UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC35A - Linguagens de Programação

Prof. Dr. Rodrigo Hübner

Aula 07 - Linguagens funcionais: `ML` e `Haskell`; levantamento geral

ML

- Inicia o grupo de LPs que possuem sintaxe **semelhante as linguagens imperativas**
- Inclui tratamento de exceção, módulos para TADs e listas
- Editor *online*: https://sosml.org/
- Declaração de funções:

```
fun <nome_da_função>([parâmetros]) = expressão

fun square(x : int) = x * x
fun square(x : int) : int = x * x
fun square(x) = x * x
```

ML

- Controle de fluxo:

```
if <expressão> then <expressão_then> else <expressão_else>
```

- Permite múltiplas definições com casamento de padrões:

```
fun fact(0) = 1
|   fact(n : int): int = n * fact(n - 1);
```

`ML` = outras especificações

- Listas literais são especificadas com colchetes e elementos separados por vírgula:
  - `[3, 5, 7, 9]`
- Lista vazia pode ser `[]` ou `nill`
- Todos os elementos da lista precisam ser do mesmo tipo (**homogêneo**)
- Operação de "construção/inserção" de lista:
  - `3 :: [5, 7, 9]` resulta em `[3, 5, 7, 9]`

`ML` = outras especificações

- Operação unária para obter a **cabeça da lista**:

```
val lista = [1, 2, 3, 4]
val head = hd lista
```

- Operação unária para obter a **cauda da lista**:

```
val tail = tl lista
```

- `hd` e `tl` são menos em relação a `Lisp`, pois podem ser definidas com casamento padrões
- Ver `exemplos.ml`

`Haskell`

- Similar a `ML`: sintaxe parecida, escopo estático, **fortemente tipada**, e usa o **mesmo método de inferência de tipo**
- Compilador *online*: https://www.jdoodle.com/execute-haskell-online/
- Funções em `Haskell` podem ser **polimórficas**:

```
fact 0 = 1
fact n = n * fact (n - 1)

fib 0 = 1
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Haskell

- **Guardas** podem ser adicionadas em funções funcionando como expressões condicionais:

```
fact n
  | n == 0 = 1
  | n > 0  = n * fact (n - 1)

sub n
  | n < 10     = 0
  | n > 20     = 2
  | otherwise = 1
```

Haskell

- Operações em listas:

```
ghci> 5 : [2,7,9]
[5,2,7,9]

ghci> [1, 3..11]
[1,3,5,7,9,11]

ghci> [1,3,5] ++ [2,4,6]
[1,3,5,2,4,6]

produto []      = 1
produto (a : x) = a * produto x
```

Haskell

- *List Comprehensions*

```
[body | qualifiers]
ghci> [n * n * n | n <- [1..10]]
[1,8,27,64,125,216,343,512,729,1000]

ghci> let squares = [n * n | n <- [1..]] in take 10 squares
[1,4,9,16,25,36,49,64,81,100]
```

Haskell

- Avaliação atrasada (ou preguiçosa):

```
media a b | isInfinite b = b
          | otherwise    = (a + b) / 2

ghci> fibos = 0 : 1 : zipWith (+) fibos (tail fibos)

ghci> take 10 fibos
[0,1,1,2,3,5,8,13,21,34]
ghci> fibos !! 8
21
```

Mais avaliações...

- Top Functional Programming Languages in 2022: https://ericnormand.me/functional-programming-languages

Próxima aula

- Laboratório e enunciado de trabalho