UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BCC6002 - Aspectos de Linguagens de Programação

Prof. Dr. Rodrigo Hübner

Aula 06: Linguagens funcionais - Introdução; Cálculo lambda; Tipos de dados

Introdução

- O projeto das **linguagens imperativas** é baseado na arquitetura de von Neumann
- O projeto das **linguagens funcionais** é baseado em funções matemáticas
  - Sem preocupação direta com a arquitetura
  - Base teórica sólida (expressões matemáticas)

Funções matemáticas

- Funções simples:
  - **Definição**: `cubo(x) ≡ x * x * x`
  - **Aplicação**: `cubo(8)`
- Notação lambda:
  - **Definição**: `λ(x) x * x * x`
  - **Aplicação**: `(λ(x) x * x * x)(8)`

Funções matemáticas

- **Composição de funções**:
  - `h ≡ f ∘ g, h(x) ≡ f(g(x))`
- ***Apply to all***:
  - Toma uma função como parâmetro e aplica uma lista de valores
  - **Forma**: `α`
  - `h(x) ≡ x * x`
  - `α(h, (2, 3, 4))`
    - resulta `(4, 9, 16)`

Fundamentos de LPs funcionais

- Em uma **linguagem imperativa** é utilizado **variáveis** para uso posterior
- Em uma **linguagem funcional (LF)**, **variáveis não são necessárias**, assim como na matemática
- Na **LF**, a avaliação de uma **função sempre produz o mesmo resultado** se os mesmos parâmetros forem passados
- **Laço de repetição** é especificado com **recursão**
- Programas consistem em **definições** de funções e **especificação de aplicações** de funções

Fundamentos de LPs funcionais

- Uma **LF deve prover**:
  - Um conjunto de **funções primitivas**
  - Um conjunto de **formas funcionais**
  - Um **operador** de **aplicação de função**
  - Algumas **estruturas** para representar **dados**
- LPs imperativas possuem suporte limitado a LPs funcionais:
  - Formas funcionais (**retorno de função**)
  - Permite **efeitos colaterais**

LPs funcionais: `Lisp`

- `Lisp` (*LISt Processing*)
  - **Dialetos**: `Common Lisp`, `Scheme`, `Closure`
- Site *online* para testar códigos: https://rextester.com/l/common_lisp_online_compiler
- Tipos de dados:
  - **Átomos**: símbolos (identificadores)
  - **Listas**:
    - `(A B C D)`
    - `(A (B C) D (E (F G)))`

LPs funcionais: `Lisp`

- **Notação lambda**:
  - `(func_name(LAMBDA(arg1 ... argn) expression))`
- A **aplicação da funções** e as **listas de dados** tem a **mesma forma**
  - `(A B C)`

LPs funcionais: `Lisp`

- Formas funcionais (`lambda`, `print`, `format`):
  - Definição: `(lambda (a b) (+ a b))`
  - Aplicação: `((lambda (a b) (+ a b)) 4 5)`
  - Mostrar: `(print ((lambda (a b) (+ a b)) 4 5))`
  - Mostrar (formatado):

```
(format t "O resultado de 4 + 5 é ~a" (
  (lambda (a b) (+ a b)) 4 5))
```

LPs funcionais: `Scheme`

- Site *online* para testar códigos: https://www.tutorialspoint.com/execute_scheme_online.php
- Podemos utilizar o `DrRacket` (terminal iterativo)
- Parâmetros são avaliados: `(func params)`

```
(* (- 5 3) (/ 8 2))
(* 2 (/ 8 2))
(* 2 4)
8
```

LPs funcionais: Scheme

```
(define symbol expression)
(lambda (parameters) expression)
(define (function-name parameters) expression)
(if predicate then-expression else-expression)
(cond
    (predicate1 expression1)
    (predicate2 expression2)
    ...
    (predicateN expressionN)
    [(else expression)]
)
```

LPs funcionais: Scheme

- Formas especiais:

```
(let ((id exp)+) exp)
> (let ((a 10) (b 20)) (+ a b))
30
> (car '(a b c))
'a
> (car '((a b) c d))
'(a b)
> (cdr '((a b) c d))
'(c d)
> (cons 'a '(b c))
'(a b c)
```

LPs funcionais: `Scheme`

- **Recursão em cauda**
  - Chamada recursiva é a última operação da função

```
(define (fat-helper n partial)
    (if (= n 0)
    partial
    (fat-helper (- n 1) (* n partial))))

(define (fat n)
    (fat-helper n 1))
```

- **Recursão sem cauda**: ver `append.scm`

LPs funcionais: `Scheme`

- Outras formas funcionais:
  - `foldl`, `foldr`
    - `(foldl funcao acumulador lista)`
  - `map`
  - `filter`

Outras funções (aplícáveis diretamente):

- `eval`
- `apply`

Próxima aula

- Aumentando possibilidades com `ML` e `Haskell`
- Levantamento de outras LPs funcionais