Efficient list recursion in R with {rrapply}

Joris Chau (Open Analytics)

July 06, 2021

UseR! 2021

List recursion with base R

Example data: `renewable_energy_by_country` is a nested list with renewable energy shares (% total energy consumption) per country in 2016[1]

```r
library(rrapply)
data("renewable_energy_by_country")
```

```
#> List of 1
#>  $ World:List of 6
#>   ..$ Africa    :List of 2
#>   .. ..$ Northern Africa   :List of 7
#>   .. .. ..$ Algeria         : num 0.08
#>   .. .. ..$ Egypt           : num 5.69
#>   .. .. .. [list output truncated]
#>   .. ..$ Sub-Saharan Africa:List of 4
#>   .. .. ..$ Eastern Africa :List of 22
#>   .. .. .. ..$ British Indian Ocean Territory: logi NA
#>   .. .. .. ..$ Burundi                       : num 89.2
....
```

[1] Source: United Nations Open SDG Data Hub UNSD-SDG07 (`https://www.sdg.org/`)

List recursion with base R

Exercise 1: Replace all missing values by zero while maintaining the structure of the list

```r
rapply(
  renewable_energy_by_country,
  f = function(x) replace(x, is.na(x), 0),
  how = "replace"
)
```

```
#> List of 1
#>  $ World:List of 6
#>   ..$ Africa    :List of 2
#>   .. ..$ Northern Africa   :List of 7
#>   .. .. ..$ Algeria         : num 0.08
#>   .. .. ..$ Egypt           : num 5.69
#>   .. .. .. [list output truncated]
#>   .. ..$ Sub-Saharan Africa:List of 4
#>   .. .. ..$ Eastern Africa :List of 22
#>   .. .. .. ..$ British Indian Ocean Territory: num 0
#>   .. .. .. ..$ Burundi                       : num 89.2
....
```

List recursion with base R

Exercise 2: Filter the European countries with a renewable energy share $>$ 50% (while maintaining the structure of the list)

List recursion with base R

Exercise 2: Filter the European countries with a renewable energy share $> 50\%$ (while maintaining the structure of the list)

```r
## example list recursion code
filt_fun <- function(x, eu = FALSE) {
  i <- 1
  while(i <= length(x)) {
    if(is.numeric(x[[i]]) && x[[i]] > 50 && eu) {
      i <- i + 1
    } else {
      if(is.list(x[[i]])) {
        val <- Recall(x[[i]], (eu || identical(names(x)[i], "Europe")))
        x[[i]] <- val
        i <- i + !is.null(val)
      } else {
        x[[i]] <- NULL
      }
      if(all(sapply(x, is.null))) {
        x <- NULL
      }
    }
  }
  return(x)
}
```

List recursion with base R

Exercise 2: Filter the European countries with a renewable energy share $>$ 50% (while maintaining the structure of the list)

```
filt_fun(renewable_energy_by_country)
```

```
#> List of 1
#>  $ World:List of 1
#>   ..$ Europe:List of 2
#>   .. ..$ Northern Europe:List of 3
#>   .. .. ..$ Iceland: num 78.1
#>   .. .. ..$ Norway : num 59.5
#>   .. .. ..$ Sweden : num 51.4
#>   .. ..$ Western Europe :List of 1
#>   .. .. ..$ Liechtenstein: num 62.9
```

List recursion with base R

Exercise 2: Filter the European countries with a renewable energy share $>$ 50% (while maintaining the structure of the list)

```r
filt_fun(renewable_energy_by_country)
```

```
#> List of 1
#>  $ World:List of 1
#>   ..$ Europe:List of 2
#>   .. ..$ Northern Europe:List of 3
#>   .. .. ..$ Iceland: num 78.1
#>   .. .. ..$ Norway : num 59.5
#>   .. .. ..$ Sweden : num 51.4
#>   .. ..$ Western Europe :List of 1
#>   .. .. ..$ Liechtenstein: num 62.9
```

This works, but the code is **difficult** to follow and/or reason about

List recursion with {rrapply}

{`rrapply`}

`rrapply` reimplements and enhances `rapply` building on its native `C` implementation
$\rightarrow$ no other package dependencies

List recursion with {rrapply}

{rrapply}

`rrapply` reimplements and enhances `rapply` building on its native `C` implementation
→ no other package dependencies

Example:

```r
rrapply(
  renewable_energy_by_country,
  condition = function(x, .xparents) x > 50 && "Europe" %in% .xparents,
  how = "prune"
)
```

```
#> List of 1
#>  $ World:List of 1
#>   ..$ Europe:List of 2
#>   .. ..$ Northern Europe:List of 3
#>   .. .. ..$ Iceland: num 78.1
#>   .. .. ..$ Norway : num 59.5
#>   .. .. ..$ Sweden : num 51.4
#>   .. ..$ Western Europe :List of 1
#>   .. .. ..$ Liechtenstein: num 62.9
```

Selected {rrapply} examples

Example: Additional options to structure the returned result, e.g. `how = "melt"`

```r
rrapply(
  renewable_energy_by_country,
  classes = "numeric",
  how = "melt"
)
```

```
#>        L1     L2               L3          L4    L5 value
#> 1   World Africa  Northern Africa     Algeria  <NA>  0.08
#> 2   World Africa  Northern Africa       Egypt  <NA>  5.69
#> 3   World Africa  Northern Africa       Libya  <NA>  1.64
....
```

Selected {rrapply} examples

Example: Additional options to structure the returned result, e.g. `how = "melt"`

```
rrapply(
  renewable_energy_by_country,
  classes = "numeric",
  how = "melt"
)
```

```
#>       L1     L2                   L3           L4    L5 value
#> 1  World Africa    Northern Africa      Algeria  <NA>  0.08
#> 2  World Africa    Northern Africa        Egypt  <NA>  5.69
#> 3  World Africa    Northern Africa        Libya  <NA>  1.64
....
```

```
large_list <- replicate(1000, renewable_energy_by_country, simplify = FALSE)
system.time(rrapply(large_list, how = "melt"))
#>    user  system elapsed
#>   0.093   0.020   0.113
```

```
system.time(reshape2::melt(large_list))
#>    user  system elapsed
#>  48.148   0.008  48.156
```

References

For more information:

- Browse the vignettes at: `https://jorischau.github.io/rrapply/`
- Download from CRAN at: `https://cran.r-project.org/package=rrapply`
- Browse the source code at: `https://github.com/JorisChau/rrapply/`