Inferência de Tipos para CPS

Vinícios Bidin Santos
Universidade do Estado de Santa Catarina
vinibidin@gmail.com

Orientador: Dr. Cristiano Damiani Vasconcellos
Coorientador: Me. Paulo Henrique Torrens

28/11/2024

Sumário

Introdução

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

Objetivos

- 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

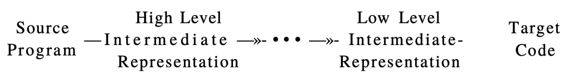

Figura: Sequência de representações intermediárias

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 (*tail-call*)

Estilo de Passagem de Continuação (CPS)

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

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

Fonte: o autor

Figura: Função fatorial em Haskell com chamada de cauda

```haskell
go :: Int -> Int -> Int
go 1 a = a
go n a = go (n - 1) (a * n)

factorial :: Int -> Int
factorial 0 = 1
factorial n = go n 1
```

Fonte: o autor

Cálculo Lambda:
Church (1932) define o cálculo-λ, que é representado pela seguinte gramática:

$$e ::= x \mid \lambda x.e \mid ee$$

- **Variável:** identificadores no sistema
- **Abstração:** função que associa um identificador x a um termo e
- **Aplicação:** aplicação de um termo a outro

Estilo de Passagem de Continuação (CPS)

Variáveis no cálculo-λ podem ser:

- **Livres:** quando não estão associadas a uma abstração de função
  - $\lambda x.y$
- **Ligadas:** quando estão associadas a uma abstração de função
  - $(\lambda x.x)y$

Para analisar expressões:

- α-**redução:** Renomeação de variáveis ligadas.

$$\lambda x.e[x] \to \lambda y.e[y]$$

- β-**redução:** Aplicação de função.

$$(\lambda x.e_1)e_2 \to e_1[e_2/x]$$

- η-**redução:** Expansão de função.

$$\lambda x.(e\,x) \to e \quad \text{se } x \text{ não ocorre livre em } e$$

Estilo de Passagem de Continuação (CPS)

Transformação CPS:

No cálculo-λ tradicional, o fluxo de execução é implícito

- Funções são aplicadas e os resultados são retornados
- $\lambda x.x + 1$

Já no CPS, o fluxo de execução é explícito

- Uma série de chamadas de funções passam o resultado para um argumento extra, a continuação, indicando o próximo passo da computação
- $\lambda x.\lambda k.k(x + 1)$

Estilo de Passagem de Continuação (CPS)

Cálculo de Continuações (*CPS-calculus*):
(THIELECKE, 1997) define como sendo um sistema formal que trata o CPS como um modelo computacional por si só
Seus termos são:

$$M ::= x\langle\vec{x}\rangle \mid M\{x\langle\vec{x}\rangle = M\}$$

- **Salto (*jump*):** uma chamada para a continuação x com os parâmetros $\vec{x}$
- **Vínculo (*binding*):** uma chamada onde o corpo M está vinculado à continuação x com os parâmetros $\vec{x}$

Estilo de Passagem de Continuação (CPS)

Tradução CPS:
Converte um código escrito em estilo direto para CPS (FLANAGAN et al., 1993)

- Modificar as funções para elas não retornarem um valor, mas sim, passarem o resultado para uma continuação

Estilo de Passagem de Continuação (CPS)

Figura: Função soma em Haskell em Estilo Direto

```haskell
add :: Int -> Int -> Int
add x y = x + y
```

Fonte: o autor

Figura: Função soma em Haskell em CPS

```haskell
addCps :: Int -> Int -> (Int -> r) -> r
addCps x y k = k (x + y)
```

Fonte: o autor

Estilo de Passagem de Continuação (CPS)

Figura: Função fatorial em Haskell em Estilo Direto

```haskell
fat :: Int -> Int
fat 0 = 1
fat n = n * fat (n - 1)
```

Fonte: o autor

Figura: Função fatorial em Haskell em CPS

```haskell
fatCps :: Int -> (Int -> r) -> r
fatCps 0 k = k 1
fatCps n k =
  fatCps
    (n - 1)
    (\x -> seq x k (n * x))
```

Fonte: o autor

Russel em 1908 apresentou uma contradição na Teoria de Conjuntos

Paradoxo de Russell:

$$\text{Seja } R = \{x \mid x \notin x\}, \text{ então } R \in R \iff R \notin R$$

Outra maneira de descrever este paradoxo é com o paradoxo do barbeiro:

Imagine uma cidade com apenas um barbeiro, onde ele somente barbeia aqueles que não se barbeiam

Aplicações:

- Formalização de sistemas de tipos para linguagens de programação
- Construção de assistentes de provas
  - Coq utiliza Cálculo de Construções (COQUAND; HUET, 1988)
- Linguagens como Idris e Agda, também permitem a verificação de provas formais

Teoria de Tipos

Em linguagens de programação (PIERCE, 2002):

- Tipos simples
  - Tipo fixo a um termo
  - $Int \rightarrow Int$
- Tipos polimórficos
  - Generalidade
  - $a \rightarrow a$
- Tipos dependentes
  - Tipos dependem de valores
  - $Vector(n) \rightarrow Vector(m) \rightarrow Vector(n+m)$

Pierce (2002) define duas variedades de polimorfismo:

- Paramétrico
  - Única definição de função genérica
  - Função identidade
- Com sobrecarga
  - Múltiplas implementações de uma função
  - Sobrecarga de operadores

Cálculo Lambda Simplesmente Tipado:

- Variante do Cálculo-λ que incorpora tipos (CHURCH, 1940)
- Cada função recebe e retorna valores de tipos específicos

Sua sintaxe básica inclui:

- Variáveis: $x, y, z, \ldots$
- Tipos: $T ::= \textbf{Int} \mid \textbf{Bool} \mid T \to T$
- Termos: $\lambda x : T.\tau \mid \tau_1 \tau_2 \mid x$

Regra de tipagem para abstrações lambda:

$$\frac{\Gamma, x : T_1 \vdash \tau : T_2}{\Gamma \vdash (\lambda x : T_1.\tau) : T_1 \to T_2}$$

Correspondência Curry-Howard:

- Correspondência entre proposições intuicionistas lógicas e tipos
- Correspondência entre provas e programas
- O tipo $A \to B$ neste cálculo pode ser visto como a implicação lógica: se A, então B
- Método sistemático para raciocinar sobre sistemas de inferência de tipos
  - Linguagem ML por Damas e Milner (1982), com o algoritmo W
  - Linguagem Haskell com a extensão do sistema Damas-Milner

Sistema Damas-Milner

Sistema de tipos robusto para linguagens funcionais (MILNER, 1978; DAMAS; MILNER, 1982)

- Inferência automática de tipos polimórficos
- Cálculo-λ com polomorfismo paramétrico introduzido via `let`

Sua sintaxe define as expressões e os tipos usados no processo de inferência:

Variáveis	x
Expressões	$e ::= x \mid e\ e' \mid \lambda x.e \mid \texttt{let}\ x = e\ \texttt{in}\ e'$
Variáveis de tipo	α
Tipos primitivos	ι
Tipos	$\tau ::= \alpha \mid \iota \mid \tau \to \tau\alpha$
Schemes	$\sigma ::= \forall\alpha.\sigma \mid \tau$

- Generalização
  - Tipo mais geral
- Instanciação
  - Tipo concreto
- `let id` $= \lambda x.x$ `in` (`id` 1, `id 'a'`)
  - $\forall\alpha.\alpha \to \alpha$

- Substituição de tipos
  - Mapeamento finito de variáveis de tipos para tipos, denotado por S
  - $[\alpha_1 \mapsto \tau_1, \alpha_2 \mapsto \tau_2, \ldots, \alpha_n \mapsto \tau_n]$
  - S associa cada variável de tipo α_i a um tipo τ_i específico
  - $S\tau$ substitui todas as ocorrências livres de α_i em τ por τ_i

Figura: Regras de Inferência do sistema Damas-Milner

$$\text{TAUT:} \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$\text{ABS:} \quad \frac{\Gamma, x : \tau \vdash e : \tau'}{\Gamma \vdash (\lambda x.e) : \tau \to \tau'}$$

$$\text{APP:} \quad \frac{\Gamma \vdash e : \tau' \to \tau \quad \Gamma \vdash e' : \tau'}{\Gamma \vdash (e\ e') : \tau}$$

Figura: Regras de Inferência do sistema Damas-Milner (continuação)

$$\text{LET:} \quad \frac{\Gamma \vdash e : \sigma \quad \Gamma, x : \sigma \vdash e' : \tau}{\Gamma \vdash (\texttt{let } x = e \texttt{ in } e') : \tau}$$

$$\text{INST:} \quad \frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \sigma'} \quad (\sigma > \sigma')$$

$$\text{GEN:} \quad \frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e : \forall \alpha \sigma} \quad (\alpha \text{ não livre em } \Gamma)$$

Fonte: o autor. Adaptado de (DAMAS; MILNER, 1982)

Algoritmo W

Algoritmo de inferência introduzido por Damas e Milner (1982):

- Para linguagens funcionais
- Algoritmo eficiente
  - Na maioria dos casos
- Unificação
  - Solucionar equações de tipos
  - Dados dois tipos τ_1 e τ_2
  - A unificação procura uma substituição S tal que $S\tau_1 = S\tau_2$
  - Atribui os tipos mais gerais possíveis

Proposta

Sistema de tipos monomórfico para CPS com um único construtor (chamado de negação poliádica) para representar continuações (TORRENS; ORCHARD; VASCONCELLOS, 2024):

$$\textit{Types} \;\; \tau \;\; ::= \;\; \neg\vec{\tau} \;\mid\; X$$

$$\textit{Environments} \;\; \Gamma ::= \cdot \mid \Gamma, x : \tau$$

$$\frac{\Gamma(k) = \neg\vec{\tau} \qquad \Gamma(\vec{x}) = \vec{\tau}}{\Gamma \vdash k\langle\vec{x}\rangle} \; (J)$$

$$\frac{\Gamma, k : \neg\vec{\tau} \vdash b \qquad \Gamma, \vec{x} : \vec{\tau} \vdash c}{\Gamma \vdash b\{k\langle\vec{x}\rangle = c\}} \; (B)$$

- Propor uma extensão ao sistema de tipos
  - Suporte a tipos polimórficos
  - Algoritmo de inferência de tipos

Conclusões Parciais

- CPS é uma escolha interessante para IR
  - Otimizações
- Sistema de tipos
  - Correção de transformações e otimizações
  - Ausência de certos comportamentos indesejados

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.

CHURCH, A. A set of postulates for the foundation of logic. *Annals of mathematics*, JSTOR, p. 346–366, 1932.

CHURCH, A. A formulation of the simple theory of types. *J. Symb. Log.*, Cambridge University Press (CUP), v. 5, n. 2, p. 56–68, jun. 1940.

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

Referências

COQUAND, T.; HUET, G. The calculus of constructions. *Information and Computation*, v. 76, n. 2, p. 95–120, 1988. ISSN 0890-5401. Disponível em: <https://www.sciencedirect.com/science/article/pii/0890540188900053>.

DAMAS, L.; MILNER, R. *Principal type-schemes for functional programs*. Tese (Doutorado) — University of Edinburgh, Scotland, 1982.

FLANAGAN, C. et al. The essence of compiling with continuations. *SIGPLAN Not.*, Association for Computing Machinery, New York, NY, USA, v. 28, n. 6, p. 237–247, jun 1993. ISSN 0362-1340. Disponível em: <https://doi.org/10.1145/173262.155113>.

MILNER, R. A theory of type polymorphism in programming. In: *Journal of Computer and System Sciences*. [S.l.]: Elsevier, 1978. v. 17, n. 3, p. 348–375.

Referências

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

PIERCE, B. C. *Types and Programming Languages*. 1st. ed. [S.l.]: The MIT Press, 2002. ISBN 0262162091.

PLOTKIN, G. Call-by-name, call-by-value and the λ-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>.

THIELECKE, H. *Categorical Structure of Continuation Passing Style*. [S.l.]: University of Edinburgh. College of Science and Engineering. School of Informatics., 1997.

TORRENS, P.; ORCHARD, D.; VASCONCELLOS, C. On the operational theory of the cps-calculus: Towards a theoretical foundation for irs. *Proc. ACM Program. Lang.*, Association for Computing Machinery, New York, NY, USA, v. 8, n. ICFP, aug 2024. Disponível em: <https://doi.org/10.1145/3674630>.