event
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
stacks by event
event
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04

Input
```
prxt(sprof01$nodes,
        caption="nodes",
        label="tab:prSREnodes")
```

Table 6: nodes

	name	self.time	self.pct	total.time	total.pct	icol
1	!	2.00	0.38	2.00	0.03	50
2	..getNamespace	0.00	0.00	1.00	0.01	59
3	.deparseOpts	2.00	0.38	4.00	0.05	43
4	.getXlevels	0.00	0.00	26.00	0.34	27
5	[	0.00	0.00	99.00	1.29	18
6	[.data.frame	57.00	10.92	99.00	1.29	19
7	[[	0.00	0.00	8.00	0.10	35
8	[[.data.frame	1.00	0.19	8.00	0.10	36
9	%in%	1.00	0.19	4.00	0.05	41
10	<Anonymous>	6.00	1.15	522.00	6.79	2
< cut >	\vdots	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
53	terms	0.00	0.00	2.00	0.03	51
54	terms.formula	1.00	0.19	1.00	0.01	58
55	try	0.00	0.00	521.00	6.78	8
56	tryCatch	0.00	0.00	521.00	6.78	5
57	tryCatchList	0.00	0.00	521.00	6.78	10
58	tryCatchOne	0.00	0.00	521.00	6.78	7
59	unique	3.00	0.57	4.00	0.05	40
60	unlist	0.00	0.00	1.00	0.01	60
61	vapply	3.00	0.57	23.00	0.30	30
62	withVisible	0.00	0.00	521.00	6.78	4

Input
```
str(sprof01$stacks, max.level=1)
```

Output
```
'data.frame':       50 obs. of  7 variables:
 $ nodes          :List of 50
 $ shortname      : Factor w/ 50 levels "S<A>eFttCtCLtCOdTCwVeesleem.m..n.n...[[.",..: 27 17 19 1 35
 $ refcount       : num  1 5 26 55 13 43 51 87 1 15 ...
 $ stacklength    : int  19 20 19 21 14 15 15 14 15 18 ...
 $ stackheadnodes: int  52 52 52 52 52 52 52 52 52 52 ...
 $ stackleafnodes: int  27 28 41 6 39 14 38 30 27 49 ...
 $ stackssrc      : Factor w/ 50 levels "! [.data.frame [ na.omit.data.frame na.omit model.frame.defa
```

Input

Input
```
str(sprof01$profiles, max.level=1)
```

Output
```
List of 4
 $ data    : int [1:522] 1 2 2 3 4 4 5 5 6 7 ...
 $ mem     : NULL
 $ malloc  : NULL
 $ timesRLE:List of 2
  ..- attr(*, "class")= chr "rle"
```

Input

A summary is provided on request.

Input
```
sumsprof01 <- summary.sprof(sprof01)
```

Output
```
$id
[1] "Profile Summary Thu Jul 18 22:06:09 2013"

$len
[1] 522

$uniquestacks
[1] 50

$nr_runs
[1] 396

$nrstacks
[1] 50

$stacklength
[1]  3 25
```

```
$nrnodesperlevel
 [1]  1  1  2  1  1  1  1  1  1  1  1  1  3 10 11  9  9 15  8  7  5  7
[23]  2  1  1

                      shortname root leaf self.time  self.pct
!                             !    - LEAF         2  0.383142
..getNamespace              ..gN    -    -         0  0.000000
.deparseOpts                .dpO    - LEAF         2  0.383142
.getXlevels                 .gtX    -    -         0  0.000000
[                              [    -    -         0  0.000000
[.data.frame                [.d.    - LEAF        57 10.919540
[[                            [[    -    -         0  0.000000
[[.data.frame               [[..    - LEAF         1  0.191571
%in%                        %in%    - LEAF         1  0.191571
<Anonymous>                 <An>    - LEAF         6  1.149425
$                              $    - LEAF         1  0.191571
anyDuplicated               anyD    - LEAF         1  0.191571
anyDuplicated.default       anD.    - LEAF        22  4.214559
as.character                as.c    - LEAF        43  8.237548
as.list                     as.l    -    -         0  0.000000
as.list.data.frame          a...    - LEAF        22  4.214559
as.list.default             as..    - LEAF         1  0.191571
as.name                     as.n    - LEAF         1  0.191571
coef                        coef    - LEAF         1  0.191571
deparse                     dprs    - LEAF         1  0.191571
doTryCatch                  dTrC    -    -         0  0.000000
eval                        eval    - LEAF         1  0.191571
evalFunc                    evlF    -    -         0  0.000000
file                        file    - LEAF         1  0.191571
FUN                          FUN    - LEAF         1  0.191571
lapply                      lppl    - LEAF         2  0.383142
lazyLoadDBfetch             lLDB    - LEAF         2  0.383142
list                        list    - LEAF         5  0.957854
lm                            lm    - LEAF        42  8.045977
lm.fit                      lm.f    - LEAF        87 16.666667
match                       mtch    - LEAF         1  0.191571
mean                        mean    -    -         0  0.000000
mean.default                mn.d    - LEAF         2  0.383142
mode                        mode    - LEAF         2  0.383142
model.frame                mdl.f    -    -         0  0.000000
model.frame.default       mdl.f.    - LEAF        12  2.298851
model.matrix               mdl.m    -    -         0  0.000000
model.matrix.default      mdl.m.    - LEAF        51  9.770115
model.response             mdl.r    - LEAF        13  2.490421
na.omit                     n.mt    - LEAF        20  3.831418
na.omit.data.frame          n...    - LEAF        26  4.980843
names                       nams    - LEAF         2  0.383142
NCOL                        NCOL    - LEAF         1  0.191571
paste                       past    -    -         0  0.000000
pmatch                      pmtc    - LEAF         2  0.383142
rep.int                     rp.n    - LEAF         7  1.340996
sapply                      sppl    - LEAF         1  0.191571
simplify2array              smp2    -    -         0  0.000000
```

```
structure                  strc     - LEAF       32  6.130268
summary                    smmr     -    -        0  0.000000
summary.lm                 smm.     - LEAF       40  7.662835
Sweave                     Swev ROOT    -         0  0.000000
terms                      trms     -    -        0  0.000000
terms.formula              trm.     - LEAF        1  0.191571
try                         try     -    -        0  0.000000
tryCatch                   tryC     -    -        0  0.000000
tryCatchList               trCL     -    -        0  0.000000
tryCatchOne                trCO     -    -        0  0.000000
unique                     uniq     - LEAF        3  0.574713
unlist                     unls     -    -        0  0.000000
vapply                     vppl     - LEAF        3  0.574713
withVisible                wthV     -    -        0  0.000000
                      total.time  total.pct
!                              2   0.383142
..getNamespace                 1   0.191571
.deparseOpts                   4   0.766284
.getXlevels                   26   4.980843
[                             99  18.965517
[.data.frame                  99  18.965517
[[                             8   1.532567
[[.data.frame                  8   1.532567
%in%                           4   0.766284
<Anonymous>                  522 100.000000
$                              1   0.191571
anyDuplicated                 23   4.406130
anyDuplicated.default         22   4.214559
as.character                  43   8.237548
as.list                       23   4.406130
as.list.data.frame            22   4.214559
as.list.default                1   0.191571
as.name                        1   0.191571
coef                           1   0.191571
deparse                        2   0.383142
doTryCatch                   521  99.808429
eval                         521  99.808429
evalFunc                     521  99.808429
file                           1   0.191571
FUN                            7   1.340996
lapply                        30   5.747126
lazyLoadDBfetch                3   0.574713
list                           5   0.957854
lm                           474  90.804598
lm.fit                       113  21.647510
match                         11   2.107280
mean                           2   0.383142
mean.default                   2   0.383142
mode                           2   0.383142
model.frame                  168  32.183908
model.frame.default          168  32.183908
model.matrix                  69  13.218391
model.matrix.default          69  13.218391
```

```
model.response                56   10.727969
na.omit                      134   25.670498
na.omit.data.frame           114   21.839080
names                          2    0.383142
NCOL                           1    0.191571
paste                          1    0.191571
pmatch                         2    0.383142
rep.int                        7    1.340996
sapply                        14    2.681992
simplify2array                 4    0.766284
structure                     33    6.321839
summary                      520   99.616858
summary.lm                    45    8.620690
Sweave                       522  100.000000
terms                          2    0.383142
terms.formula                  1    0.191571
try                          521   99.808429
tryCatch                     521   99.808429
tryCatchList                 521   99.808429
tryCatchOne                  521   99.808429
unique                         4    0.766284
unlist                         1    0.191571
vapply                        23    4.406130
withVisible                  521   99.808429
```

Input

```
str(sumsprof01, max.level=2, width=70
)
```

Output

```
'data.frame':        62 obs. of  7 variables:
 $ shortname : Factor w/ 62 levels "!","..gN",".dp0",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ root      : Factor w/ 2 levels "-","ROOT": 1 1 1 1 1 1 1 1 1 1 ...
 $ leaf      : Factor w/ 2 levels "-","LEAF": 2 1 2 1 1 2 1 2 2 2 ...
 $ self.time : num  2 0 2 0 0 57 0 1 1 6 ...
 $ self.pct  : num  0.383 0 0.383 0 0 ...
 $ total.time: num  2 1 4 26 99 99 8 8 4 522 ...
 $ total.pct : num  0.383 0.192 0.766 4.981 18.966 ...
```

The classical approach hides the work that has been done. Actually it breaks down the data to record items. This figure is not reported anywhere. In our case, it can be reconstructed. The profile data have 8456 words in 524 lines.

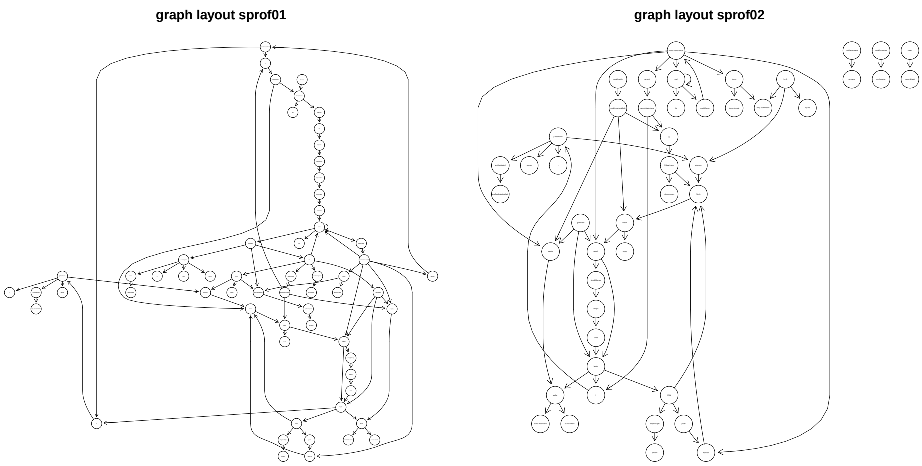
In our approach, we break down the information. Two lines of control information are split off. We have 522 lines of profile with 50 unique stacks, referencing 62 nodes. Instead of reducing it to a summary, we keep the full information. Information is always kept on its original level.

On the profiles level, we know the sample interval length, and the id of the stack recorded. On the stack level, for each stack we have a reference count, with the sample interval lengths used as weights. This reference count is added up for each node in the stack to give the node timings.

Cheap statistics are collected as the come by. For example, from the stacks table it is cheap to identify root and leaf nodes, and this mark is propagated to the nodes table.

3.1. graph Package.

Input
```
oldpar <- par(mfrow=c(1,2))
library(graph)
plot(as(adjacency(sprof01),"graphNEL"),  main="graph layout sprof01", cex.main=2)
plot(as(adjacency(sprof02),"graphNEL"),  main="graph layout sprof02", cex.main=2)
par(oldpar)
```



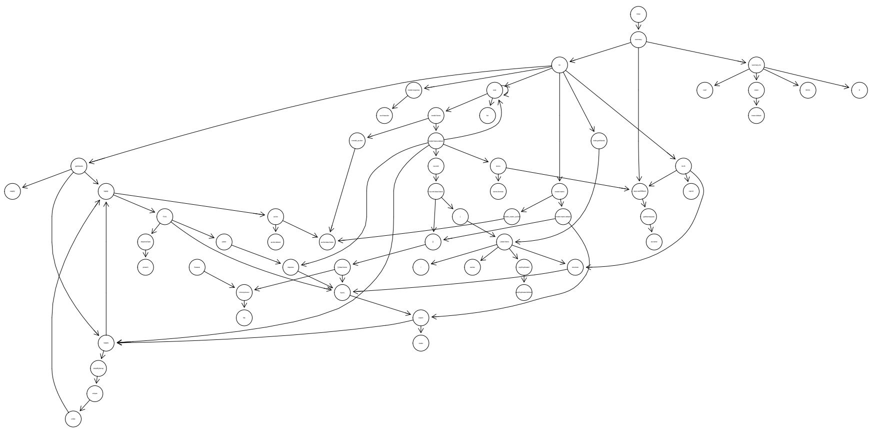
R is function based, and control structures in general are implemented as functions. In a graph representation, they appear as nodes, concentrating and seeding to unrelated paths. We can detect these on the stack level and replace them by surrogates, introducing new nodes. **ToDo:** fix null name

Input
```
sprof03 <- readRprof("RprofsRegressionExpl03.out")
#sprof03$nodes$name[1] <-  sprof03$nodes$name[2]
#sprof03$nodes$name[1]<-"<noop>"??
```

ToDo: cut top levels

Input
```
library(graph)
a03<-adjacency(sprof03)
rnames <- rownames(a03)
rnames[1]<-"noop";rownames(a03) <- rnames; colnames(a03) <- rnames;
plot(as(a03,"graphNEL"),  main="graph layout sprof01", cex.main=2)
```

graph layout sprof01



4. Standard output

For a reference, here are complete outputs of the standard function.

Input
```
sprof <- sprof01
```

4.1. **Print.** We omit the (lenghty) print output here and just give the commands as a reference.

ToDo: Clarify:"print prints its argument and returns it invisibly (via invisible(x))." Return the argument, or some print represntation?

ToDo: is there a print=FALSE variant to postpone printing to e.g. xtable?

Input
```
print_nodes(sprof)
```

Output
```
                     shortname root leaf self.time   self.pct
!                            !    - LEAF         2   0.383142
..getNamespace            ..gN    -    -         0   0.000000
.deparseOpts              .dpO    - LEAF         2   0.383142
.getXlevels               .gtX    -    -         0   0.000000
[                            [    -    -         0   0.000000
[.data.frame              [.d.    - LEAF        57  10.919540
[[                          [[    -    -         0   0.000000
[[.data.frame             [[..    - LEAF         1   0.191571
%in%                      %in%    - LEAF         1   0.191571
<Anonymous>               <An>    - LEAF         6   1.149425
$                            $    - LEAF         1   0.191571
anyDuplicated             anyD    - LEAF         1   0.191571
anyDuplicated.default     anD.    - LEAF        22   4.214559
as.character              as.c    - LEAF        43   8.237548
as.list                   as.l    -    -         0   0.000000
as.list.data.frame        a...    - LEAF        22   4.214559
as.list.default           as..    - LEAF         1   0.191571
as.name                   as.n    - LEAF         1   0.191571
coef                      coef    - LEAF         1   0.191571
deparse                   dprs    - LEAF         1   0.191571
```

```
doTryCatch                 dTrC    -    -      0  0.000000
eval                       eval    - LEAF      1  0.191571
evalFunc                   evlF    -    -      0  0.000000
file                       file    - LEAF      1  0.191571
FUN                         FUN    - LEAF      1  0.191571
lapply                     lppl    - LEAF      2  0.383142
lazyLoadDBfetch            lLDB    - LEAF      2  0.383142
list                       list    - LEAF      5  0.957854
lm                           lm    - LEAF     42  8.045977
lm.fit                     lm.f    - LEAF     87 16.666667
match                      mtch    - LEAF      1  0.191571
mean                       mean    -    -      0  0.000000
mean.default               mn.d    - LEAF      2  0.383142
mode                       mode    - LEAF      2  0.383142
model.frame               mdl.f    -    -      0  0.000000
model.frame.default      mdl.f.    - LEAF     12  2.298851
model.matrix              mdl.m    -    -      0  0.000000
model.matrix.default     mdl.m.    - LEAF     51  9.770115
model.response            mdl.r    - LEAF     13  2.490421
na.omit                    n.mt    - LEAF     20  3.831418
na.omit.data.frame         n...    - LEAF     26  4.980843
names                      nams    - LEAF      2  0.383142
NCOL                       NCOL    - LEAF      1  0.191571
paste                      past    -    -      0  0.000000
pmatch                     pmtc    - LEAF      2  0.383142
rep.int                    rp.n    - LEAF      7  1.340996
sapply                     sppl    - LEAF      1  0.191571
simplify2array             smp2    -    -      0  0.000000
structure                  strc    - LEAF     32  6.130268
summary                    smmr    -    -      0  0.000000
summary.lm                 smm.    - LEAF     40  7.662835
Sweave                     Swev ROOT    -      0  0.000000
terms                      trms    -    -      0  0.000000
terms.formula              trm.    - LEAF      1  0.191571
try                         try    -    -      0  0.000000
tryCatch                   tryC    -    -      0  0.000000
tryCatchList               trCL    -    -      0  0.000000
tryCatchOne                trCO    -    -      0  0.000000
unique                     uniq    - LEAF      3  0.574713
unlist                     unls    -    -      0  0.000000
vapply                     vppl    - LEAF      3  0.574713
withVisible                wthV    -    -      0  0.000000
                      total.time  total.pct
!                              2   0.383142
..getNamespace                 1   0.191571
.deparseOpts                   4   0.766284
.getXlevels                   26   4.980843
[                             99  18.965517
[.data.frame                  99  18.965517
[[                             8   1.532567
[[.data.frame                  8   1.532567
%in%                           4   0.766284
<Anonymous>                  522 100.000000
```

```
$                          1   0.191571
anyDuplicated             23   4.406130
anyDuplicated.default     22   4.214559
as.character              43   8.237548
as.list                   23   4.406130
as.list.data.frame        22   4.214559
as.list.default            1   0.191571
as.name                    1   0.191571
coef                       1   0.191571
deparse                    2   0.383142
doTryCatch               521  99.808429
eval                     521  99.808429
evalFunc                 521  99.808429
file                       1   0.191571
FUN                        7   1.340996
lapply                    30   5.747126
lazyLoadDBfetch            3   0.574713
list                       5   0.957854
lm                       474  90.804598
lm.fit                   113  21.647510
match                     11   2.107280
mean                       2   0.383142
mean.default               2   0.383142
mode                       2   0.383142
model.frame              168  32.183908
model.frame.default      168  32.183908
model.matrix              69  13.218391
model.matrix.default      69  13.218391
model.response            56  10.727969
na.omit                  134  25.670498
na.omit.data.frame       114  21.839080
names                      2   0.383142
NCOL                       1   0.191571
paste                      1   0.191571
pmatch                     2   0.383142
rep.int                    7   1.340996
sapply                    14   2.681992
simplify2array             4   0.766284
structure                 33   6.321839
summary                  520  99.616858
summary.lm                45   8.620690
Sweave                   522 100.000000
terms                      2   0.383142
terms.formula              1   0.191571
try                      521  99.808429
tryCatch                 521  99.808429
tryCatchList             521  99.808429
tryCatchOne              521  99.808429
unique                     4   0.766284
unlist                     1   0.191571
vapply                    23   4.406130
withVisible              521  99.808429
```

Input
```
print_stacks(sprof)
```

Output
```
$nrstacks
[1] 50

$stacklength
[1]  3 25

$nrnodesperlevel
 [1]  1  1  2  1  1  1  1  1  1  1  1  1  3 10 11  9  9 15  8  7  5  7
[23]  2  1  1
```

Input
```
print_profiles(sprof)
```

Output
```
$id
[1] "Profile Summary Thu Jul 18 22:06:10 2013"

$len
[1] 522

$uniquestacks
[1] 50

$nr_runs
[1] 396
```

The *print()* method for *sprof* objects concatenates these three functions.

4.2. Summary.

Input
```
summary_nodes(sprof)
```

Output
```
                      shortname root leaf self.time  self.pct
!                             !    - LEAF         2  0.383142
..getNamespace             ..gN    -    -         0  0.000000
.deparseOpts               .dpO    - LEAF         2  0.383142
.getXlevels                .gtX    -    -         0  0.000000
[                             [    -    -         0  0.000000
[.data.frame               [.d.    - LEAF        57 10.919540
[[                           [[    -    -         0  0.000000
[[.data.frame              [[..    - LEAF         1  0.191571
%in%                       %in%    - LEAF         1  0.191571
<Anonymous>                <An>    - LEAF         6  1.149425
$                             $    - LEAF         1  0.191571
anyDuplicated              anyD    - LEAF         1  0.191571
anyDuplicated.default      anD.    - LEAF        22  4.214559
as.character               as.c    - LEAF        43  8.237548
as.list                    as.l    -    -         0  0.000000
```

```
as.list.data.frame          a...    - LEAF        22  4.214559
as.list.default             as..    - LEAF         1  0.191571
as.name                     as.n    - LEAF         1  0.191571
coef                        coef    - LEAF         1  0.191571
deparse                     dprs    - LEAF         1  0.191571
doTryCatch                  dTrC    -    -         0  0.000000
eval                        eval    - LEAF         1  0.191571
evalFunc                    evlF    -    -         0  0.000000
file                        file    - LEAF         1  0.191571
FUN                          FUN    - LEAF         1  0.191571
lapply                      lppl    - LEAF         2  0.383142
lazyLoadDBfetch             lLDB    - LEAF         2  0.383142
list                        list    - LEAF         5  0.957854
lm                            lm    - LEAF        42  8.045977
lm.fit                      lm.f    - LEAF        87 16.666667
match                       mtch    - LEAF         1  0.191571
mean                        mean    -    -         0  0.000000
mean.default                mn.d    - LEAF         2  0.383142
mode                        mode    - LEAF         2  0.383142
model.frame                mdl.f    -    -         0  0.000000
model.frame.default       mdl.f.    - LEAF        12  2.298851
model.matrix               mdl.m    -    -         0  0.000000
model.matrix.default      mdl.m.    - LEAF        51  9.770115
model.response             mdl.r    - LEAF        13  2.490421
na.omit                     n.mt    - LEAF        20  3.831418
na.omit.data.frame          n...    - LEAF        26  4.980843
names                       nams    - LEAF         2  0.383142
NCOL                        NCOL    - LEAF         1  0.191571
paste                       past    -    -         0  0.000000
pmatch                      pmtc    - LEAF         2  0.383142
rep.int                     rp.n    - LEAF         7  1.340996
sapply                      sppl    - LEAF         1  0.191571
simplify2array              smp2    -    -         0  0.000000
structure                   strc    - LEAF        32  6.130268
summary                     smmr    -    -         0  0.000000
summary.lm                  smm.    - LEAF        40  7.662835
Sweave                      Swev ROOT    -         0  0.000000
terms                       trms    -    -         0  0.000000
terms.formula               trm.    - LEAF         1  0.191571
try                          try    -    -         0  0.000000
tryCatch                    tryC    -    -         0  0.000000
tryCatchList                trCL    -    -         0  0.000000
tryCatchOne                 trCO    -    -         0  0.000000
unique                      uniq    - LEAF         3  0.574713
unlist                      unls    -    -         0  0.000000
vapply                      vppl    - LEAF         3  0.574713
withVisible                 wthV    -    -         0  0.000000
                      total.time  total.pct
!                              2   0.383142
..getNamespace                 1   0.191571
.deparseOpts                   4   0.766284
.getXlevels                   26   4.980843
[                             99  18.965517
```

```
[.data.frame                99  18.965517
[[                           8   1.532567
[[.data.frame                8   1.532567
%in%                         4   0.766284
<Anonymous>                522 100.000000
$                            1   0.191571
anyDuplicated               23   4.406130
anyDuplicated.default       22   4.214559
as.character                43   8.237548
as.list                     23   4.406130
as.list.data.frame          22   4.214559
as.list.default              1   0.191571
as.name                      1   0.191571
coef                         1   0.191571
deparse                      2   0.383142
doTryCatch                 521  99.808429
eval                       521  99.808429
evalFunc                   521  99.808429
file                         1   0.191571
FUN                          7   1.340996
lapply                      30   5.747126
lazyLoadDBfetch              3   0.574713
list                         5   0.957854
lm                         474  90.804598
lm.fit                     113  21.647510
match                       11   2.107280
mean                         2   0.383142
mean.default                 2   0.383142
mode                         2   0.383142
model.frame                168  32.183908
model.frame.default        168  32.183908
model.matrix                69  13.218391
model.matrix.default        69  13.218391
model.response              56  10.727969
na.omit                    134  25.670498
na.omit.data.frame         114  21.839080
names                        2   0.383142
NCOL                         1   0.191571
paste                        1   0.191571
pmatch                       2   0.383142
rep.int                      7   1.340996
sapply                      14   2.681992
simplify2array               4   0.766284
structure                   33   6.321839
summary                    520  99.616858
summary.lm                  45   8.620690
Sweave                     522 100.000000
terms                        2   0.383142
terms.formula                1   0.191571
try                        521  99.808429
tryCatch                   521  99.808429
tryCatchList               521  99.808429
tryCatchOne                521  99.808429
```

```
unique                        4   0.766284
unlist                        1   0.191571
vapply                       23   4.406130
withVisible                 521  99.808429
```

Input

```
summary_stacks(sprof)
```

Output

```
$nrstacks
[1] 50

$stacklength
[1]  3 25

$nrnodesperlevel
 [1]  1  1  2  1  1  1  1  1  1  1  1  1  3 10 11  9  9 15  8  7  5  7
[23]  2  1  1
```

Input

```
summary_profiles(sprof)
```

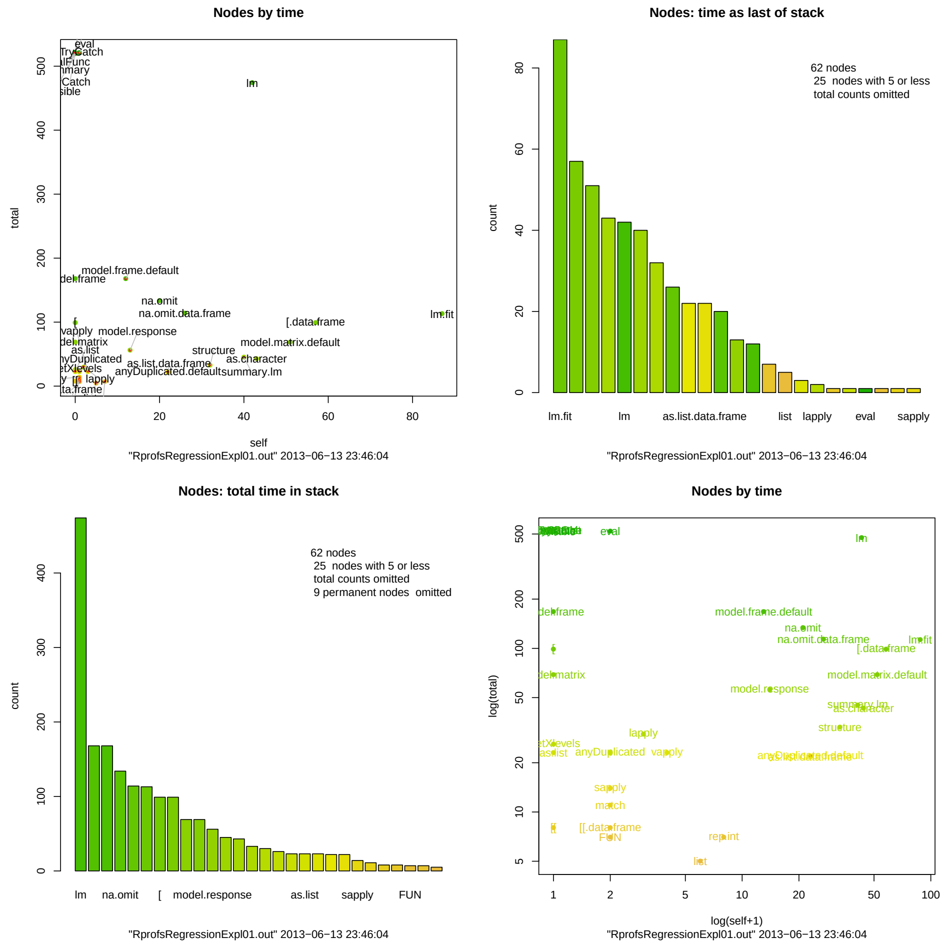
Output

```
$id
[1] "Profile Summary Thu Jul 18 22:06:10 2013"

$len
[1] 522

$uniquestacks
[1] 50

$nr_runs
[1] 396
```

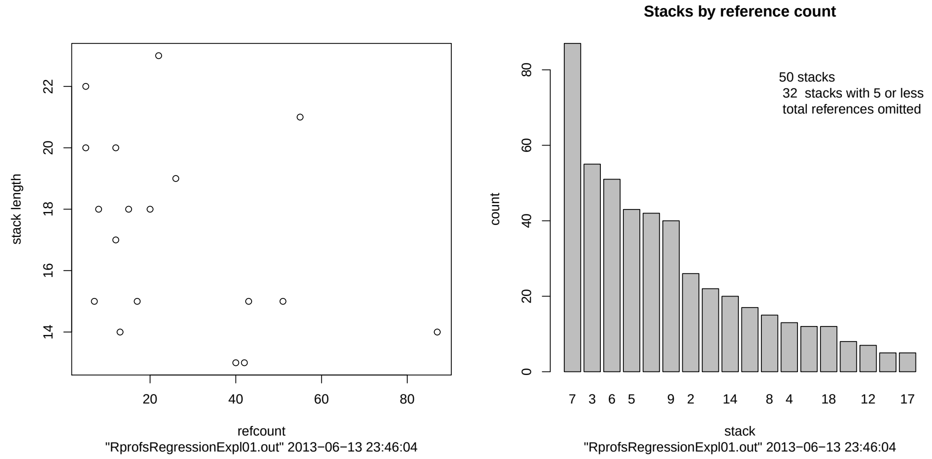
The *summary()* method for *sprof* objects concatenates these three functions.

4.3. **Plot.** Looking at lists of numbers is not too informative. We get a first impression by plotting the data.

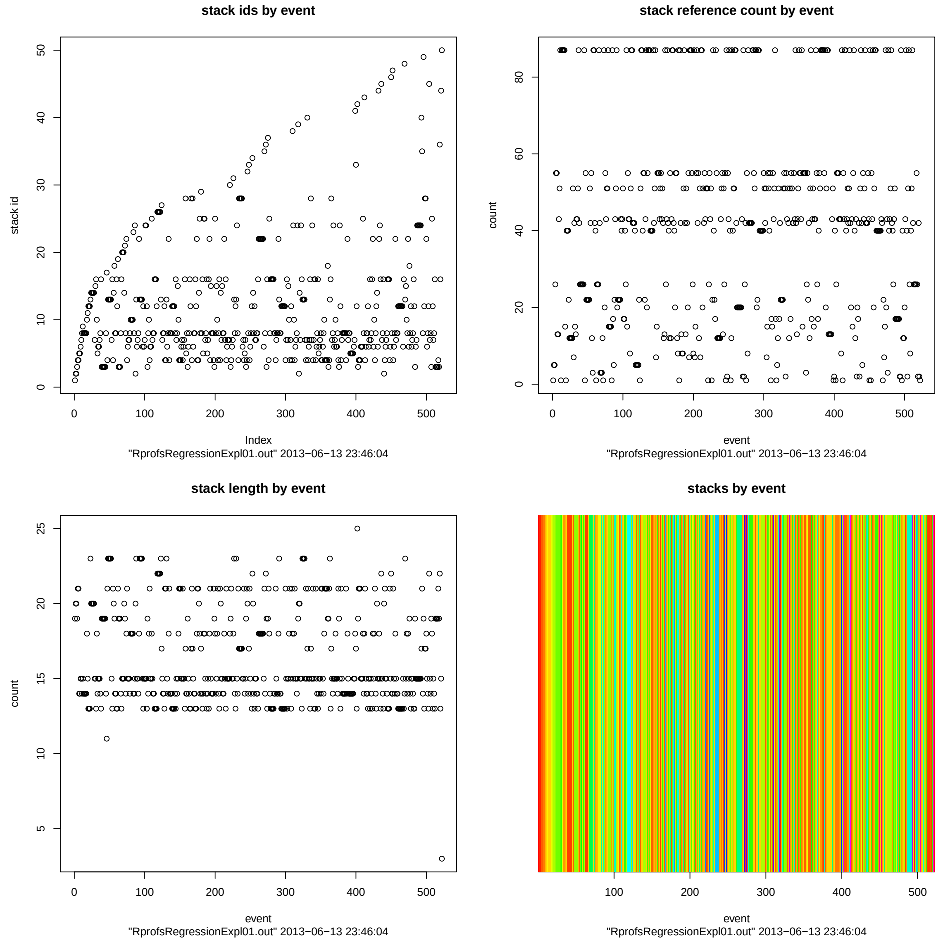
Input
```
#plot_nodes(sprof01, col=nodescol[nodescore])
oldpar <- par(mfrow=c(2,2))
plot_nodes(sprof01)
par(oldpar)
```



Input
```
oldpar <- par(mfrow=c(1,2))
plot_stacks(sprof01)
par(oldpar)
```



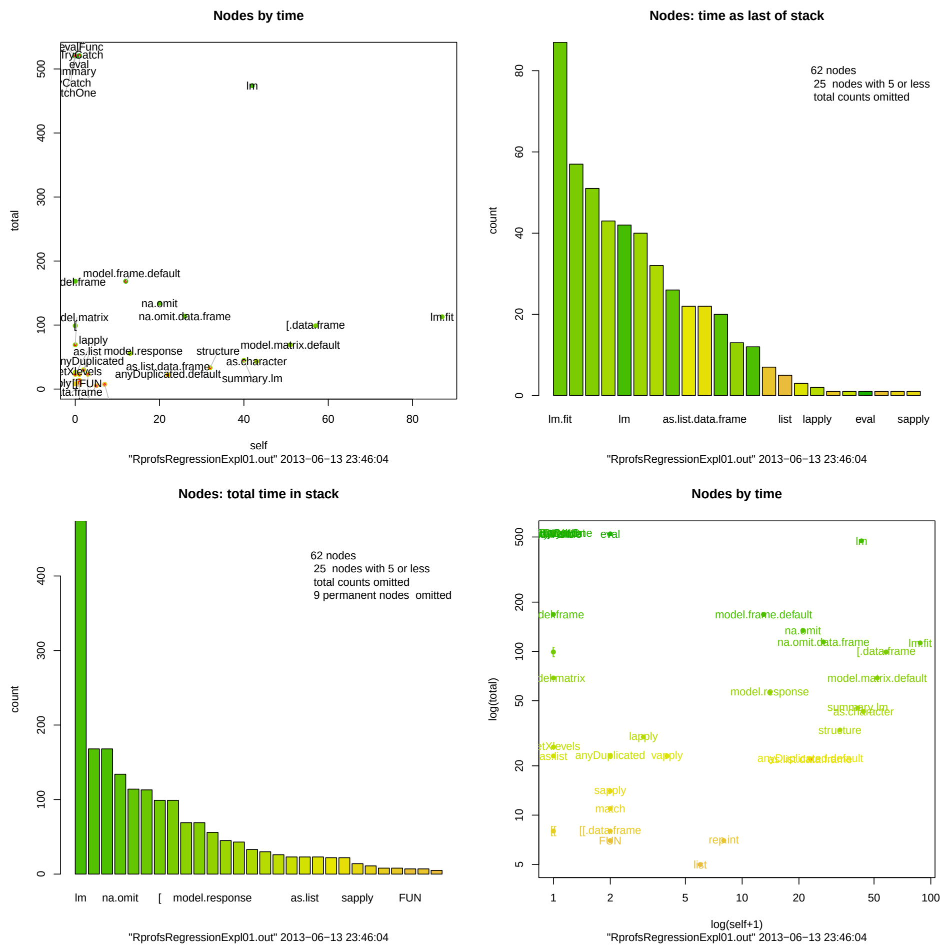
Input
```
oldpar <-par(mfrow=c(2,2))
plot_profiles(sprof01)
par(oldpar)
```



The *plot()* method for *sprof* objects concatenates these three functions.

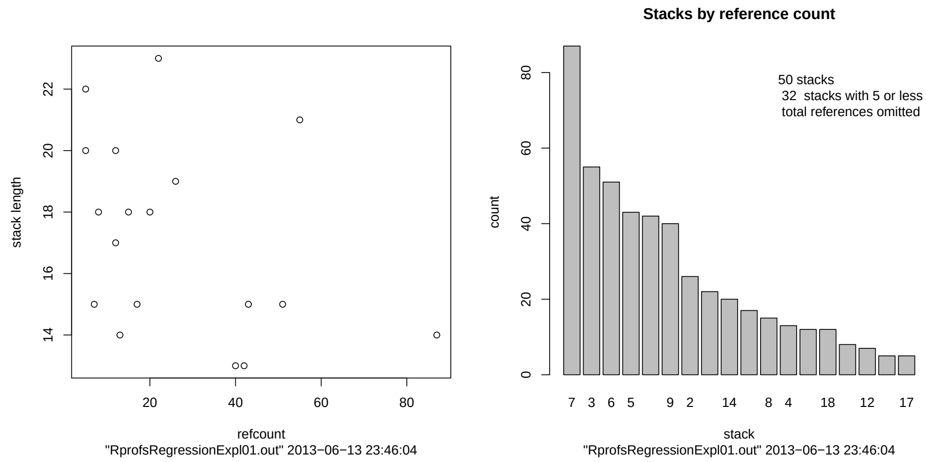
Input

```
oldpar<- par(mfrow=c(2,2))
plot_nodes(sprof)
par(oldpar)
```


Nodes by time
total
self
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
evalFunc
tryCatch
eval
lm
model.frame.default
na.omit
na.omit.data.frame
[.data.frame
lm.fit
lapply
as.list
model.response
structure
model.matrix.default
as.character
as.list.data.frame
anyDuplicated.default
summary.lm
Nodes: time as last of stack
count
62 nodes
25 nodes with 5 or less
total counts omitted
lm.fit
lm
as.list.data.frame
list
lapply
eval
sapply
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
Nodes: total time in stack
count
62 nodes
25 nodes with 5 or less
total counts omitted
9 permanent nodes omitted
lm
na.omit
[
model.response
as.list
sapply
FUN
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04
Nodes by time
log(total)
log(self+1)
eval
lm
model.frame.default
na.omit
na.omit.data.frame
[.data.frame
lm.fit
model.matrix.default
model.response
structure
lapply
anyDuplicated
vapply
sapply
match
[[.data.frame
FUN
reprint
list
"RprofsRegressionExpl01.out" 2013−06−13 23:46:04

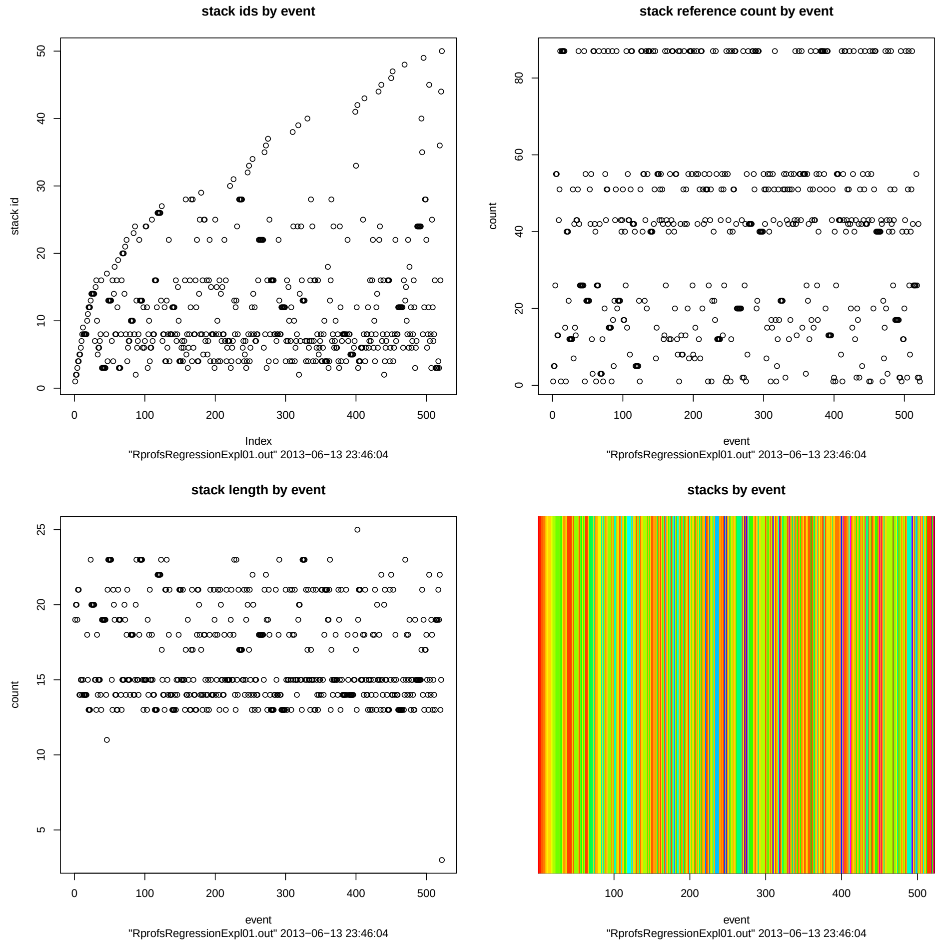
Input

```
oldpar<- par(mfrow=c(1,2))
plot_stacks(sprof)
par(oldpar)
```



Input

```
oldpar<- par(mfrow=c(2,2))
plot_profiles(sprof)
par(oldpar)
```



The *plot()* method for *sprof* objects concatenates these three functions.

5. Graph

Graph layout is a theme of its own. Proposals are readily available, as are their implementation. For some of them, there are R interfaces or re-implementations in R. Their usefulness in out context has to be explored, and the answers will vary with personal preferences.

For some graph layout packages we illustrate an interface here and show a sample result. We use the original profile data here. This is a nasty graph with some R stack peculiarites. The corresponding results for the trimmed profile data are shown in the next section section 5.2 on page 75. This is a more realistic example of the kind of graphs you will have to work with.

ToDo: by graph package: preferred input format?

ToDo: use attributes. Edge with should be easy.

ToDo: include information from stack connectivity.

5.1. **regression example.** In this section, we use the recent version of our example, *sprof02* for demonstration. You can re-run it, using your *sprof* data by modifying this instruction:

Input
```
sprof <- sprof02
```

To interface *sprof* to a graph handling package, `adjacency()` can extract the adjacency matrix from the profile.

There are various packages for finding a graph layout, and the choice is open to your preferences. The R packages for most of these are just wrapper

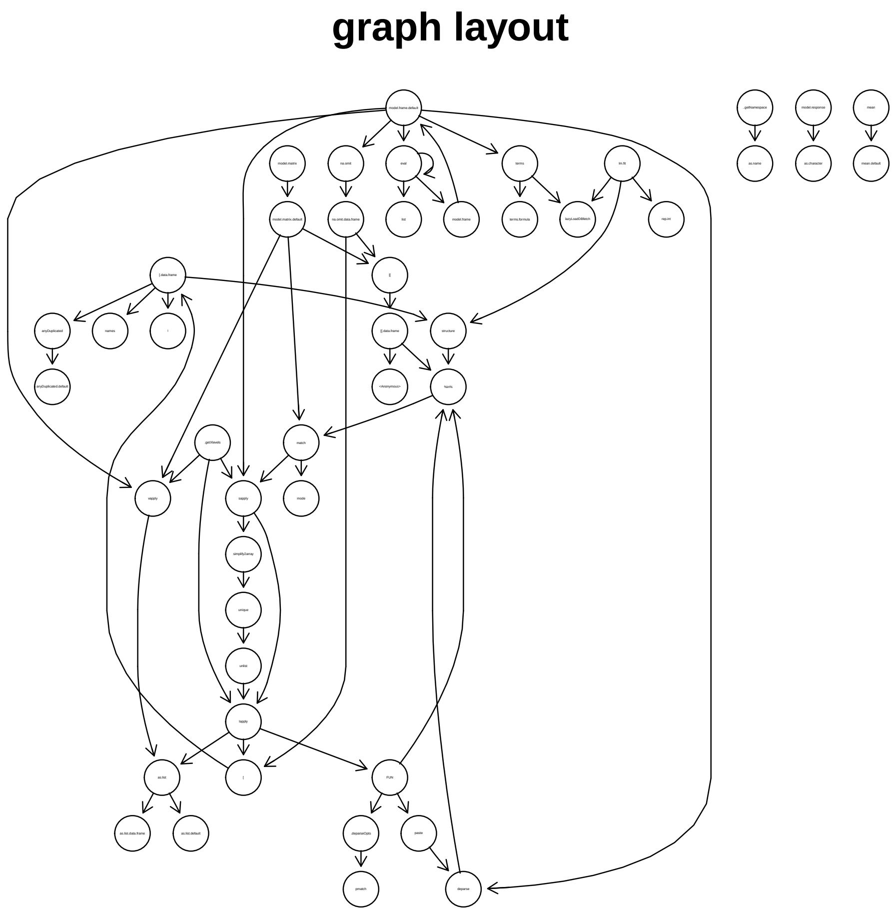
Input
```
sprofadj <- adjacency(sprof)
```

This is a format any graph package can handle (maybe).

5.1.1. *graph Package.*

Input
```
library(graph)
sprofadjNEL <- as(sprofadj,"graphNEL")
```

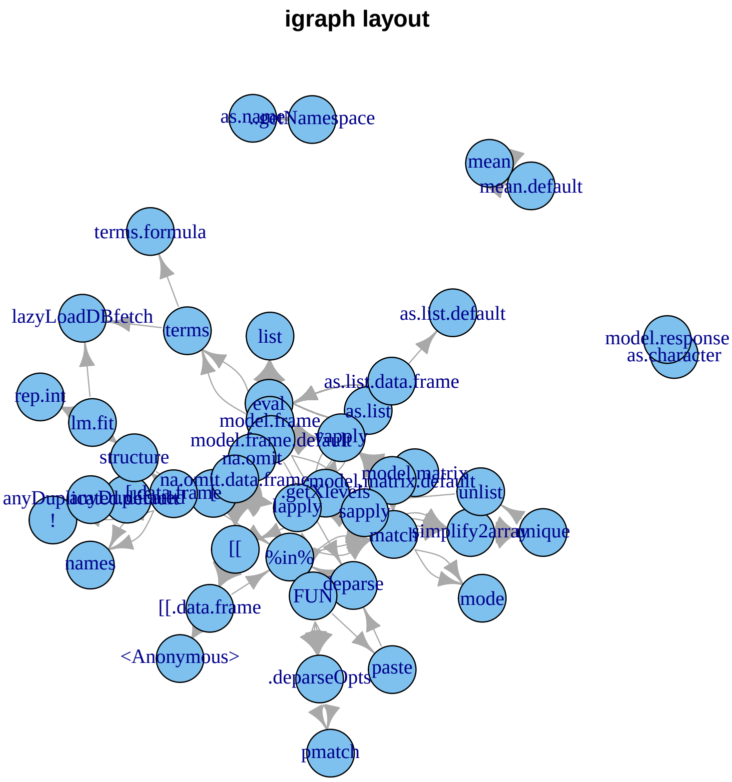
Input
```
plot(sprofadjNEL,  main="graph layout", cex.main=2)
#detach("package:graph")
```



5.1.2. *igraph Package.*

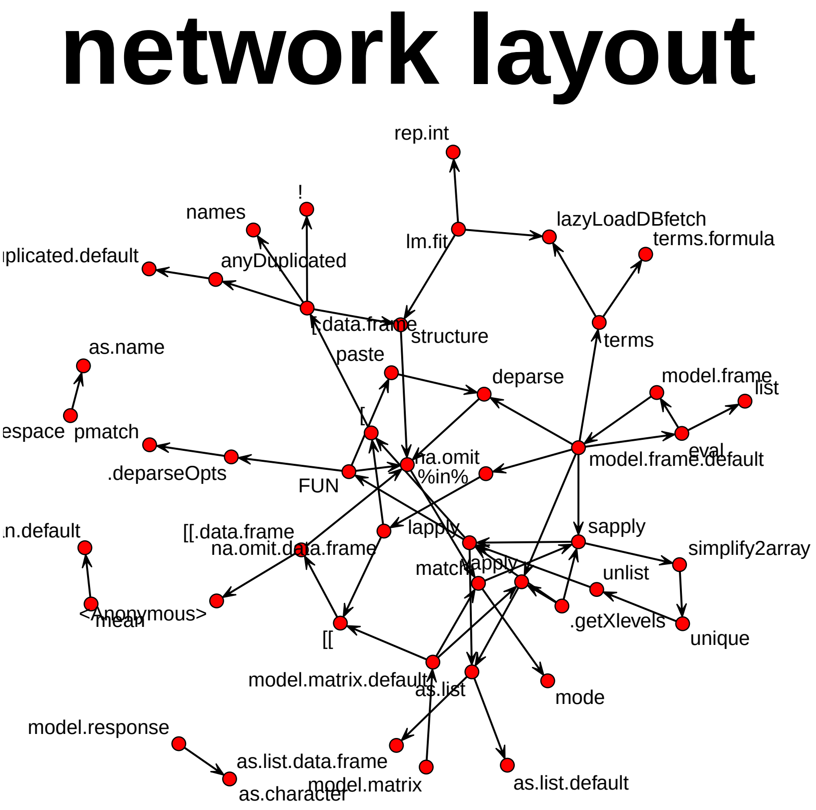
Input
```
library(igraph)
sprofig <- graph.adjacency(sprofadj)
```

Input
```
#plot(sprofig, main="igraph layout", cex.main=5)
plot(sprofig, main="igraph layout")
detach("package:igraph")
```



5.1.3. *network Package.*

Input
```
library(network)
nwsprofadj <- as.network(sprofadj) # names is not imported
network.vertex.names(nwsprofadj) <- rownames(sprofadj) # not honoured by plot
plot(nwsprofadj, label=rownames(sprofadj), main="network layout", cex.main=5)
```

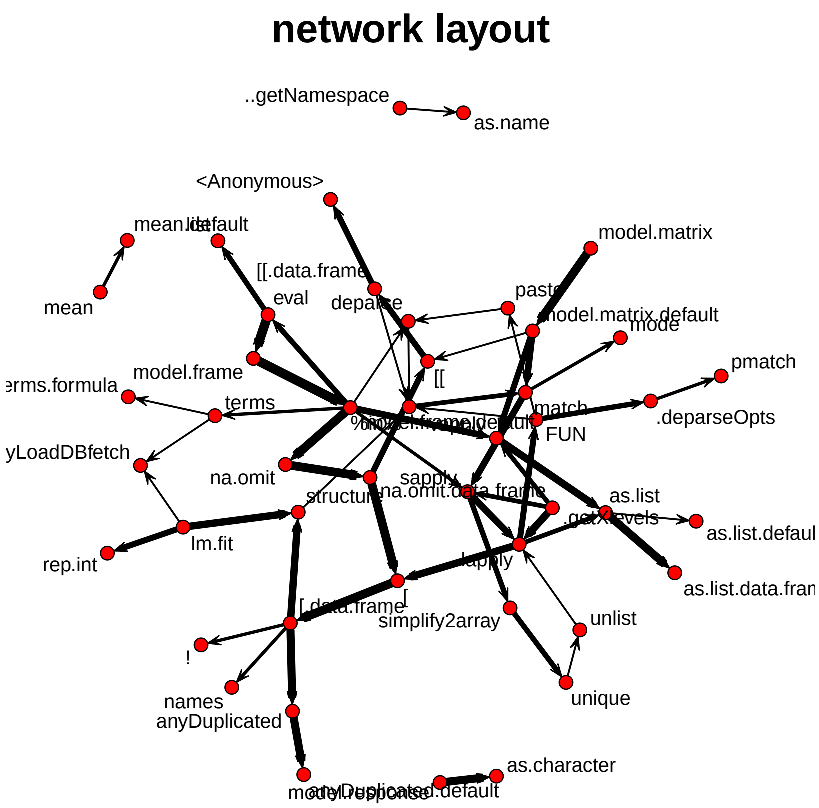


Experiments to include weight.

ToDo: maximum edge.lwd?

Input

```
edge.lwd<-sprofadj
edge.lwd[edge.lwd>0]<- rank(edge.lwd[edge.lwd>0], ties.method="min")
#edge.lwd <- trunc(sprofadj/max(sprofadj)*10)+1
edge.lwd <- round(edge.lwd/max(edge.lwd)*12)
plot(nwsprofadj, label=rownames(sprofadj), main="network layout", cex.main=2, edge.lwd=edge.lwd)
detach("package:network")
```



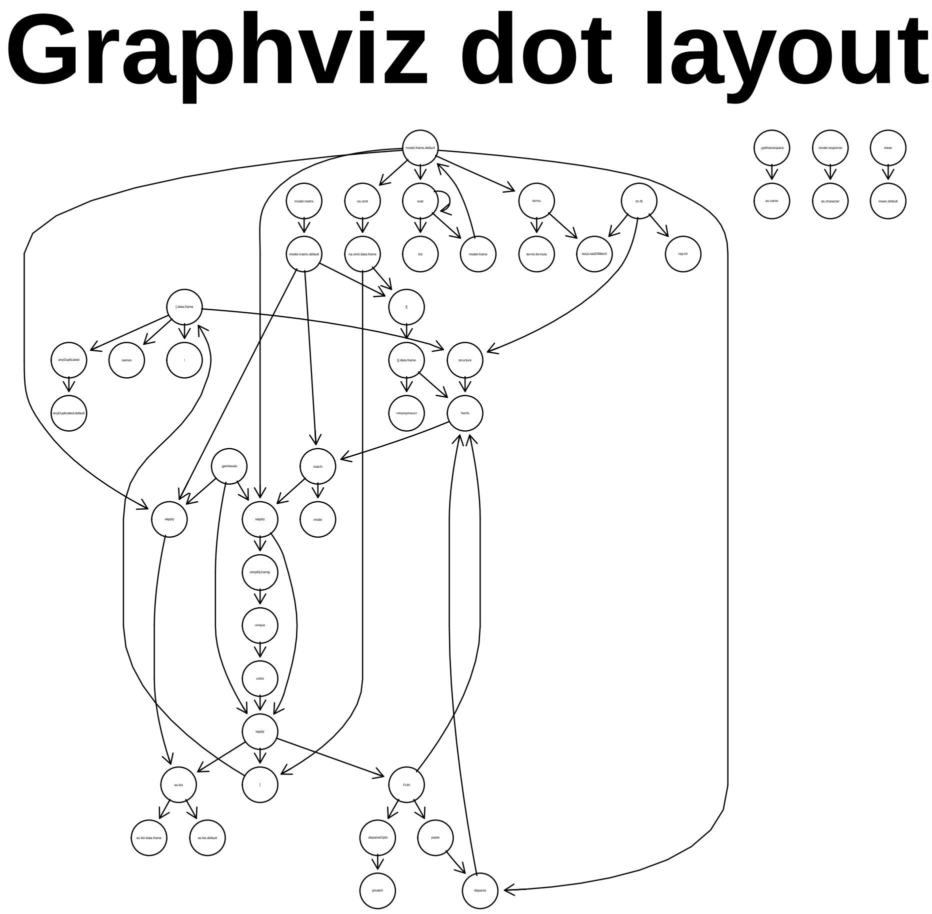
5.1.4. *Rgraphviz Package.*

Input

```
library(Rgraphviz)
sprofadjRag <- agopen(sprofadjNEL, name="Rprof Example")
```

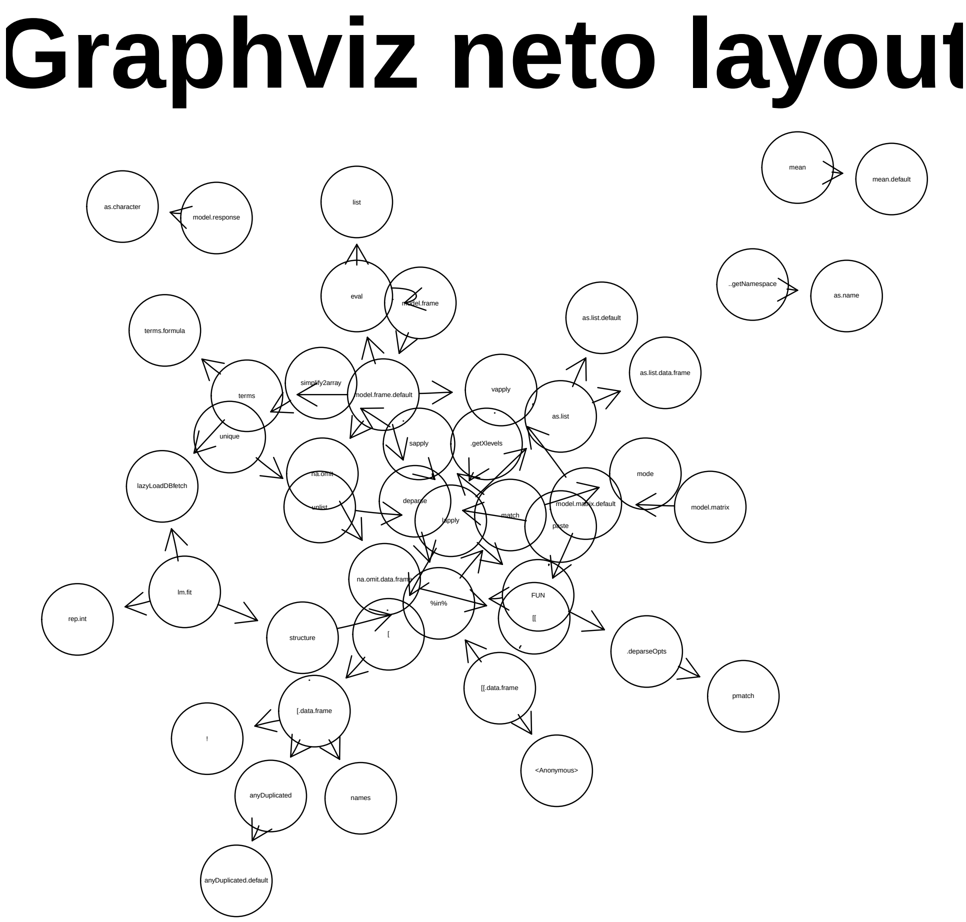
Input

```
plot(sprofadjRag, main="Graphviz dot layout", cex.main=5)
```



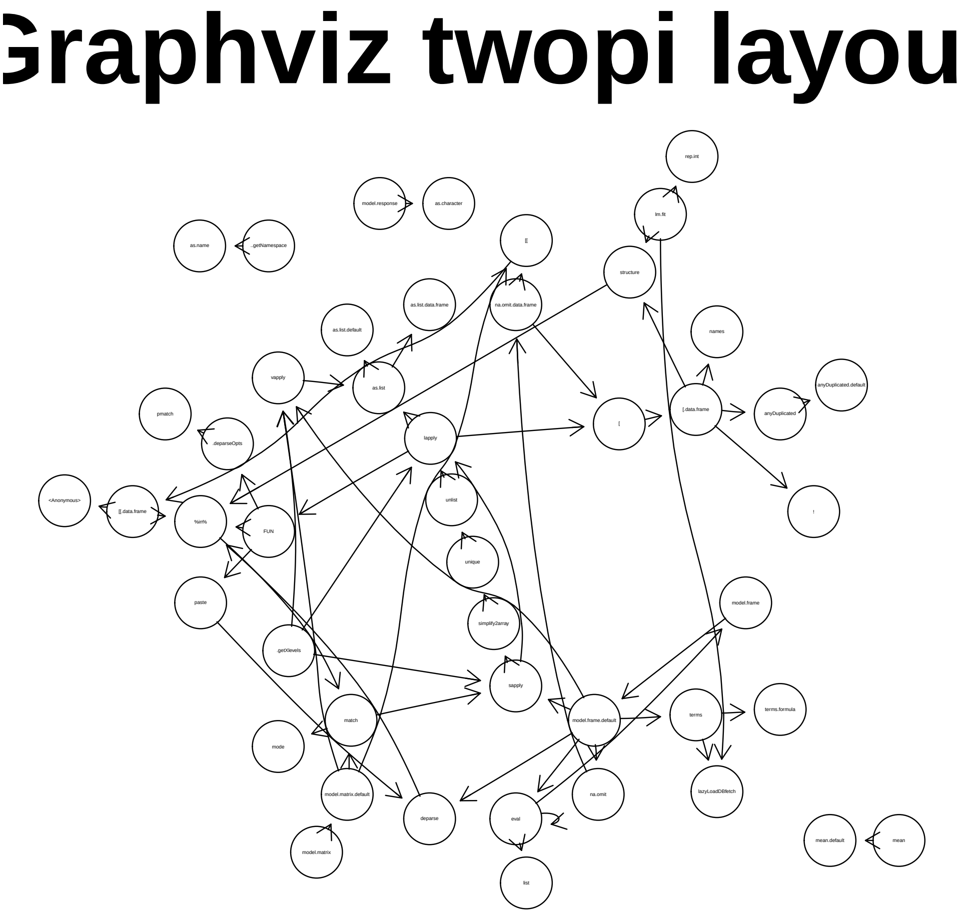
Input

```
plot(sprofadjRag,"neato", main="Graphviz neto layout", cex.main=5)
```

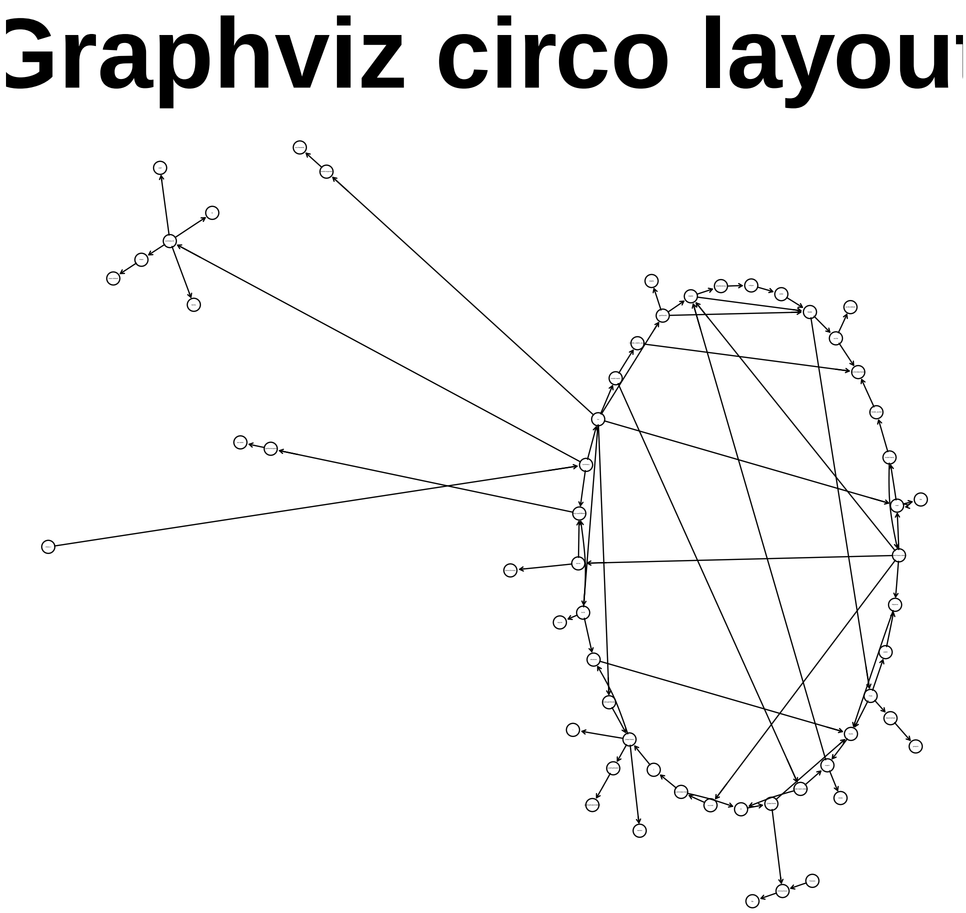


Input

```
plot(sprofadjRag,"twopi", main="Graphviz twopi layout", cex.main=5)
```

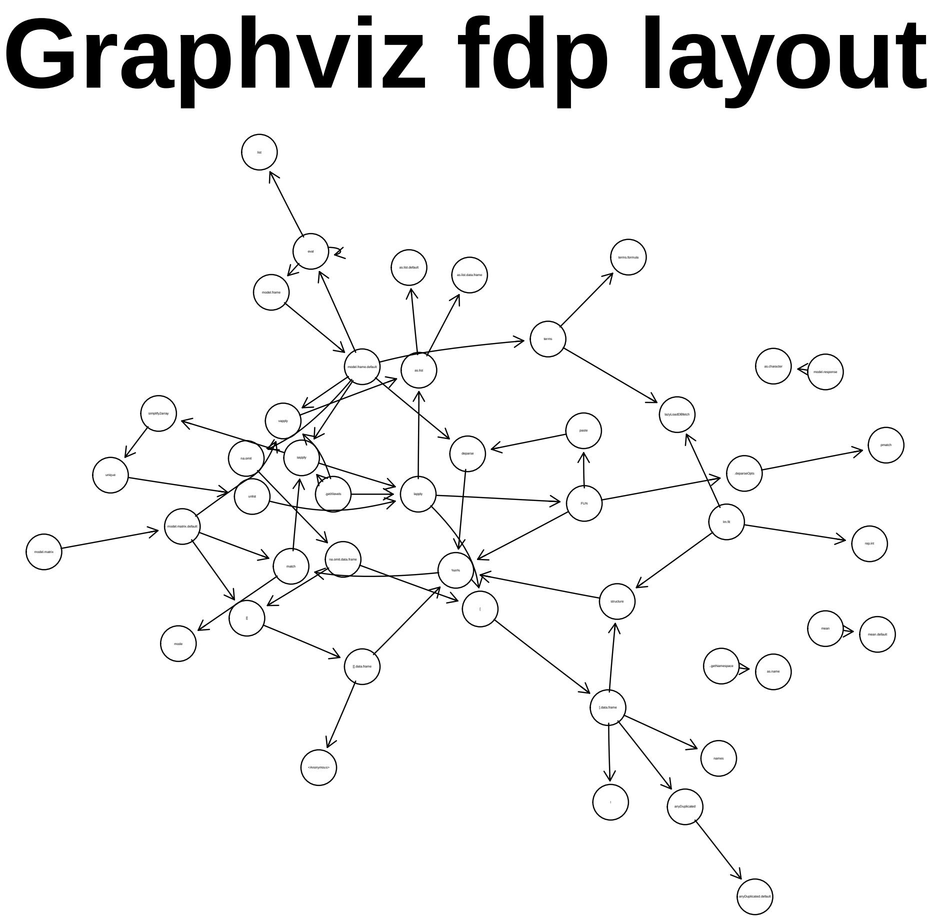


Input
```
plot(sprofadjRag,"circo", main="Graphviz circo layout", cex.main=5)
```



Input

```
plot(sprofadjRag,"fdp", main="Graphviz fdp layout", cex.main=5)
```



5.2. **trimmed regression example.** In this section, we use the reduced version of our example, *sprof03* for demonstration. Except for the change of the data set, this is just a copy of the previous chapter, collecting the various layouts for easy reference.

Some experiments may have found their way to this chapter. They will be expelled.

You can re-run it, using your *sprof* data by modifying this instruction:

Input
```
sprof <- sprof03
```

To interface *sprof* to a graph handling package, *until()* can extract the adjacency matrix from the profile.

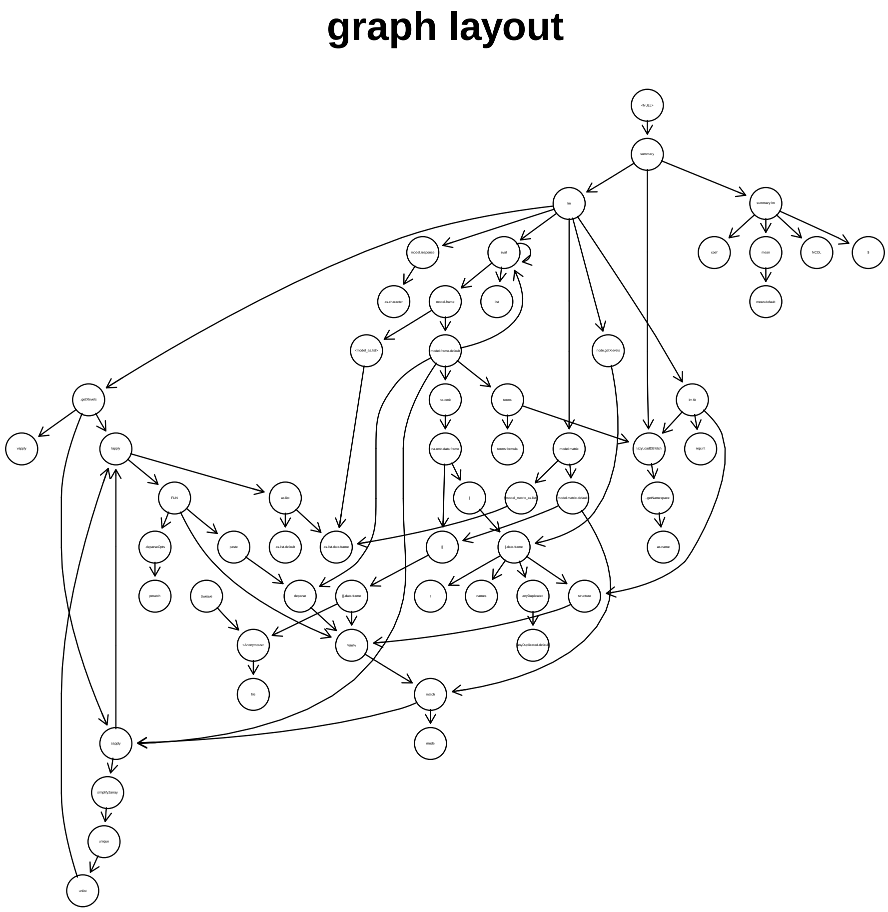
Input
```
sprofadj <- adjacency(sprof)
adjname <- colnames(sprofadj)
adjname[adjname==""] <- "<NULL>"
 colnames(sprofadj) <- adjname
 rownames(sprofadj) <- adjname
```

This is a format any graph package can handle (maybe).

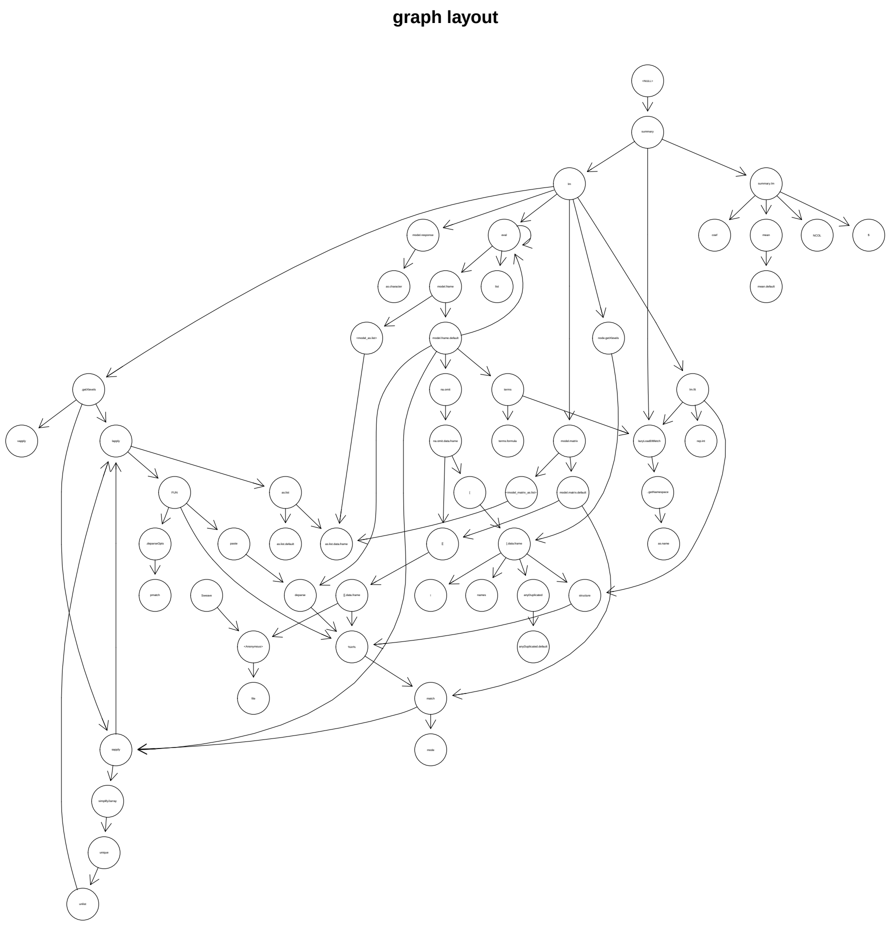
5.2.1. *graph Package.*

Input
```
library(graph)
sprofadjNEL <- as(sprofadj,"graphNEL")
```

Input
```
#24
plot(sprofadjNEL,  main="graph layout", cex.main=2)
#detach("package:graph")
```

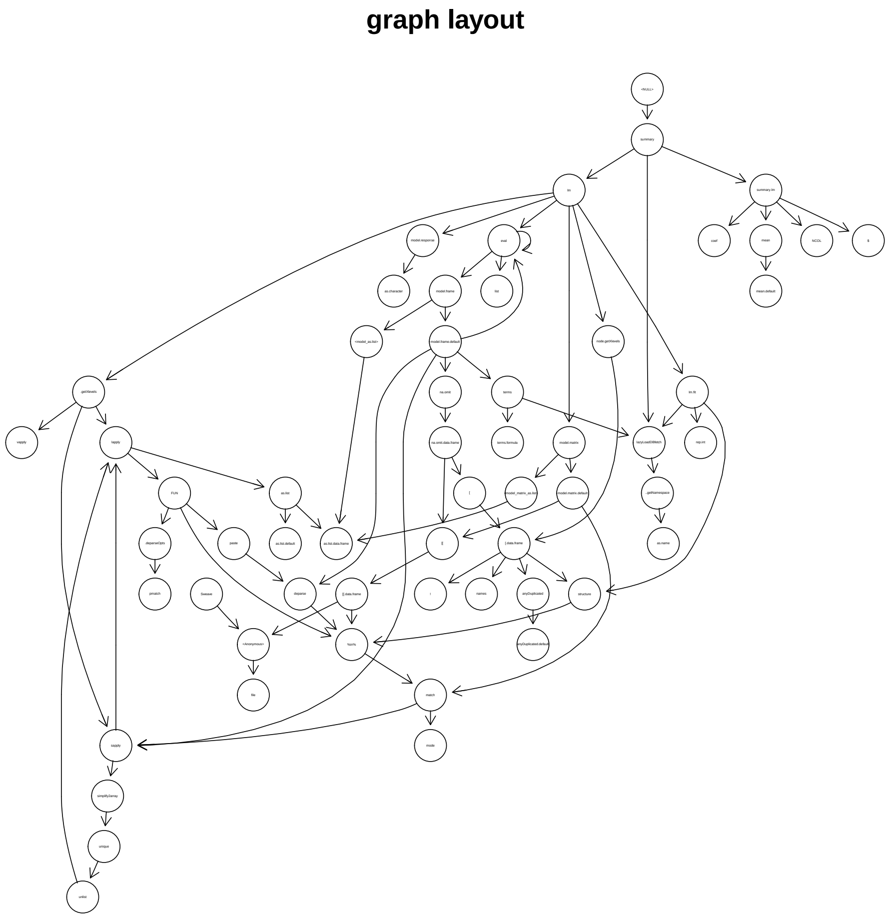


Input
```
#18
plot(sprofadjNEL,  main="graph layout", cex.main=2)
#detach("package:graph")
```



Input

```
#12
plot(sprofadjNEL,  main="graph layout", cex.main=2)
#detach("package:graph")
```


graph layout
<NULL>
summary
lm
summary.lm
model.response
eval
coef
mean
NCOL
$
as.character
model.frame
list
mean.default
<model_as.list>
model.frame.default
node.getXlevels
.getXlevels
na.omit
terms
lm.fit
vapply
lapply
na.omit.data.frame
terms.formula
model.matrix
lazyLoadDBfetch
rep.int
FUN
as.list
[
<model_matrix_as.list>
model.matrix.default
..getNamespace
.deparseOpts
paste
as.list.default
as.list.data.frame
[[
[.data.frame
as.name
pmatch
Sweave
deparse
[[.data.frame
!
names
anyDuplicated
structure
<Anonymous>
%in%
anyDuplicated.default
file
match
sapply
mode
simplify2array
unique
unlist

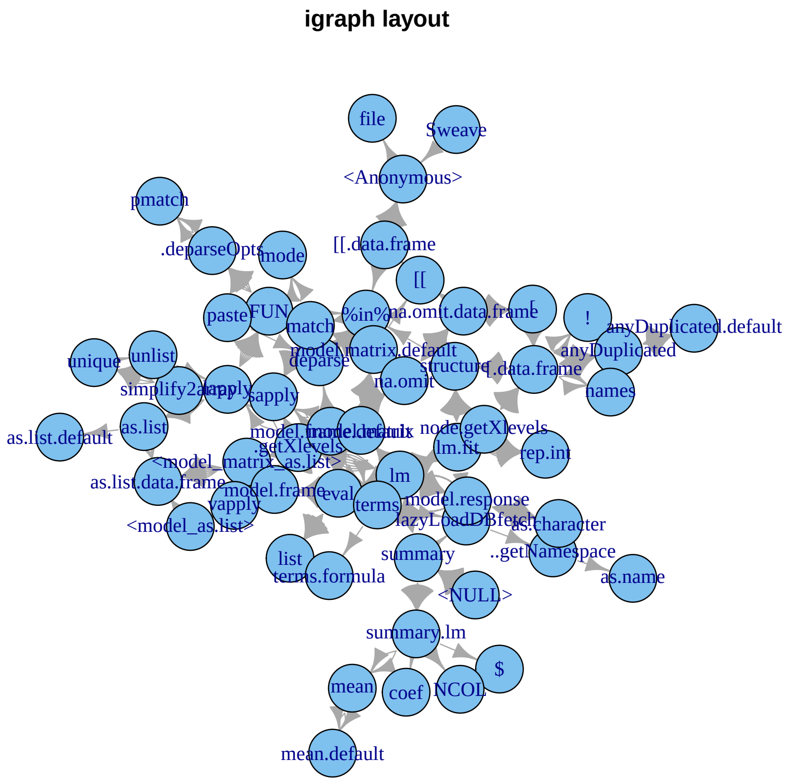
5.2.2. *igraph Package.*

Input

```
library(igraph)
sprofig <- graph.adjacency(sprofadj)
```

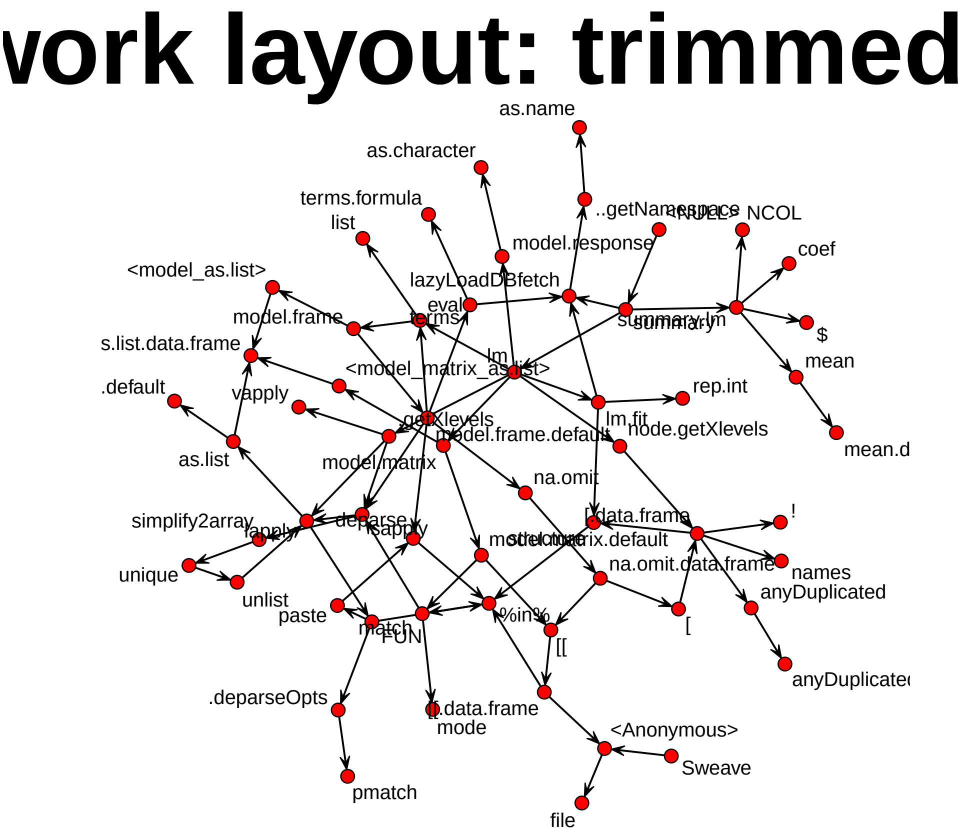
Input

```
#plot(sprofig, main="igraph layout", cex.main=5)
plot(sprofig, main="igraph layout")
detach("package:igraph")
```



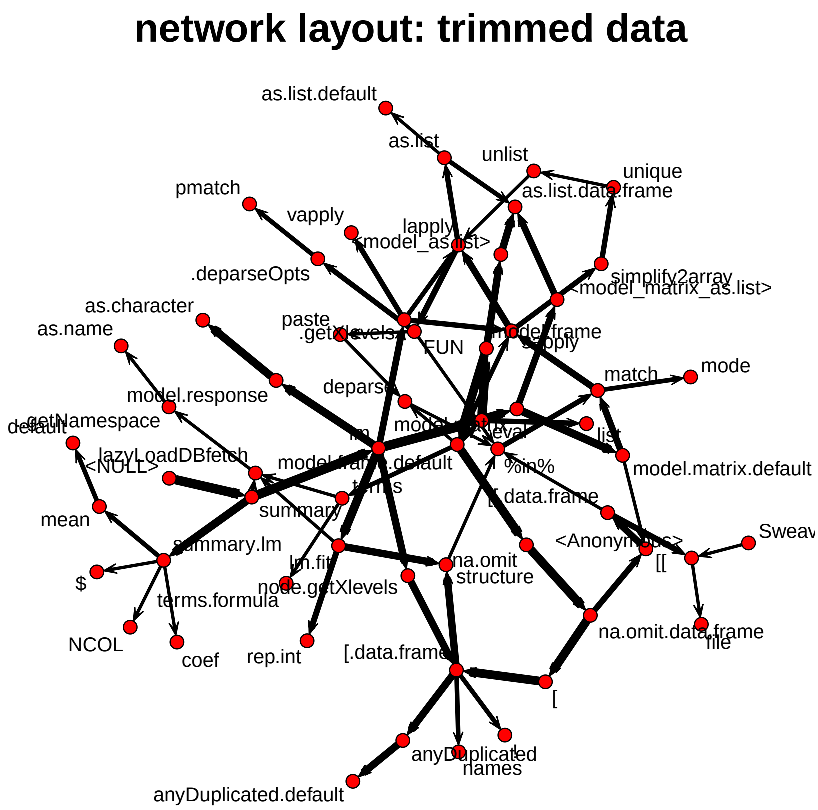
5.2.3. *network Package.*

Input
```
library(network)
nwsprofadj <- as.network(sprofadj) # names is not imported
network.vertex.names(nwsprofadj) <- rownames(sprofadj) # not honoured by plot
plot(nwsprofadj, label=rownames(sprofadj), main="network layout: trimmed data", cex.main=5)
```



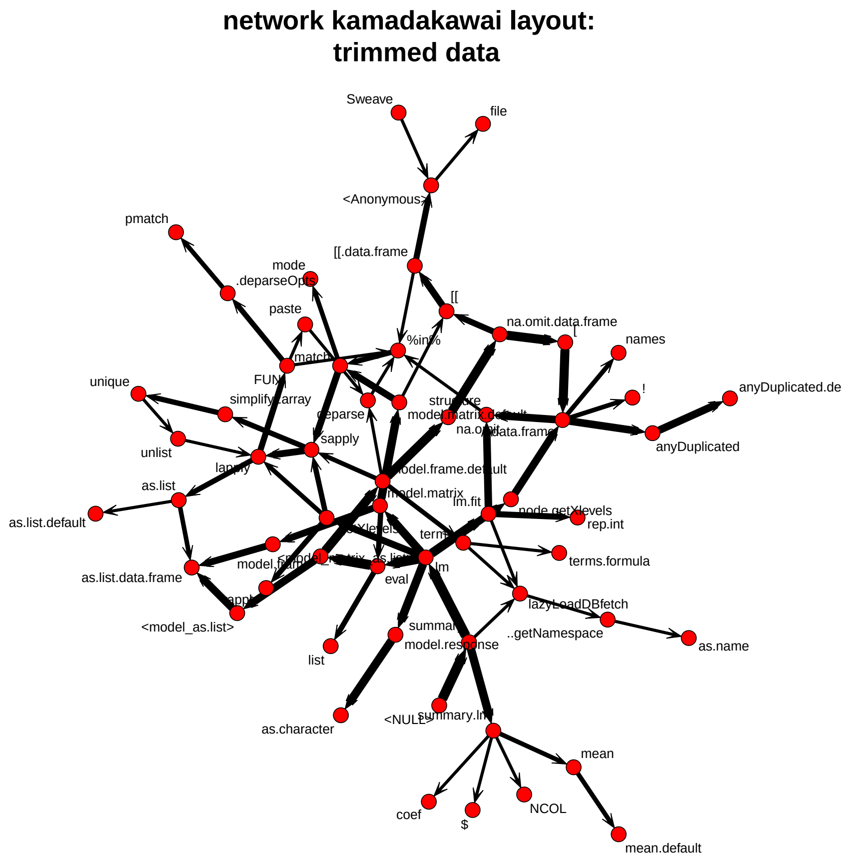
Input

```
edge.lwd<-sprofadj
edge.lwd[edge.lwd>0]<- rank(edge.lwd[edge.lwd>0], ties.method="max")
#edge.lwd <- trunc(sprofadj/max(sprofadj)*10)+1
edge.lwd <- round(edge.lwd/max(edge.lwd)*12)
plot(nwsprofadj, label=rownames(sprofadj), main="network layout: trimmed data", cex.main=2, edge.lw
```



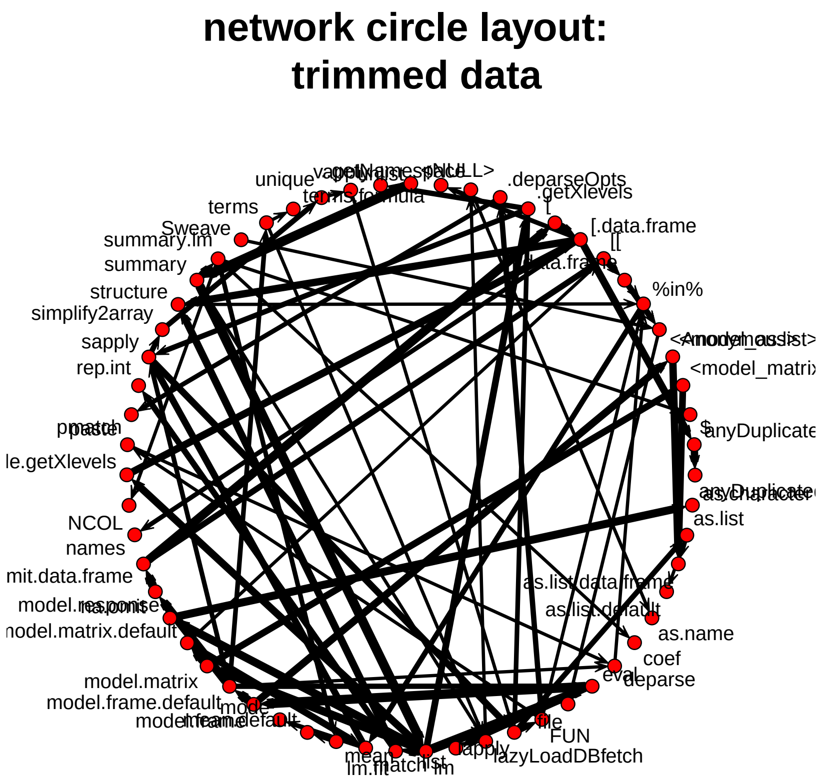
Input

```
#12
plot(nwsprofadj, label=rownames(sprofadj),
main="network kamadakawai layout: \n trimmed data",
mode="kamadakawai",
cex.main=2, edge.lwd=edge.lwd)
```



Input

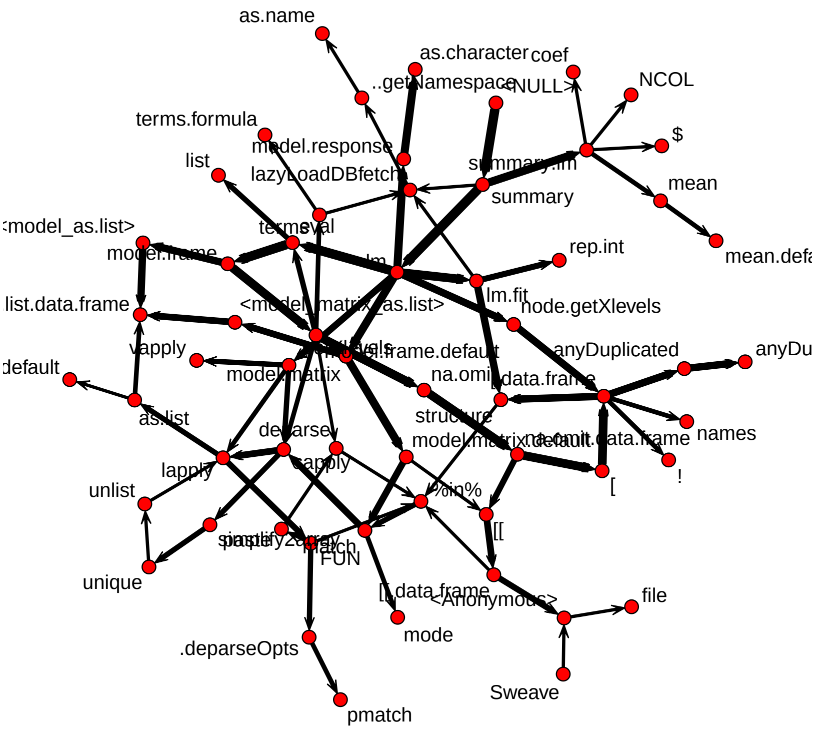
```
plot(nwsprofadj, label=rownames(sprofadj),
main="network circle layout: \n trimmed data",
mode="circle",
cex.main=2, edge.lwd=edge.lwd)
```



Input

```
plot(nwsprofadj, label=rownames(sprofadj),
main="network fruchtermanreingold layout: \n trimmed data",
mode="fruchtermanreingold",
cex.main=2, edge.lwd=edge.lwd)
detach("package:network")
```

network fruchtermanreingold layout: trimmed data



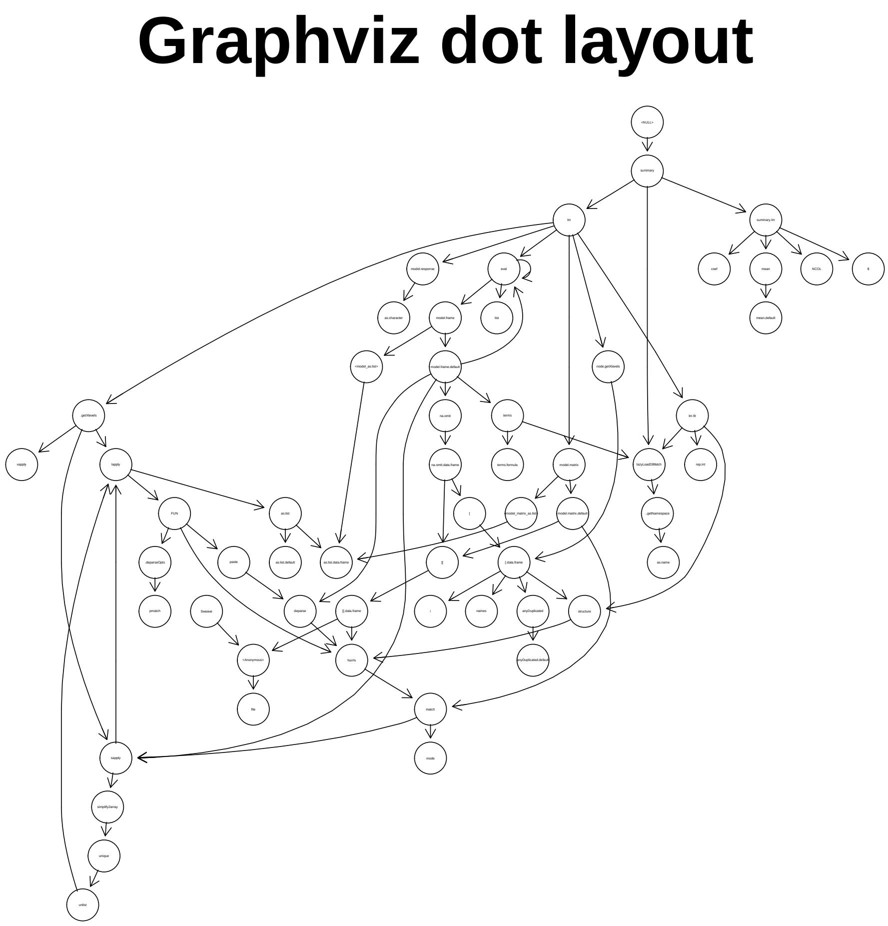
5.2.4. *Rgraphviz Package.*

Input

```
library(Rgraphviz)
sprofadjRag <- agopen(sprofadjNEL, name="Rprof Example")
```

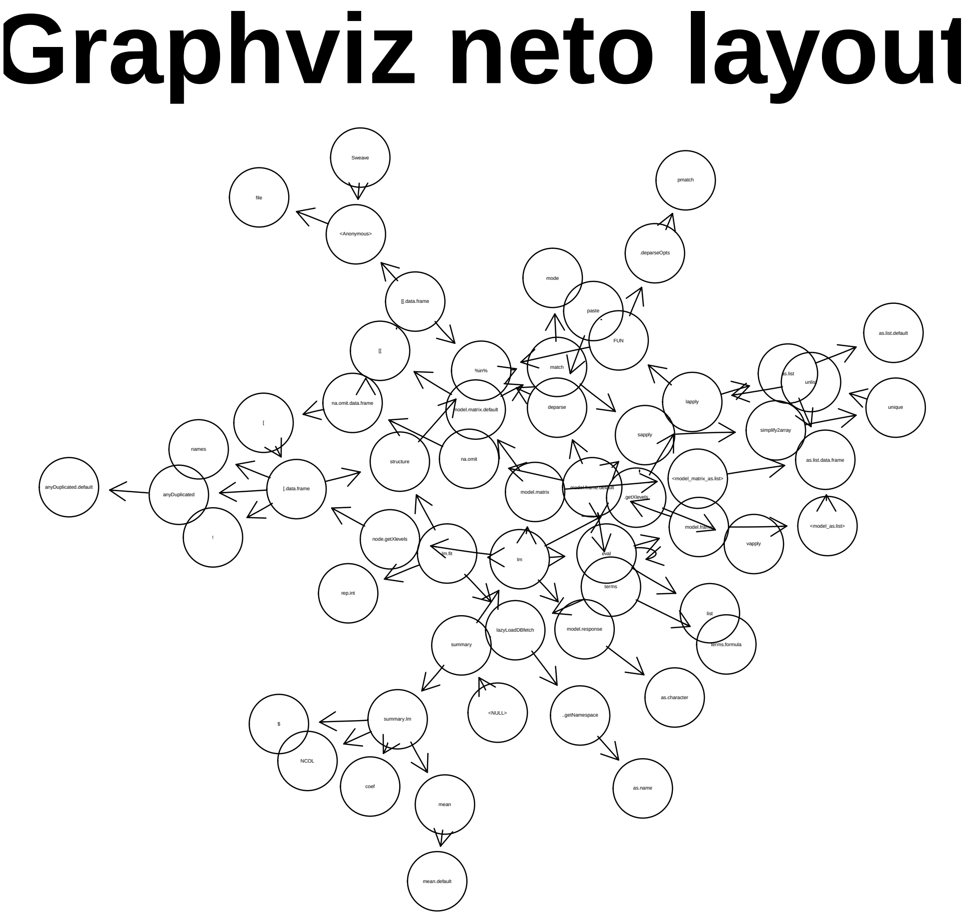
Input

```
#12
plot(sprofadjRag, main="Graphviz dot layout", cex.main=5)
```

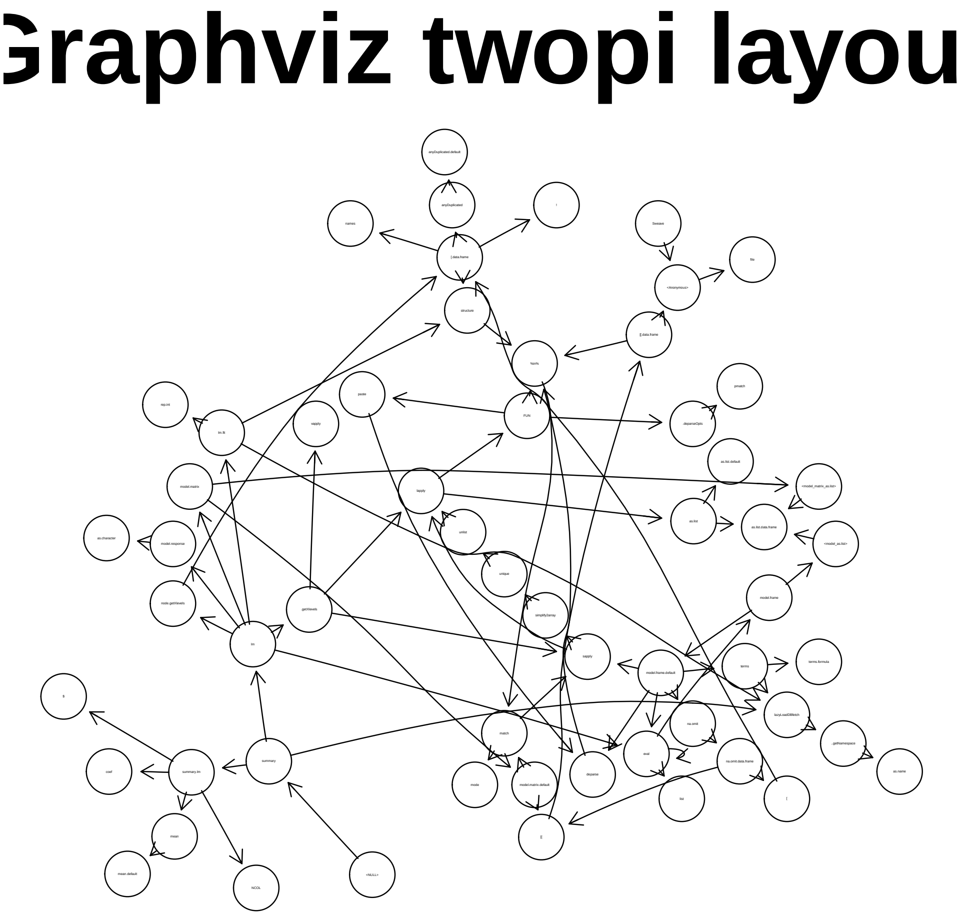


Input

```
plot(sprofadjRag,"neato", main="Graphviz neto layout", cex.main=5)
```

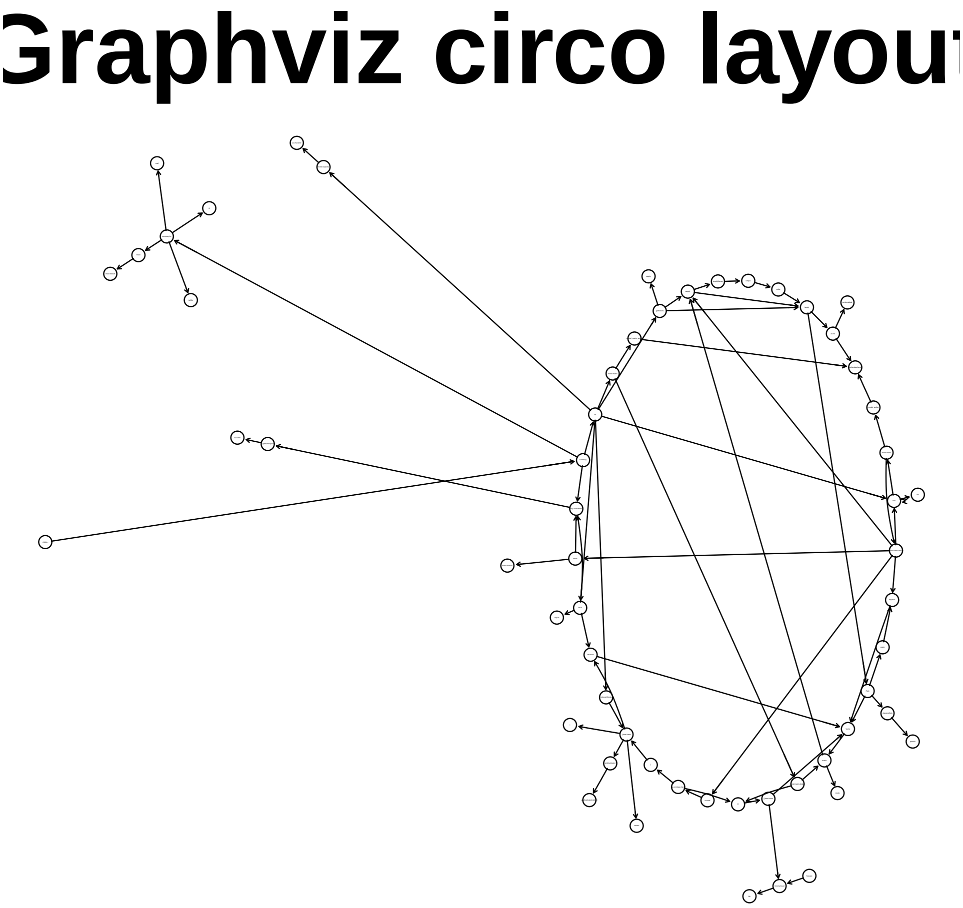


Input
```
plot(sprofadjRag,"twopi", main="Graphviz twopi layout", cex.main=5)
```

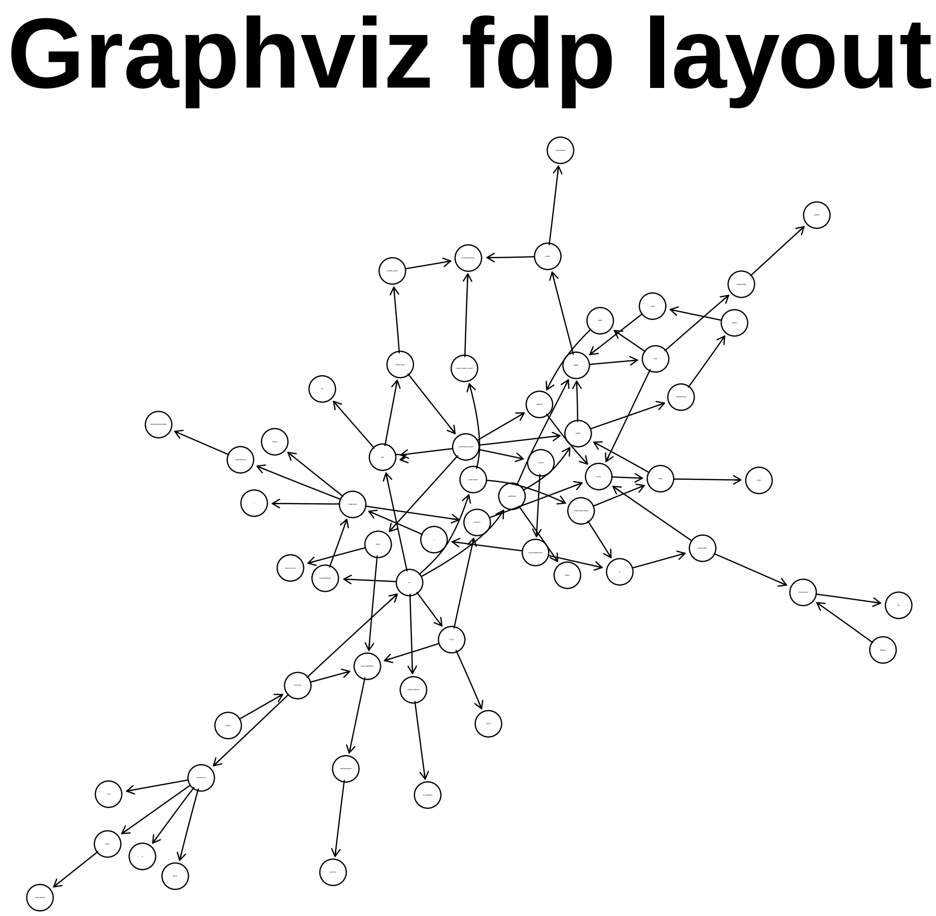


Input

```
plot(sprofadjRag,"circo", main="Graphviz circo layout", cex.main=5)
```



Input
```
plot(sprofadjRag,"fdp", main="Graphviz fdp layout", cex.main=5)
```



Index

R session info:

- R version 3.0.1 (2013-05-16), `x86_64-apple-darwin10.8.0`
- Locale: `en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: graph 1.38.2, RColorBrewer 1.0-5, Rcpp 0.10.3, Rgraphviz 2.4.0, sna 2.3-1, sprof 0.0-5, wordcloud 2.4, xtable 1.7-1
- Loaded via a namespace (and not attached): BiocGenerics 0.6.0, igraph 0.6.5-2, network 1.7.2, parallel 3.0.1, slam 0.1-28, stats4 3.0.1, tools 3.0.1

LaTeX information:

textwidth: 4.9823in linewidth:4.9823in
textheight: 8.0824in

Svn repository information:

$HeadURL: svnssh://gsawitzki@svn.r-forge.r-project.org/svnroot/sintro/pkg/sprof/vignettes/sprofiling.Rnw +

$Source: /u/math/j40/cvsroot/lectures/src/insider/profile/Rnw/profile.Rnw,v $

$Id: sprofiling.Rnw 175 2013-07-17 18:28:50Z gsawitzki $

$Revision: 175 $

$Date: 2013-07-17 20:28:50 0200($Wed, 17 Jul 2013$)+

$name: $

$Author: gsawitzki $

Günther Sawitzki
StatLab Heidelberg
Im Neuenheimer Feld 294
D 69120 Heidelberg

E-mail address: gs@statlab.uni-heidelberg.de

URL: http://sintro.r-forge.r-project.org/