Trabalho 1 - Untyped Lambda Calculus

Linguagens Declarativas - Disciplina do PPGI - UnB
Thiago F. Figueiredo - 120023199

September 27, 2017

1 Alpha-Equivalência

Essa nomenclatura se origina na análise semântica de um termo. Para dois termos serem alpha-equivalentes, eles precisam ter o mesmo significado semântico, independente do nome que se dê para as variáveis da expressão. Se tomarmos como exemplo a função identidade do cálculo lambda, vemos que \x.x tem o mesmo significado que \y.y. Eis o termo alpha-equivalência, uma propriedade entre dois termos que possuem o mesmo significado em sua evaluação.

1.1 Motivação para a conversão alpha

Nos processos de substituição nos termos do cálculo lambda, pode haver um choque nos nomes da variável que se quer substituir, pelo termo de sbustituição em si e o corpo da expressão. As variáveis ligadas que estão no corpo da expressão não podem sofrer substituição, correndo o risco de perderem o seu significado semântico caso sejam. Para garantir que isso não aconteça, alguma estratégia deve ser tomada durante o processo de substituição. O livro da nossa disciplina apresenta cinco possíveis possibilidades, porém a escolha neste trabalho e a do livro são as de representar os termos lambda canonicamente, com os termos DeBruijn. Dessa forma, a renomeação durante a substituição não é necessária, pois os termos não possuem nome próprio.

1.2 Implementação - Termos DeBruijn

Tanto no livro quanto neste trabalho, a estratégia escolhida para o *"Nameless representation of terms"* foi o do cientista Nicolas De Bruijn. Sua ideia era a representação dos termos da forma mais direta possível, onde as ocorrências de variáveis apontam diretamente para o seu lambda binder. Isso foi feito representando as variáveis por números inteiros ao invés de caractéres, onde um número n significaria o n-ésimo lambda binder.

A conversão de termos lambda para termos DeBruijn foi feita de maneira relativamente direta, através da função *makeDeBruijn :: Term -¿ BTerm*. Nessa função, foi utilizado uma lista auxil-

iar, comumente chamada de environment, ou ambiente, onde guardamos todas as ocorrências de variáveis durante a conversão. Temos três casos distintos de conversão:

1. Variável: no caso de uma variável, verificamos se ela está presente na lista environment; se estiver, pegamos o seu índice e esse passa a ser a representação da variável no índice de DeBruijn.
2. Abstração: no caso da abstração lambda, adicionamos sua variável à lista environment, e continuamos com a conversão recursivamente
3. Aplicação: a conversão é bem direta, aplicando a conversão nos dois termos da aplicação

Sendo assim, sempre que formos converter uma variável ao índice de DeBruijn, ela será a representação correta do n-ésimo lambda binder daquela variável.

Mais explicações da implementação se encontram em comentários no código.