Introdução ao Cálculo Lambda

Juliana Kaizer Vizzotto

Universidade Federal de Santa Maria

Disciplina de Teoria da Computação

Roteiro

.

- Introdução
- Sintaxe
- Semântica Operacional

Introdução

- Cálculo-lambda ou Cálculo-λ é um modelo de computação baseado na ideia que algoritmos podem ser vistos como *funções matemáticas mapeando inputs para outputs*.
- Foi introduzido por Alonzo Church da década de 30.
- Modelo genérico de programação equivalente a Máquina de Turing.
- Máquina de Turing $\equiv$ modelo para linguagens imperativas.
- Cálculo-λ $\equiv$ modelo para linguagens funcionais.

Introdução

- Em meados de 1960, Peter Landin observou que uma linguagem de programação complexa pode ser entendida através da formulação de um pequeno kernel de um cálculo que captura os mecanismos essenciais da linguagens.
- A linguagem kernel usada por Landin foi o *cálculo*-λ, no qual todas as computações são reduzidas a operações básicas de **definição de funções e aplicação**.
- Base para o desenvolvimento de Lisp, John MacCarthy.
- Além disso o cálculo-λ tem sido bastante utilizado na especificação de características de linguagens de programação, projeto e implementação de linguagens e no estudo de sistemas de tipos.

Sintaxe do Cálculo-lambda

- Church observou que quando denotamos uma função pela expressão: $x + y$, não é sempre claro o está função está realmente denotando!
- Por exemplo, a expressão $x + y$ pode ser interpretada como:
  1. O número $x + y$, onde x e y são alguns números fornecidos;
  2. A função $f : x \mapsto x + y$, que associa ao número x, o número $x + y$ paral algum y pre-definido.
  3. A função $g : y \mapsto x + y$, que associa ao número y, o número $x + y$ para algum valor de x pre-definido;
  4. Ou ainda, a função $h : x, y \mapsto x + y$, que recebe como argumentos x e y e retorna o valor $x + y$.

Sintaxe do Cálculo-lambda

- Devido a essa ambiguidade, Church propôs uma nova notação para funções que enfatiza na distinção entre variáveis usadas como argumentos e variáveis que representam valores pré-definidos.
- Nesta notação, uma função com um argumento x é precedido pelo símbolo λ.
- Por exemplo, a função $f : x \mapsto x + y$, que associa a uma entrada x o número $x + y$ para algum valor de y pré-determinado é escrita $\lambda x.x + y$.

Sintaxe do Cálculo-lambda

- As funções mencionadas acima podem ser facilmente diferenciadas no cálculo-λ:
  1. a função $f : x \mapsto x + y$ é escrita como $\lambda x.x + y$.
  2. a função $g : y \mapsto x + y$, é escrita como $\lambda y.x + y$.
  3. a função $h : x, y \mapsto x + y$, é escrita como $\lambda xy.x + y$.
- Funções anônimas!

Sintaxe do Cálculo-lambda

- A sintaxe de uma linguagem de programação é o conjunto de regras que define as combinações de símbolos que são são considerados programas corretamente *estruturados* em tal linguagem.
- Vamos definir a sintaxe do cálculo-λ usando BNF (*Backus Normal Form or Backus–Naur Form*).
- A sintaxe do cálculo-λ tem somente três tipos de termos:
  1. Uma variável, x, é um termo;
  2. A **abstração** de uma variável x de um termo t_1, escrita $\lambda x.t_1$ é um termo;
  3. A **aplicação** de um termo t_1 a um termo t_2 é também um termo.

Sintaxe do Cálculo-lambda

- Essas três maneiras de formar termos é resumida na seguinte gramática (BNF):

$$\begin{array}{lllr} t & ::= & & \text{termos} \\ & & x & \text{variavel} \\ & & \lambda x.t & \text{abstracao} \\ & & t\ t & \text{aplicacao} \end{array}$$

Sintaxe Abstrata $\times$ Sintaxe Concreta

- A sintaxe concreta de uma linguagem se refere a *strings* de caracteres que os programadores diretamente escrevem.
- A sintaxe abstrata é uma representação interna de programas como árevores com *labels* (também chamadas de árvores de sintaxe abstrata, *abstract syntax trees* ou AST).
- A representação como árvores fornece uma estrutura óbvia para os termos, facilitando sua manipulação em compiladores e interpretadores.

Sintaxe Abstrata × Sintaxe Concreta

- A transformação da sintaxe concreta em sintaxe abstrata acontece em dois estágios:
    1. Primeiro, o *analisador léxico* (ou *lexer*) converte a string de caracteres escrita pelo programador em uma sequência de *tokens* - identificadores, palavras-chave, constantes, pontuação, etc. O lexer remove comentários e trata de assuntos como espaços em branco, convenções sobre letras maiúsculas/minúsculas, e formatos para constantes numéricas e strings.
    2. Na sequência, o *parser* transforma essa sequência de *tokens* em árvores de sintaxe abstrata. Durante o parser várias convenções como *precedência* e *associatividade* reduzem a necessidade de programas com muitos parênteses.

Sintaxe Abstrata $\times$ Sintaxe Concreta

- Por exemplo, como ficaria a árvore de sintaxe abstrata para a seguinte expressão: $1 + 2 * 3$?
- Vamos focar na representação de termos sendo a sua sintaxe abstrata.
- Gramáticas como a apresentada acima devem ser vistas como representando estruturas de árvores, não strings de tokens de caracteres, i.e., sempre temos que ter em mente a árvore de sintaxe que o termo representa.
- Na escrita de termos λ, vamos adotar duas convenções na sua escrita em forma linear:
  1. Primeiro, aplicação associa à esquerda, i.e., $s\ t\ u$ é o mesmo que $(s\ t)\ u$.
  2. Segundo, o corpo de uma abstração estende o máximo possível à direita, e.g., $\lambda x.\lambda y.x\ y\ x$ representa a mesma árvore que $\lambda x.(\lambda y.((x\ y)\ x))$.

Escopo das Variáveis

- Uma ocorrência de uma variável x é dita *ligada* quando ela ocorre no corpo t de uma abstração $\lambda x.t$.
- Uma ocorrência de x é *livre* se ela aparece em uma posição onde ela não está ligada por uma abstração em x.
- Por exemplo, as ocorrências de x em $x\ y$ e $\lambda y.x\ y$ são livres.
- Já as ocorrências de x em $\lambda x.x$ e $\lambda z.\lambda x.\lambda y.x\ (y\ z)$ são ligadas.
- Em $(\lambda x.x)\ x$ a primeira ocorrência de x é ligada e a segunda é livre.

Variáveis livres: definição formal

- O conjunto de variáveis livres de um termo t, escrito $FV(t)$, é definido como segue:

$$\begin{array}{lcl} FV(x) & = & \{x\} \\ FV(\lambda x.t_1) & = & FV(t_1) - \{x\} \\ FV(t_1\ t_2) & = & FV(t_1) \cup FV(t_2) \end{array}$$

Semântica Operacional

- Na sua forma pura, o cálculo-λ não possui constantes nem operadores pré-definidos - não possui números, nem operações aritméticas, nem condicionais, nem *arrays*, nem *loops*, I/O, etc.
- A única coisa que temos para "computar" é a aplicação de funções em argumentos (os quais podem ser funções também).
- Cada passo na computação consiste de reescrever uma aplicação cujo componente esquerdo é uma abstração, substituindo a variável ligada no corpo da abstralção pelo componente do lado direito:

$$(\lambda x.t_{12})t_2 \to [x \mapsto t_2]t_{12},$$

- Onde $[x \mapsto t_2]t_{12}$ significa "o termo obtido pela **substituição** de todas as ocorrências livres de x em t_{12} por t_2."

Semântica Operacional (Beta-reduction)

- Por exemplo, o termo $(\lambda x.x)y$ avalia para y.
- O termo $(\lambda x.x(\lambda x.x))(u\ r)$ avalia para $u\ r(\lambda x.x)$.
- De acordo com Church, um termo da forma $(\lambda x.t_{12})t_2$ é chamado **redex** (ou "expressão redutível").
- A operação de reescrever um redex de acordo com a regra acima é chamada redução-beta (beta-reduction).

Estratégias de Avaliação

- Existem várias estratégias de avaliação para o cálculo-λ.
- Cada estratégia define qual redex ou redexes em um termo podem ser reduzidos no próximo passo de avaliação:
  - *Full beta-reduction*:
  - *Normal order*:
  - *Call-by-need*:
  - *Call-by-value*: