Единицы измерения и Active Patterns в F#

Артемий Патов

19.05.2023

Единицы измерения

- Позволяет добавить дополнительную информацию о числовых типах
- Нужны при проверке типов во время компиляции
- В рантайме размываются

```
// Mass, kilograms.
[<Measure>] type kg
// Distance, meters.
[<Measure>] type m
// Time, seconds.
[<Measure>] type s
// Force, Newtons.
[<Measure>] type N = kg m / s^2
```

Пример

```
[<Measure>] type degC // temperature, Celsius/Centigrade
[<Measure>] type degF // temperature, Fahrenheit

let convertCtoF temp = 9.0<degF> / 5.0<degC> * temp + 32.0<degF>

let convertFtoC temp = 5.0<degC> / 9.0<degF> * (temp - 32.0<degF>)

// Define conversion functions from dimensionless floating point values.
let degreesFahrenheit temp = temp * 1.0<degF>
let degreesCelsius temp = temp * 1.0<degC>

let toDimensionless (deg: float<degC>) = deg / 1.0<degC>
```

Pattern Matching (ограничения)

- Не first-class (нельзя относиться к ним, как к данным)
- Можно использовать только 16 предопределенных шаблонов, о которых знает компилятор

Active Patterns

- Способ расширения паттерн матчинга в F#
- Способ сделать first-class паттерн матчинг

Single-case Total Patterns

- Используется при декомпозиции данных

```
open System.Numerics

let (|Rect|) (x: Complex) =
    Rect (x.Real, x.Imaginary)

let add one two =
    match (one, two) with
    | (Rect (r1, i1), Rect (r2, i2)) ->
        Complex (r1 + r2, i1 + i2)
```

Multiple-case Total Patterns

```
let (|Even|Odd|) input = if input % 2 = 0 then Even else Odd

let TestNumber input =
  match input with
  | Even -> printfn "%d is even" input
  | Odd -> printfn "%d is odd" input
```

Partial Active Patterns

```
let (|Integer|_|) (str: string) =
  match System.Int32.TryParse(str) with
  | (true, num) -> Some num
  | _ -> None

let (|Bool|_|) (str: string) =
  match System.Boolean.TryParse(str) with
  | (true, num) -> Some num
  | _ -> None

let parseNumeric str =
  match str with
  | Integer i -> printfn "%d : Integer" i
  | Bool b -> printfn "%f : Bool" b
  | _ -> printfn "%s : Not matched." str
```

Parameterized Active Patterns

- Добавляет дополнительные агрументы

```
let (|Default|) onNone value =
    match value with
    | None -> onNone
    | Some e -> e

let greet (Default "random citizen" name) =
    printfn "Hello, %s!" name
```

First-class Patterns

```
let unfold (|Q|_|) input =
  let rec loop values = function
    | Q(v, next) -> loop (v::values) next
    | otherwise -> (List.rev values, otherwise)
  loop [] input

type Expr =
  | App of Expr * Expr
  | Lam of head : string * body : Expr
  | Var of name : string

let (|Lambda|_|) = function
  | Lam(head, body) -> Some(head, body)
  | _ -> None

let (|Lambdas|) expr = unfold (|Lambda|_|) expr
```

Active Patterns

- Внутри — просто функции с метаданными
  - Всё, что можно делать с функцией, можно делать с Active Patterns