

# Inferência de Tipos para CPS

Vinícios Bidin Santos

Universidade do Estado de Santa Catarina

[vinibidin@gmail.com](mailto:vinibidin@gmail.com)

Orientador: Dr. Cristiano Damiani Vasconcellos

Coorientador: Me. Paulo Henrique Torrens

24/11/2024

- 1 Introdução
  - Objetivos
- 2 Representação Intermediária de Código
  - Estilo de Passagem de Continuação (CPS)
- 3 Teoria de Tipos
- 4 Sistema Damas-Milner
  - Algoritmo W
- 5 Proposta
- 6 Referências

## Compilação:

- Tradução de código de uma linguagem para outra
  - Geralmente do código-fonte para o de máquina
- Composta por diferentes etapas como:
  - Análise léxica
  - Análise sintática
  - Análise semântica
  - Otimizações
  - Geração de código
- Ligadas por Representações Intermediárias
  - Principalmente nas otimizações (PLOTKIN, 1975)

## Representações Intermediárias:

- Linguagens imperativas
  - Atribuição Única Estática (SSA)
- Linguagens funcionais
  - Forma Normal Administrativa (ANF)
  - Estilo de Passagem de Continuação (CPS)
- CPS
  - Continuações explícitas
    - Parâmetro extra na função
    - Funções sem retorno
  - Otimizações
    - Eliminação da pilha de chamadas
    - Eliminação de chamadas de cauda

- Formalizar um sistema de tipos para CPS
- Propor e implementar em Haskell um algoritmo de inferência de tipos para CPS
- Validar a implementação do algoritmo por meio do teste de inferência para expressões

# Representação Intermediária de Código

- Estrutura de dados usada para manter integridade semântica e possibilitar otimizações (COOPER; TORCZON, 2014)
  - Classificadas de acordo com o nível de abstração
  - Muitas vezes aplicadas em sequência

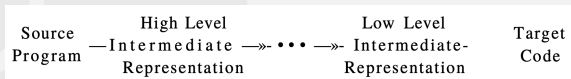


Figura: Sequência de representações intermediárias

Fonte: (AHO et al., 2008)

- Fluxo de controle
  - Ordem das instruções
  - Escopo

# Estilo de Passagem de Continuação (CPS)

- Técnica de transformação de código que torna o fluxo de controle explícito
  - Chamadas de função passam o controle para a próxima etapa explicitamente, conhecida como continuação (APPEL, 1992)
  - Ao invés das funções retornarem o resultado da computação, é invocado uma continuação, representando o próximo passo
- Toda chamada de função passa então a ser uma chamada de cauda (do inglês *tail-call*)

## Chamada de cauda:

- Última instrução executada em uma função é uma chamada a outra função, sem que restem computações adicionais a serem feitas após essa chamada (MUCHNICK, 1997)
  - Função atual pode liberar seu quadro de ativação

## Chamada não de cauda:

- Ainda restam operações, como somas ou multiplicações, após a chamada da função
  - Função atual precisa manter seu quadro de ativação até que as operações sejam concluídas



# Estilo de Passagem de Continuação (CPS)

Figura: Função fatorial em Haskell com chamada não de cauda

```
1 factorial :: Int -> Int
2 factorial 0 = 1
3 factorial n = n * factorial (n - 1)
```

Fonte: o autor

Figura: Função fatorial em Haskell com chamada de cauda

```
1 go :: Int -> Int -> Int
2 go 1 a = a
3 go n a = go (n - 1) (a * n)
4
5 factorial :: Int -> Int
6 factorial 0 = 1
7 factorial n = go n 1
```

Fonte: o autor


















 AHO, A. V. et al. *Compiladores: Princípios, técnicas e ferramentas*. 2th. ed. São Paulo, SP, Brasil: Pearson Education, 2008.

 APPEL, A. W. *Compiling with continuations*. USA: Cambridge University Press, 1992. ISBN 0521416957.

 COOPER, K. D.; TORCZON, L. *Contruindo Compiladores*. 2th. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2014.

 MUCHNICK, S. S. *Advanced Compiler Design and Implementation*. Oxford, England: Morgan Kaufmann, 1997.

 PLOTKIN, G. Call-by-name, call-by-value and the  $\lambda$ -calculus. *Theoretical Computer Science*, v. 1, n. 2, p. 125–159, 1975. ISSN 0304-3975. Disponível em: <<https://www.sciencedirect.com/science/article/pii/0304397575900171>>.