Package 'pmatch'

February 6, 2018

Type Package

Title Pattern Matching

Version 0.1.1

Author Thomas Mailund <mailund@birc.au.dk>

Maintainer Thomas Mailund <mailund@birc.au.dk>

Description Implements type constructions and pattern matching.
Using this package, you can specify a type of object and write functions
that matches against the structure of objects of such types to program
data structure transformations more succinctly.

License GPL-3

Depends R (>= 3.2), dplyr, purrr, rlang, tibble, glue

Suggests shiny, DT, covr, styler, lintr, testthat, knitr, rmarkdown,
magrittr

ByteCompile true

Encoding UTF-8

RoxygenNote 6.0.1

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Repository CRAN

Date/Publication 2018-02-06 09:17:24 UTC

R topics documented:

`:=` *Define a new data type from a sequence of constructors.*

Description

This assignment operator introduces a domain-specific language for specifying new types. Types are defined by the ways they can be constructed. This is provided as a sequence of `|`-separated constructors, where a constructor is either a constant, i.e., a bare symbol, or a function.

Usage

```
":="(data_type, constructors)
```

Arguments

`data_type`	The name of the new data type. Should be given as a bare symbol.
`constructors`	A list of `|`-separated constructor specifications.

Details

We can construct an enumeration like this:

```
numbers := ONE | TWO | THREE
```

This will create the type `numbers` and three constants, `ONE`, `TWO`, and `THREE` that can be matched against using the `cases` function

```
x <- TWO cases(x,      ONE -> 1,      TWO -> 2,      THREE -> 3)
```

Evaluating the `cases` function will compare the value in `x` against the three patterns and recognize that `x` holds the constant `TWO` and it will then return `2`.

With function constructors we can create more interesting data types. For example, we can create a linked list like this

```
linked_list := NIL | CONS(car, cdr : linked_list)
```

This expression defines constant `NIL` and function `CONS`. The function takes two arguments, `car` and `cdr`, and requires that `cdr` has type `linked_list`. We can create a list with three elements, 1, 2, and 3, by writing

```
CONS(1, CONS(2, CONS(3, NIL)))
```

and we can, e.g., test if a list is empty using

```
cases(lst, NIL -> TRUE, CONS(car,cdr) -> FALSE)
```

A special pattern, `otherwise`,can be used to capture all patterns, so the emptiness test can also be written

```
cases(lst, NIL -> TRUE, otherwise -> FALSE)
```

Arguments to a constructor function can be typed. To specify typed variables, we use the `:`-operator. The syntax is then `var : type`. The type will be checked when you construct a value using the constructor.

Examples

```
linked_list := NIL | CONS(car, cdr : linked_list)
lst <- CONS(1, CONS(2, CONS(3, NIL)))
len <- function(lst, acc = 0) {
    cases(lst,
          NIL -> acc,
          CONS(car,cdr) -> len(cdr, acc + 1))
}
len(lst)

list_sum <- function(lst, acc = 0) {
    cases(lst,
          NIL -> acc,
          CONS(car,cdr) -> list_sum(cdr, acc + car))
}
list_sum(lst)
```

`cases` *Dispatches from an expression to a matching pattern*

Description

Given an expression of a type defined by the `:=` operator, `cases` matches it against patterns until it find one that has the same structure as `expr`. When it does, it evaluates the expression the pattern is associated with. During matching, any symbol that is not quasi-quoted will be considered a variable, and matching values will be bound to such variables and be available when an expression is evaluated.

Usage

```
cases(expr, ...)
```

Arguments

`expr`	The value the patterns will be matched against.
`...`	A list of `pattern -> expression` statements.

Value

The value of the expression associated with the first matching pattern.

See Also

`:=`

Examples

```
linked_list := NIL | CONS(car, cdr : linked_list)
lst <- CONS(1, CONS(2, CONS(3, NIL)))
len <- function(lst, acc = 0) {
    cases(lst,
          NIL -> acc,
          CONS(car,cdr) -> len(cdr, acc + 1))
}
len(lst)

list_sum <- function(lst, acc = 0) {
    cases(lst,
          NIL -> acc,
          CONS(car,cdr) -> list_sum(cdr, acc + car))
}
list_sum(lst)
```

Index