

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC**  
**CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT**  
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO – BCC**

**VINÍCIOS BIDIN SANTOS**

**INFERÊNCIA DE TIPOS PARA CPS**

**JOINVILLE**

**2024**

**VINÍCIOS BIDIN SANTOS**

**INFERÊNCIA DE TIPOS PARA CPS**

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Cristiano Damiani Vasconcellos

Coorientador: Paulo Henrique Torrens

**JOINVILLE**

**2024**

Para gerar a ficha catalográfica de teses e  
dissertações acessar o link:  
<https://www.udesc.br/bu/manuais/ficha>

Santos, Vinícios Bidin  
Inferência de tipos para CPS / Vinícios Bidin Santos.  
- Joinville, 2024.  
22 p. : il. ; 30 cm.

Orientador: Cristiano Damiani Vasconcellos.  
Coorientador: Paulo Henrique Torrens.  
Trabalho de Conclusão de Curso - Universidade  
do Estado de Santa Catarina, Centro de Ciências  
Tecnológicas, Bacharelado em Ciência da Computação,  
Joinville, 2024.

1. Inferência de Tipos. 2. Estilo de Passagem de  
Continuação. 3. Algoritmo W. 4. Damas-Milner. 5. Haskell.  
I. Vasconcellos, Cristiano Damiani . II. Torrens, Paulo  
Henrique . III. Universidade do Estado de Santa Catarina,  
Centro de Ciências Tecnológicas, Bacharelado em Ciência  
da Computação. IV. Título.

**VINÍCIOS BIDIN SANTOS**

**INFERÊNCIA DE TIPOS PARA CPS**

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Cristiano Damiani Vasconcellos

Coorientador: Paulo Henrique Torrens

**BANCA EXAMINADORA:**

Orientador:

---

Dr. Cristiano Damiani Vasconcellos  
UDESC

Coorientador:

---

Me. Paulo Henrique Torrens  
University of Kent

Membros:

---

Dra. Karina Girardi Roggia  
UDESC

---

Me. Gabriela Moreira

Joinville, Novembro de 2024

Um salve pra geral que me apoiou nesses anos!

*“Eu sou a senhora Marocas.”* (Senhora Marocas  
– Toy Story, [1995])

## RESUMO

Este trabalho propõe uma investigação sobre a inferência de tipos para o Estilo de Passagem de Continuação (CPS) - representação intermediária amplamente utilizada em compiladores de linguagens funcionais. A pesquisa se concentra na extensão do algoritmo W, tradicionalmente usado para inferência de tipos no sistema Damas-Milner, para abranger o cálculo de continuções. A proposta inclui a implementação dessa extensão na linguagem Haskell e a validação do algoritmo por meio de programas de teste, assegurando que os tipos inferidos estejam corretos.

**Palavras-chave:** Inferência de Tipos, Estilo de Passagem de Continuação (CPS), Algoritmo W, Damas-Milner, Haskell, Sistema de Tipos.

## **ABSTRACT**

This work proposes an investigation into type inference for Continuation Passing Style (CPS) - an intermediate representation widely used in compilers for functional languages. The research focuses on extending the W algorithm, traditionally used for type inference in the Damas-Milner system, to encompass continuation calculus. The proposal includes implementing this extension in the Haskell programming language and validating the algorithm through test programs, ensuring that the inferred types are correct.

**Keywords:** Type Inference, Continuation Passing Style (CPS), W Algorithm, Damas-Milner, Haskell, Type System.



## **LISTA DE ILUSTRAÇÕES**

Figura 1 – Sequência de representações intermediárias . . . . .	15
---	----

## **LISTA DE TABELAS**

## **LISTA DE ABREVIATURAS E SIGLAS**

IR	Intermediate Representation
CPS	Continuation Passing Style

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>12</b>
1.1	OBJETIVO GERAL . . . . .	13
1.2	OBJETIVOS ESPECÍFICOS . . . . .	13
1.3	METODOLOGIA . . . . .	13
1.4	ESTRUTURA DO TRABALHO . . . . .	13
<b>2</b>	<b>REPRESENTAÇÃO INTERMEDIÁRIA DE CÓDIGO . . . . .</b>	<b>15</b>
2.1	CPS . . . . .	16
<b>3</b>	<b>TEORIA DE TIPOS . . . . .</b>	<b>17</b>
<b>4</b>	<b>SISTEMA DAMAS-MILNER . . . . .</b>	<b>18</b>
4.1	ALGORITMO W . . . . .	18
<b>5</b>	<b>FORMALIZAÇÃO . . . . .</b>	<b>19</b>
<b>6</b>	<b>IMPLEMENTAÇÃO . . . . .</b>	<b>20</b>
<b>7</b>	<b>CONCLUSÃO . . . . .</b>	<b>21</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>22</b>

## 1 INTRODUÇÃO

A compilação de programas envolve diversas fases, cada uma com funções específicas, como análise léxica, análise sintática, análise semântica, otimizações, e, finalmente, a geração de código. Uma etapa crítica nesse processo é a otimização, que frequentemente se baseia em representações intermediárias (IRs). Essas representações atuam como ponte entre o código fonte e o código de máquina, permitindo que transformações e otimizações sejam aplicadas de maneira mais eficaz (PLOTKIN, 1975).

As representações intermediárias variam conforme o paradigma da linguagem de programação. Para linguagens imperativas, a Representação em Atribuição Única Estática (SSA) é amplamente adotada. Já em linguagens funcionais, a Forma Normal Administrativa (ANF) e o Estilo de Passagem de Continuação (CPS) se destacam. Este trabalho foca especificamente no CPS, uma IR que oferece vantagens particulares em termos de otimização e simplicidade na geração de código.

Essas características do CPS se tornam ainda mais evidentes quando comparamos como diferentes linguagens lidam com o fluxo de execução. Em linguagens de alto nível, por exemplo, a pilha de chamadas atua como uma abstração fundamental para gerenciar o controle de retorno das funções. No entanto, em linguagens de baixo nível, como *assembly*, não há tal abstração, exigindo que o controle do fluxo seja manualmente tratado por meio de endereços de retorno.

Nesse contexto, o CPS se destaca ao tornar as continuações explícitas no código. Ao invés de confiar na pilha de chamadas para gerenciar retornos, o CPS introduz um parâmetro adicional em cada função, representando a continuação — isto é, o que deve ser feito com o resultado da função (KENNEDY, 2007). Desta forma, em vez de simplesmente retornar um valor diretamente, a função invoca essa continuação, transferindo explicitamente o controle à próxima etapa da computação. Isso elimina a dependência da pilha de chamadas, simplificando o modelo de execução e tornando-o mais alinhado com as necessidades de linguagens de baixo nível.

Além disso, a adoção do CPS como representação intermediária vai além da tradução de linguagens de alto nível para código de máquina. O CPS facilita a aplicação de otimizações avançadas, como a eliminação de chamadas de cauda e a fusão de funções, além de permitir uma correspondência mais direta com o código gerado em linguagens de montagem (FLANAGAN et al., 1993).

Por outro lado, um ponto importante a ser considerado é que muitas implementações do CPS optam por uma representação não tipada (MORRISETT et al., 1999). Embora essa abordagem simplifique a implementação inicial, ele pode comprometer a segurança e a correção do código. Um sistema de tipos robusto pode não apenas garantir a correção de certas transformações e otimizações, mas também identificar uma classe inteira de erros antes da execução, proporcionando assim maior confiabilidade ao processo de compilação.

Diante dessas considerações, a linguagem de programação Haskell foi escolhida para

a implementação deste trabalho. Haskell, sendo uma linguagem funcional pura fortemente tipada, possui características desejáveis, como transparência referencial (SØNDERGAARD; SESTOFT, 1990) e um sistema de tipos robusto para explorar as vantagens do CPS e aplicar o sistema de tipos de maneira rigorosa. Dessa forma, a escolha de Haskell não apenas facilita o desenvolvimento de uma implementação segura e eficiente do CPS, como também conta com garantias de seguranças que são fundamentais para o sucesso deste trabalho.

## 1.1 OBJETIVO GERAL

Este trabalho visa explorar a inferência de tipos para CPS, propondo uma extensão do algoritmo W - um algoritmo de inferência de tipos para o sistema Damas-Milner - para o cálculo de continuções, definido em (TORRENS; ORCHARD; VASCONCELLOS, 2024) na linguagem de programação Haskell.

## 1.2 OBJETIVOS ESPECÍFICOS

- Formular uma extensão do algoritmo W para CPS;
- Implementar essa extensão em Haskell;
- Validar a implementação do algoritmo por meio do teste de inferência para expressões. Se possível, realizar a geração de programas para verificação de que o algoritmo infere corretamente os tipos a eles.

## 1.3 METODOLOGIA

A metodologia deste trabalho consistirá em duas principais etapas: pesquisa bibliográfica e implementação. A primeira etapa envolve uma extensa revisão de literatura sobre continuções e seu cálculo, bem como um aprofundamento do sistema Damas-Milner, com o objetivo de proporcionar uma compreensão completa ao autor. A segunda etapa trata da formulação do algoritmo W estendido para o cálculo de continuções, junto com sua implementação.

Para validar a implementação, será testado a inferência do algoritmo com testes. Caso seja possível, será feito ainda a geração de programas de teste, nos quais a inferência de tipos será verificada. A validação consistirá em comparar os tipos inferidos pelo algoritmo com os tipos esperados para esses programas.

## 1.4 ESTRUTURA DO TRABALHO

Este trabalho está estruturado de maneira que será fornecida, antes da formalização da extensão e implementação do algoritmo W, nos capítulos 5 e 6, uma fundamentação teórica. Inicialmente, será abordada a representação intermediária no capítulo 2, com um aprofundamento

em CPS na seção 2.1. Na sequência, o capítulo 3 discutirá a teoria de tipos, apresentando os conceitos e definições necessários para a compreensão do tema. O sistema Damas-Hindley-Milner será detalhado no capítulo 4, incluindo o algoritmo W, discutido de forma mais específica na seção 4.1. Por fim, o capítulo 7 trará as considerações finais, sintetizando os principais pontos e contribuições do trabalho.

## 2 REPRESENTAÇÃO INTERMEDIÁRIA DE CÓDIGO

Na computação, um compilador é um programa responsável por traduzir um código escrito em uma linguagem de programação para outra, geralmente do código-fonte para código de máquina, permitindo assim a execução do programa. Durante esse processo, é fundamental que o mínimo de informações seja perdido, uma vez que a semântica original deve ser preservada no processo de tradução. Uma comum escolha de estrutura utilizada para manter a integridade semântica e possibilitar otimizações, são as representações intermediárias (IR, do inglês *intermediate representation*) (COOPER; TORCZON, 2014).

Compiladores modernos, amplamente utilizados na indústria, empregam mais de uma IR para tirar proveito das vantagens de cada uma, uma vez que essas representações possuem diferentes especialidades. As IRs podem ser classificadas de acordo com o nível de abstração e são comumente aplicados em sequência. Representações com um nível maior de abstração são usadas próximas ao código-fonte, enquanto aquelas de nível mais baixo estão mais próximas do código de máquina (AHO et al., 2008), como ilustrado na figura 1.

Uma das principais informações que deve ser preservada em uma IR, é o fluxo de controle, isto é, a ordem em que as instruções do programa são executadas, como chamadas de função, *loops* e condições. Para garantir que o compilador mantenha a semântica correta do programa, o fluxo de controle deve ser repassado de alguma maneira durante o processo de tradução (COOPER; TORCZON, 2014). Uma das maneiras disso ser feito explicitamente, é com o uso de continuções, que são funções que descrevem o próximo passo de uma computação em um ponto particular da execução do programa.

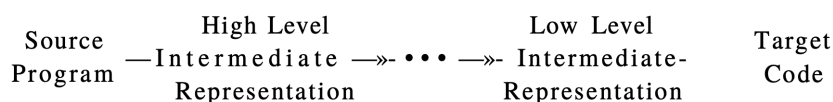


Figura 1 – Sequência de representações intermediárias

Fonte: (AHO et al., 2008)



## 2.1 CPS

### **3 TEORIA DE TIPOS**

## **4 SISTEMA DAMAS-MILNER**

### **4.1 ALGORITMO W**

## **5 FORMALIZAÇÃO**

## **6 IMPLEMENTAÇÃO**

## **7 CONCLUSÃO**

## REFERÊNCIAS

- AHO, Alfred V et al. **Compiladores: Princípios, técnicas e ferramentas**. 2th. ed. São Paulo, SP, Brasil: Pearson Education, 2008. Citado na página 15.
- COOPER, Keith D; TORCZON, Linda. **Contruindo Compiladores**. 2th. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2014. Citado na página 15.
- FLANAGAN, Cormac 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>>. Citado na página 12.
- KENNEDY, Andrew. Compiling with continuations, continued. In: **Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming**. New York, NY, USA: Association for Computing Machinery, 2007. (ICFP '07), p. 177–190. ISBN 9781595938152. Disponível em: <<https://doi.org/10.1145/1291151.1291179>>. Citado na página 12.
- MORRISETT, Greg et al. From system f to typed assembly language. **ACM Trans. Program. Lang. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 21, n. 3, p. 527–568, may 1999. ISSN 0164-0925. Disponível em: <<https://doi.org/10.1145/319301.319345>>. Citado na página 12.
- PLOTKIN, G.D. 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>>. Citado na página 12.
- SØNDERGAARD, Harald; SESTOFT, Peter. Referential transparency, definiteness and unfoldability. **Acta Inform.**, Springer Nature, v. 27, n. 6, maio 1990. Citado na página 13.
- TORRENS, Paulo; ORCHARD, Dominic; VASCONCELLOS, Cristiano. 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>>. Citado na página 13.