R: Data structures

Sławek Staworko

Univ. Lille 3

2018, January

Atomic values

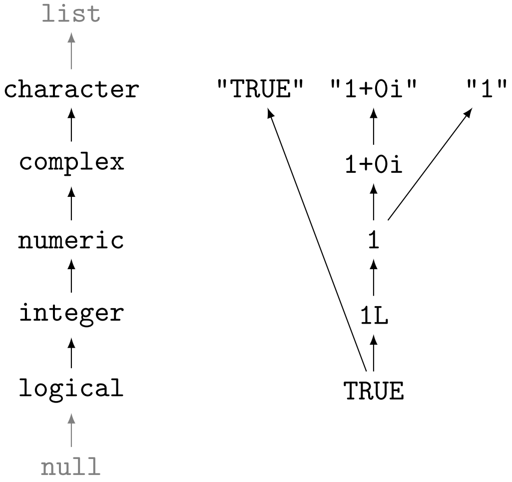
Literals

- `TRUE, FALSE, T, F` : `logical`
- `1L, -20L` : `integer`
- `10, 10.1, -3.14` : `numerical`
- `2+2i` : `complex`
- `"abc", "", "John Smith"` : `character`

Big literals

- `1.5 1.5e2` $\equiv$ `1.5*10`^{2} : `numerical`
- `-2e3L` $\equiv$ `-2000L` : `integer`

Type hierarchy and coercion



Type coercion (contd.)

The class of an object

- `class(TRUE)` $\mapsto$ `"logical"`
- `class(1L)` $\mapsto$ `"integer"`
- `class(1)` $\mapsto$ `"numeric"`
- `class(1.0)` $\mapsto$ `"numeric"`
- `class(1+1i)` $\mapsto$ `"complex"`

Not exactly the same as the type

- `typeof(1.0)` $\mapsto$ `"double"`
- `typeof(1L)` $\mapsto$ `"integer"`
- `typeof(TRUE)` $\mapsto$ `"logical"`

Type coercion (contd.)

Explicit coercion

- `as.integer(TRUE)` $\mapsto$ `1L`
- `as.character(TRUE)` $\mapsto$ `"TRUE"`

Implicit coercion

- `1 + 2` $\mapsto$ `3` : `numerical`
- `1L + 2L` $\mapsto$ `3L` : `integer`
- `1L + 2` $\mapsto$ `3` : `numerical`
- `TRUE + 2L` $\mapsto$ `3L` : `integer`
- `3 * FALSE` $\mapsto$ `0` : `numerical`
- `paste("Jean","Dubois")` $\mapsto$ `"Jean Dubois"` : `character`
- `paste("abc", 2, F)` $\mapsto$ `"abc 2 FALSE"` : `character`

Vectors

vector is a sequence (ordered sequence) of elements of the same atomic type.

Is it horizontal or vertical?

Most operations interprets vectors in a *flexible* way:
when R displays `1.5 -0.5 4.1` the vector can be viewed as

$$\begin{bmatrix} 1.5 \\ -0.5 \\ 4.1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1.5 & -0.5 & 4.1 \end{bmatrix}$$

Building block of all expressions in R!

A singular value is a vector consisting of one element:

`1.25` interpreted by R is essentially $\begin{bmatrix} 1.25 \end{bmatrix}$

Combining contents of several vectors

The function c

Collates the contents of any number of given vectors

$$\mathtt{c}(1.5, -0.5, 4.1) \mapsto \begin{bmatrix} 1.5 \\ -0.5 \\ 4.1 \end{bmatrix} : \mathtt{numerical}$$

$$\mathtt{c}(-2\mathtt{L}, 3\mathtt{L}, 7\mathtt{L}) \mapsto \begin{bmatrix} -2\mathtt{L} \\ 3\mathtt{L} \\ 7\mathtt{L} \end{bmatrix} : \mathtt{integer}$$

Combining contents of several vectors (contd.)

Implicit type coercion

$$\mathtt{x} \leftarrow \mathtt{c}(1.5, -0.5, 4.1)$$

$$\mathtt{y} \leftarrow \mathtt{c}(-2L, 3L, 7L)$$

$$\mathtt{c}(\mathtt{x}, \mathtt{y}) \mapsto \begin{bmatrix} 1.5 \\ -0.5 \\ 4.1 \\ -2.0 \\ 3.0 \\ 7.0 \end{bmatrix} : \mathtt{numerical}$$

Is `c` associative?

Associative operations $\oplus$

The order of evaluation does not matter i.e,

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

and therefore we can forgo the parentheses and simply write

$$a \oplus b \oplus c$$

The function `c` does seem to be associative

Both `c(c(1,2),3)` and `c(1,c(2,3))` yield

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} : \texttt{numerical}$$

Is c associative? (contd.)

However, implicit type coercion complicates things

$$\mathtt{c}(\mathtt{c}(\mathtt{TRUE}, 2), "3") \mapsto \begin{bmatrix} "1" \\ "2" \\ "3" \end{bmatrix} : \mathtt{character}$$

$$\mathtt{c}(\mathtt{TRUE}, \mathtt{c}(2, "3")) \mapsto \begin{bmatrix} "\mathtt{TRUE}" \\ "2" \\ "3" \end{bmatrix} : \mathtt{character}$$

Creating vectors

Using atomic type constructor

- `integer(5)` $\mapsto$ `0 0 0 0 0`
- `character(3)` $\mapsto$ `"" "" ""`

Using the repetition function `rep`

- `rep(1, 4)` $\mapsto$ `1 1 1 1`
- `rep(c(2,6), 3)` $\mapsto$ `2 6 2 6 2 6`

Regular sequences generation

- `1:5` $\mapsto$ `1 2 3 4 5 : integer`
- `seq(1,10,2)` $\mapsto$ `1 3 5 7 9 : numerical`

Getting the size of a vector

- `length(c("a",TRUE,2))` $\mapsto$ `3`

Missing values

`NA` is a special (logical) constant indicating a missing values

Most functions are sensitive to missing data

- `sum(c(1,2,NA))` $\mapsto$ `NA` : `numerical`
- `TRUE && NA` $\mapsto$ `NA`
- `TRUE || NA` $\mapsto$ `TRUE`

It can be basically coerced into any atomic type

`character`	`NA_character_`
↑	↑
`complex`	`NA_complex_`
↑	↑
`numeric`	`NA_real_`
↑	↑
`integer`	`NA_integer_`
↑	↑
`logical`	`NA`

Point-wise operations

Arithmetic operations are performed point-wise

- `c(3,5,7) + c(1,3,5)` $\mapsto$ `4 8 12`

Standard coercion rules apply

- `c(3,5,7) * c(1L,3L,5L)` $\mapsto$ `3 15 35 : numerical`

Point-wise Boolean operators

The short operators `|` (or), `&` (and), and `!` (negation)

- `c(TRUE,FALSE) | c(FALSE,FALSE)` $\mapsto$ `TRUE FALSE`

Long Boolean operators

The operators `||` and `&&` evaluate on the first element only

- `c(TRUE,FALSE) || c(FALSE,FALSE)` $\mapsto$ `TRUE`
- `c(TRUE,FALSE) && c(FALSE,FALSE)` $\mapsto$ `FALSE`

Recycling

What happens if the operands are of different length?

- The shorter is REpeated in CYCLE
- `c(3,6,9,12) + c(2,4)` $\equiv$
  `c(3,6,9,12) + c(2,4,2,4)` $\mapsto$ `5 10 11 16`

Length should be compatible

- A warning is raised if the length of one vector is not a multiple of the length of the other.
- `c(3,6,9,12,15) + c(2,4)` $\equiv$
  `c(3,6,9,12,15) + c(2,4,2,4,2)` $\mapsto$ `5 10 11 16 17`

Addressing and slicing vector

Using indices and ranges

- `v ← c(1,3,5,7,9)`
- `v[2]` $\mapsto$ `3`
- `v[c(1,3,4)]` $\mapsto$ `1 5 7`
- `v[2:4]` $\mapsto$ `3 5 7`
- `v[1] ← 0` changes `v` to `0 3 5 7 9`
- `v[2:5] ← c(2,4)` further changes `v` to `0 2 4 2 4` (recycling)

Elementary access functions

- `v[2]` is equivalent to `` `[`(v,2) ``
- `v[1] ← 0` is equivalent to `` `[←`(v,1,0) ``

Matrices

Matrix is internally represented as a vector

$$\mathtt{m} \leftarrow \begin{bmatrix} 0 & 6 & 12 & 18 \\ 2 & 8 & 14 & 20 \\ 4 & 10 & 16 & 22 \end{bmatrix}$$

is represented as

$$0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \quad 18 \quad 20 \quad 22$$

Matrix can be addressed as a vector or array

- `m[4]` $\mapsto$ 6
- `m[2,3]` $\mapsto$ 14
- `m[1:3]` $\equiv$ `m[,1]` $\mapsto$ 0 2 4
- `m[seq(1,12,3)]` $\equiv$ `m[1,]` $\mapsto$ 0 6 12 18
- `m[1:2,2:4]` $\mapsto \begin{bmatrix} 6 & 12 & 18 \\ 8 & 14 & 20 \end{bmatrix}$

Matrix creation

From a vector

- `matrix(1:6,nrow=2,ncol=3)` $\mapsto \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$
- `matrix(1:6,nrow=2,ncol=3,byrow=TRUE)` $\mapsto \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

By column/row addition

- `rbind(1:3,4:6)` $\mapsto \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
- `cbind(1:2,3:4,5:6)` $\mapsto \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

Matrix operations

Point-wise operations

- both arrays must have the same dimensions
- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 6 & 15 \\ 8 & 20 & 36 \end{bmatrix}$

Vectors can be recycled

- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \texttt{c(0,1)} \mapsto \begin{bmatrix} 0 & 0 & 0 \\ 4 & 5 & 6 \end{bmatrix}$
- vector is interpreted column-wise.

Matrix multiplication

- $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \%{*}\% \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \mapsto \begin{bmatrix} 22 & 49 \\ 28 & 64 \end{bmatrix}$

Decorating vectors and matrices

Giving names to columns and rows

- `demand ← c(2,5)`
- `names(demand) ← c("Apples", "Oranges")`
- `price ← matrix(c(1,1.25,2.75,3),2,2)`
- `colnames(price) ← c("Apples", "Oranges")`
- `rownames(m) ← c("Carrefour", "Monoprix")`

Names are preserved by most of operations

```
price %*% demand
```

$$\downarrow$$

```
           [,1]
Carrefour 15.75
Monoprix  17.50
```

Multi-dimensional arrays

Arrays are generalizations of matrices

- may have more than two dimensions
- but are still represented with a vector, and consequently,
- all of its elements are of the same atomic type

Example

`a = array(1:24,dim=c(4,3,2))` : $\texttt{integer}^{4\times3\times2}$

$$\downarrow$$

$$\begin{bmatrix} a_{1,1,1}=1 & a_{1,2,1}=5 & a_{1,3,1}=9 \\ a_{2,1,1}=2 & a_{2,2,1}=6 & a_{2,3,1}=10 \\ a_{3,1,1}=3 & a_{3,2,1}=7 & a_{3,3,1}=11 \\ a_{4,1,1}=4 & a_{4,2,1}=8 & a_{4,3,1}=12 \end{bmatrix} \begin{bmatrix} a_{1,1,2}=13 & a_{1,2,2}=17 & a_{1,3,2}=21 \\ a_{2,1,2}=14 & a_{2,2,2}=18 & a_{2,3,2}=22 \\ a_{3,1,2}=15 & a_{3,2,2}=19 & a_{3,3,2}=23 \\ a_{4,1,2}=16 & a_{4,2,2}=20 & a_{4,3,2}=24 \end{bmatrix}$$

Lists

Data type for collections of values of different types

- `l` $\leftarrow$ `list(name="John Smith", salary=30000)`

Convenient addressing

- ordered and labeled (labels can be repeated)
  `l[[1]]` $\mapsto$ `"John Smith"`
  `l$salary` $\mapsto$ `30000`
- allows *sublisting*
  `l[1]` $\mapsto$ `list(name="John Smith")`

Concatenation with the function c

- `c(list(name="John",age=35),list(city="NYC"))`

$$\downarrow$$

`list(name="John",age=35,city="NYC")`

Factors

Categorical variables

Variable that takes a limited number of possible values (called levels). The set of possible values may be (linearly) ordered. E.g.,

- gender (unordered): `M` (male), `F` (female), `F2M` (female-to-male), `M2F` (male-to-female), `I` (intersex), `A` (agender)
- education (ordered): `P` (primary school), `HS` (high school), `B` (bachelor), `M` (master), `D` (doctorate)

`factor` is a enumerated data type

a specialization of a vector, typically of atomic `character` type.

Example

```
list(gender=factor(c('M','F','M','F2M')),
     edu=factor(c('M','M','D','B'), ordered=TRUE,
                levels=c('P','HS','B','M','D')),
     income=c(65000, 25000, 30000, 55000))
```

Data frames

`data.frame` specialized list for representing tabular database

- its components (columns) are vectors of the same length
- `character` vectors are by default coerced to factors

Example

```
data.frame(income=c(30000, 20000),
           list(edu=c('HS','M'), gender=c('F','M')),
           matrix(c(1.0,1.2,0.75,0.76,0.99,0.81),2,3))
```

$$\downarrow$$

income	edu	gender	X1	X2	X3
30000	HS	F	1.0	0.75	0.99
20000	M	M	1.2	0.76	0.81