Líné vyhodnocování

Někdy by se nám hodilo vyhodnocovat líně. Můžeme to zařídit ručně: místo hodnoty `x` si předáme funkci `x_lazy : unit -> x`, která hodnotu vrátí teprve, až ji budeme chtít, tj. až zavoláme `x_lazy ()`. Takovéhle "líné vyhodnocování" ale nesdílí výsledky –– hodnota se počítá pokaždé znova. Nicméně i to by se dalo řešit.

Můžeme však použít i podporu kompilátoru a typu `Lazy<'a>`.

Typ `Lazy<'a>` v sobě uchovává hodnotu typy `'a`. Tato hodnota ale není vypočtena, dokud není poprvé potřeba. Pokud už je jednou vypočtena, tak se hodnota zapamatuje a příště je vrácena zapamatovaná hodnota.

```
type 'a lazy = Lazy<'a>
type Lazy<'a> =
  member this.Value with get () : 'a
  member this.IsValueCreated with get () : bool
  static member Create : Lazy<'T>
  static member CreateFromValue : Lazy<'T>
  static member Force : unit -> 'T
```

Vytváří se pomocí **`lazy`** `(výraz)`, například **`lazy`** `(1/0)`. Vyhodnotit se dá pomocí `.Force`, nebo pomocí pattern–matchingu jako

```
let force l = match l with
  Lazy x -> x
```

V .NETu 4.0 je `Lazy` jenom zkratka za `System.Lazy`, a **`lazy`** `expr` se převede na **`new`** `System.Lazy<_> (`**`fun`**` () -> expr)`. V F#pu jsou všechny volání **`lazy`** a `Force` chráněné zámky kvůli přístupu z několik vláken, i když samotný `System.Lazy` lze zkonstruovat tak, aby při vyhodnocení zámky nepoužíval.

Použití **`lazy`** –– vzpomeňte si například na testovací framework:

```
  static member max_as_reduce (xs : list<int>) =
    not xs.IsEmpty ==> lazy (List.max xs = List.reduce max xs)
```

Bez slova **`lazy`** by se vyhodnotila pravá strana ještě před vyhodnocením operátory ==>. Pomocí **`lazy`** se místo toho vytvoří jenom hodnota `Lazy<'a>`, která se nevyhodnotí, pokud je `xs` prázdný.

Active patterns, aneb vlastní patterny při matchování

Při pattern matchování můžeme definovat vlastní patterny, a to dvou typů:

♣ Výčtové patterny (jedna a víc alternativ, jedna z nich musí vždy nastat):

```
let (|Sude|Liche|) n = if n &&& 1 = 0 then Sude else Liche
match 3 with | Sude -> printfn "sude"
             | Liche -> printfn "liche"
```

Patterny také mohou nést data a mohou tedy sloužit jako extrakce dat z objektu:

```
let (|RGB|) (col : System.Drawing.Color) = RGB ( col.R, col.G, col.B )
let (|HSB|) (c : Color) = HSB (c.GetHue(), c.GetSaturation(), c.GetBrightness())
```

Každá data mohou být jiná, takže:

```
let (|Nil|Cons|) xs = match xs with | [] -> Nil xs | x::xs -> Cons (x, xs)
```

♣ Částečné patterny (právě jedna alternativa, ale nemusí nastat):

```
let (|Integer|_|) str = let success, value = System.Int32.TryParse str
                        if success then Some value else None
let (|Float|_|) str = let success, value = System.Double.TryParse str
                      if success then Some value else None
let parseNumber = function
    | Integer i -> printfn "%d : Integer" i
    | Float f -> printfn "%f : Floating point" f
    | _ -> printfn "%s : Not matched." str
```

Patterny také mohou samy brát hodnoty –– můžeme vytvořit parametrizované patterny:

```
let (|Nasobek|_|) k n = if n%k = 0 then Some (n/k) else None
match 21 with | Nasobek 4 n -> printfn "21 = 4 * %d" n
              | Nasobek 3 n -> printfn "21 = 3 * %d" n
```

Často používaný pattern na regulární výrazy. Vrací obsahy namatchovaných skupin z regulárního výrazu

```
let (|ParseRegex|_|) rgx s = match Regex(rgx).Match(s) with
                             | m when not m.Success -> None
                             | m -> Some(List.tail [for x in m.Groups->x.Value])
```

Pattern matchuje se rekurzivně, takže můžete použít víc vlastních patternů najednou, takže (pozor, magie:)

```
(* Načítáme data tří formátů, MM/DD/YY, MM/DD/YYYY, YYYY-MM-DD.*)
let parseDate = function
  | ParseRegex "(\d{1,2})/(\d{1,2})/(\d{1,2})$" [Integer m; Integer d; Integer y]
    -> new System.DateTime(y + 2000, m, d)
  | ParseRegex "(\d{1,2})/(\d{1,2})/(\d{3,4})" [Integer m; Integer d; Integer y]
    -> new System.DateTime(y, m, d)
  | ParseRegex "(\d{1,4})-(\d{1,2})-(\d{1,2})" [Integer y; Integer m; Integer d]
    -> new System.DateTime(y, m, d)
```