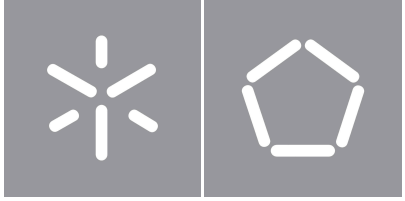
Universidade do Minho
Escola de Engenharia

André Bernardo Coelho Nunes

Regras de Scope em Ztrategic

janeiro 2024



Universidade do Minho
Escola de Engenharia

André Bernardo Coelho Nunes

Regras de Scope em Ztrategic

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de
João Alexandre Batista Vieira Saraiva
José Nuno Castro de Macedo janeiro 2024

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:

[Caso o autor pretenda usar uma das licenças Creative Commons, deve escolher e deixar apenas um dos seguintes ícones e respetivo lettering e URL, eliminando o texto em itálico que se lhe segue. Contudo, é possível optar por outro tipo de licença, devendo, nesse caso, ser incluída a informação necessária adaptando devidamente esta minuta]

CC BY

https://creativecommons.org/licenses/by/4.0/ *[Esta licença permite que outros distribuam, remixem, adaptem e criem a partir do seu trabalho, mesmo para fins comerciais, desde que lhe atribuam o devido crédito pela criação original. É a licença mais flexível de todas as licenças disponíveis. É recomendada para maximizar a disseminação e uso dos materiais licenciados.]*

CC BY-SA

https://creativecommons.org/licenses/by-sa/4.0/ *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho, mesmo para fins comerciais, desde que lhe atribuam o devido crédito e que licenciem as novas criações ao abrigo de termos idênticos. Esta licença costuma ser comparada com as licenças de software livre e de código aberto «copyleft». Todos os trabalhos novos baseados no seu terão a mesma licença, portanto quaisquer trabalhos derivados também permitirão o uso comercial. Esta é a licença usada pela Wikipédia e é recomendada para materiais que seriam beneficiados com a incorporação de conteúdos da Wikipédia e de outros projetos com licenciamento semelhante.]*

CC BY-ND

https://creativecommons.org/licenses/by-nd/4.0/ *[Esta licença permite que outras pessoas usem o seu trabalho para qualquer fim, incluindo para fins comerciais. Contudo, o trabalho, na forma adaptada, não poderá ser partilhado com outras pessoas e têm que lhe ser atribuídos os devidos créditos.]*

CC BY-NC

https://creativecommons.org/licenses/by-nc/4.0/ *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho para fins não comerciais, e embora os novos trabalhos tenham de lhe atribuir o devido crédito e não possam ser usados para fins comerciais, eles não têm de licenciar esses trabalhos derivados ao abrigo dos mesmos termos.]*

CC BY-NC-SA

https://creativecommons.org/licenses/by-nc-sa/4.0/ *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho para fins não comerciais, desde que lhe atribuam a si o devido crédito e que licenciem as novas criações ao abrigo de termos idênticos.]*

CC BY-NC-ND

https://creativecommons.org/licenses/by-nc-nd/4.0/ *[Esta é a mais restritiva das nossas seis licenças principais, só permitindo que outros façam download dos seus trabalhos e os comparti-*

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, janeiro 2024

Nome completo do autor André Bernardo Coelho Nunes

Resumo

O objetivo deste projeto é estabelecer um novo nível de abstração em cima da biblioteca de Programação Estratégica (Ztrategic), permitindo a expressão de regras de análise de nomes de forma flexível e universal. Para alcançar esse objetivo é necessário definir estratégias para representar regras de nomes inspiradas em diferentes linguagens, como C, Java, Haskell e outras.

Essas estratégias proporcionarão aos desenvolvedores a capacidade de aplicar regras de Scope específicas de cada linguagem, tornando a análise de nomes mais precisa e eficiente.

No geral, este projeto visa melhorar a consistência e a legibilidade do código-fonte, fornecendo ferramentas poderosas para a análise de nomes e regras de Scope em diferentes contextos de programação.

Os resultados esperados incluem uma melhoria na qualidade do código, aumento da produtividade dos programadores e maior aplicabilidade em diferentes linguagens de programação. Este projeto de pesquisa tem o potencial de revolucionar a abordagem dos programadores em relação à análise de nomes e à consistência no código, tendo um impacto duradouro na indústria de engenharia de software.

Palavras-chave Programação Estratégica, Scope, estratégias, análise de nomes, regras de Scope, Ztrategic

Abstract

The aim of this project is to establish a new level of abstraction on top of the library (Ztrategic), allowing the expression of name analysis rules in a flexible and universal way. To achieve this goal it is necessary to define strategies for representing rules inspired by different languages, such as C, Java, Haskell and others.

These strategies will give developers the ability to apply language-specific Scope rules, making name analysis more precise and efficient.

Overall, this project aims to improve the consistency and readability of source code by providing powerful tools for analyzing names and Scope Rules in different programming contexts.

The expected results include an improvement in code quality and productivity of programmers and greater applicability in different programming languages. This has the potential to revolutionize programmers approach to name analysis and code consistency, having a lasting impact on the software engineering industry.

Keywords Strategic Programming, Scope, strategies, name analysis, Scope rules, Ztrategic

Conteúdo

Lista de Tabelas

Parte I

Material Introdutório

Capítulo 1

Introdução

No contexto do desenvolvimento de software a análise de nome e regras de Scope desempenha um papel crucial. A busca por software mais robusto, eficiente e flexível tem gerado uma procura por abordagens mais precisas e dúcteis de modo a enfrentar os desafios específicos de cada linguagem.

No entanto, as práticas atuais mostram-se muitas vezes limitadas e incapazes de se adaptar às diversas nuances das linguagens existentes. Esta incapacidade reflete-se em métodos de desenvolvimento que usam análises estáticas e manuais, que carecem de atualizações sempre que a linguagem evolui.

A nova abstração possibilitará aos desenvolvedores modelar e aplicar regras de Scope e análise de nomes de maneira eficaz e ajustável, reconhecendo a natureza dinâmica das linguagens de programação. Consequentemente, para cada linguagem, o utilizador apenas necessita de facultar algumas informações acerca da mesma para o sistema derivar regras de análise de Scopes automaticamente.

Por último, este projeto visa revolucionar a abordagem dos desenvolvedores no que toca à análise de nomes e regras de Scope, projetando essas mudanças para que tenham um impacto duradouro na indústria de software, elevando o conceito de programação estratégica, como uma abordagem inovadora e adequada para este tipo de problemas e ajudando a tornar a análise de nomes num componente "plug-and-play"de linguagens, com regras e práticas bem definidas.

1.1 Motivação

O objetivo deste projeto é estabelecer um novo nível de abstração em cima da biblioteca de Programação Estratégica (Ztrategic) Macedo (2022), permitindo a expressão de regras de análise de nomes de forma mais eficaz e flexível uma vez que a natureza dinâmica das linguagens de programação assim o exige, viabilizando lidar com as nuances específicas de cada contexto.

Este projeto emerge então da necessidade pressurosa de aprimorar as práticas de análise de nomes e regras de Scope no âmbito do desenvolvimento de software, reconhencendo a importância que esse que-

sito tem na qualidade e consistencia de código-fonte.

Atualmente a biblioteca de Programação estratégica oferece uma base sólida para abordar esse tipo de questões. Incluindo ferramentas ao nível de Gramáticas de atributos, que permitem expressar algoritmos de análise de nomes de uma forma natural e intuitíva. Processador Block, que pode ser ajustado para obter informações, ainda que incompletas, sobre os erros de Scope de uma linguagem, revelando-se eficaz na modelagem do problema. Zippers, que permitem a navegação uniforme em estruturas de dados independentemente de que linguagem de programaçao representam, bem como estratégias adequadas para a manipulação destes.

No entanto, uma expressão mais flexível e intuitiva das regras de análise de nomes e Scope que combinasse estas técnicas traria um novo caso de uso a estas tecnologias.

1.2 Principais objetivos

O cerne deste projeto é a busca por uma evolução significativa neste âmbito de análise de nomes e regras de Scope utilizando Programação estratégica, Saraiva (2007). Esta iniciativa visa transformar a forma como as análises de nomes e regras de Scope são concebidas e implementadas, almejando impactos substanciais na qualidade do código-fonte bem como a sua validação, evolução e manutenção.

Uma das metas deste projeto é portanto criar essa camada de abstração inovadora que visa simplificar a expressão de regras de Scope, tornando-as mais acessíveis e intuitivas para os desenvolvedores.

Deste modo, é necessário definir um algoritmo capaz de processar regras de Scope de uma determinada linguagem e verificar se estas são aplicadas de forma correta.

A solução proposta visa facilitar este processo eliminando a complexidade associada à análise de nomes e regras de Scope. Assim, será desenvolvida uma interface que permitirá aos programadores especificar regras de Scope de forma intuitiva e universal. Após essa definição, a abstração desenvolvida terá a capacidade de efetuar todo o tipo de verificações necessárias, produzindo um relatório de erros após o seu processamento, e validando assim, ou não, a coerência de nomes e regras de Scope do código a analisar.

De notar que este projeto não é uma busca por eficiência técnica mas sim uma pretenção pela melhoria da qualidade do código e um aumento de expressividade para os desenvolvedores de linguagens.

1.3 Questões de investigação

Atualmente existem diversas soluções para atender ao problema de resolução de nomes e regras de Scope. No entanto, nenhuma delas se mostra abrangente o suficiente para ser adaptável a qualquer linguagem de uma forma elegante e eficaz, tornando esta prática simples e efetuada de um modo "plug-and-play".

O objetivo deste trabalho é fornecer aos desenvolvedores uma ferramenta prática e eficiente, mas ao mesmo tempo suficientemente abrangente para lidar com este problema tornando transparentes todas as intrícacias relativas à resolução de nomes e regras de Scope.

Embora muitas questões tenham surgido de forma colateral e espontânea durante a fase inicial de pesquisa, as seguintes Questões de Investigação de Tese (QIT) serão o foco deste trabalho:

- **QIT1:** Que influência tem a resolução de nomes e regras de Scope no desenvolvimento, validação, manutenção e evolução de código fonte?

  Elaboração de QIT1:

- **QIT2:** De que forma podemos tornar a resolução de nomes e regras de Scope numa prática transparente e abrangente usando programação estratégica?

  Elaboração de QIT2:

- **QIT3:** Será viável efetuar a fusão entre o funcionamento do processador da linguagem Block e o processo de parsing de uma linguagem de modo a efetuar a análise de nomes e regras de Scope sem perda de informação da linguagem em questão?

  Elaboração de QIT3:

Capítulo 2

Estado da arte

Neste capítulo encontram-se relatados os conceitos abordados durante a dissertação em questão.

Inicialmente será apresentada e interpretada toda a teoria, relacionada com a resolução de nomes Neron (2015), bem como a vinculação de nomes declarativa e regras de Scope, Konat (2013), assentando as noções necessárias para a consequente exposição de conceitos.

Depois iremos explorar como os **Ambientes de desenvolvimento integrado** atuais lidam com as nuances e problemas adjacentes à resolução de nomes e verificação de regras de Scope, Konat (2013), de modo a constatar tanto as suas limitações bem como os seus pontos fortes. Por forma a verificar que técnicas podem ser reaproveitadas e que aspetos precisam de uma evolução neste âmbito.

Posteriormente abordaremos os conceitos relacionados com a associação de Scopes como tipos Antwerpen (2018), os quais apresentam a teoria base para o uso de gramáticas de atributos e Programação estratégica na resolução de nomes e verificação de regras de Scope.

Por último, teorizaremos de que forma podemos combinar essas ferramentas para desenvolver a nova camada de abstração na biblioteca de Programação estratégica (Ztrategic), utilizando uma abordagem alicerçada em gramáticas de atributos com recurso a Zippers e estratégias.

2.1 Vinculação de nomes declarativa e regras de Scope

A vinculação declarativa de nomes diz respeito à relação entre definições e referências de identificadores em linguagens de software textuais, incluindo as regras de Scope que regem estas relações de acordo com o trabalho de Konat (2013). No contexto de processadores de linguagens é essencial recolher informações acerca das definições disponíveis e as suas respetivas referências.

Estas práticas desempenham um papel fundamental em vários processos de engenharia de linguagens como ambientes de desenvolvimento integrado, resolução de referências, conclusão de código, refactorings, verificação de tipos e compilação.

Os diferentes requisitos inerentes a diferentes linguagens conduzem a múltiplas reimplementações das regras de Scope de nomes para cada um destes objetivos, ou a uma integração manual não trivial de uma única implementação que suporte todos os objetivos. Isto resulta em duplicação de código, tendo como resultado erros, inconsistências e maior esforço de manutenção.

2.1.1 Scopes

Um Scope refere-se à área em que uma função ou variável é visível e acessível a outro código, restringindo a visibilidade dos sítios de definição.

Como tal, é necessário classificar esses mesmos Scopes em duas categorias distintas como referido em Konat (2013), Named Scopes e Anonymous Scopes. Named Scopes são o locais de definição de um nome que definem o âmbito de outros locais de definição. Por outro lado, Anonymous Scopes não definem um nome mas sim um local de ação.

Devido à sua natureza e propósito os Scopes normalmente são aninhados e portanto a resolução de nomes procura uma implementação de Scopes interiores para Scopes exteriores.

2.1.2 Algoritmo de resolução de nomes

Segundo o Konat (2013), um algoritmo que busque dar resposta a este problema deve ser dividido em três fases distintas. Fase de anotação, fase de definição do local de análise e fase de análise do local de uso.

Nesta primeira fase a **AST**[1] é percorrida de forma descendente de forma a recolher informações sobre todos os sítios de definição e uso de modo a gerar uma referência para as fases seguintes.

A segunda fase analisa cada local de definição, outra vez de forma descendente, com o objetivo de recolher informações sobre essas definições, como o seu tipo, e armazena essa informação para ser processada pela próxima fase.

Por último são resolvidas as referências e determinados os tipos que dependem de informação não local.

2.2 Teoria de resolução de Nomes e Scopes como tipos

A resolução de nomes é muitas vezes complexa, uma vez que atravessa a estrutura indutiva local dos programas (descrita por uma **AST**). Esta resolução está subjacente à maior parte das operações em

[1] Árvores de sintaxe abstratas são estruturas de dados em árvore que representam estruturas sintáticas de cadeias, de acordo com uma gramática formal.

linguagens e programas, incluindo verificação estática, tradução, descrição mecanizada da semântica e fornecimento de serviços de editor em **IDE**s.

Para tal, é necessário uma teoria que unifique todos os conceitos inerentes, mas que por outro lado, seja suficientemente fléxivel para atender às nuances específicas de cada linguagem.

O trabalho de Neron (2015) fornece uma teoria independente de linguagem adequada a linguagens com regras de Scope complexas incluido Scoping lexical[2] e modular[3].

2.2.1 Simply-Typed Lambda Calculus

Em Neron (2015) somos presenteados com uma especificação declarativa da resolução de referências para declarações por meio de um Lambda Calculus[4] que define essa mesma resolução num grafo de Scope. Esta teoria será fundamental no desenvolvimento da nova camada de abstração sobre a biblioteca de Programação Estratégica, Macedo (2022), uma vez que servirá como base teórica para a implementação produzida. Visto que prova por meio de métodos formais as complexidades deste tipo de operações.

2.2.2 Grafos de Scope e a sua construção

Um grafo de Scope captura a estrutura de ligação de um programa. Um Scope é um local num programa que se comporta de forma uniforme relativamente à resolução de nome.

Segundo Antwerpen (2018), no contexto de um grafo de Scope, as declarações e referências de nomes estão associadas a Scopes. Assim, estes são representados por meio de nodos e a sua visibilidade e acessibilidade é modelada por arestas entre Scopes.

Um algoritmo de resolução de nomes interpreta um grafo de Scope para resolver referências a declarações, encontrando o caminho bem formado mais específico.

Para exprimir as regras de Scope de uma linguagem de programação, define-se um mapeamento de **Árvores de sintaxe abstrata** para grafos de Scope. O mapeamento agrupa todos os nós da **AST** que se comportam uniformemente em relação à resolução de nomes num único nó de Scope.

Para qualquer linguagem, a construção pode ser especificada por uma definição convencional dirigida por sintaxe sobre a gramática da linguagem.

[2] É a área de definição de uma expressão. Por outras palavras, o Scope lexical de um item é o local onde foi criado.

[3] Scope referente a uma área de código modular como por exemplo uma classe ou interface do Java.

[4] Um sistema formal em lógica matemática para expressar computação com base na abstração e aplicação de funções usando ligação e substituição de variáveis.

2.2.3 Cálculo e procura de caminhos de resolução

No trabalho de Neron (2015) é explicado que um algoritmo de resolução e procura de caminhos num grafo de Scope é correto se for acíclico e se todos os nomes tiverem um caminho de Scope visível e acessível. O algoritmo descrito nesse mesmo artigo fornece uma base para a implementação de ferramentas assentes em grafos de Scope, como é o caso do objetivo da presente tese.

2.3 Ambientes de desenvolvimento integrado na resolução de nomes

Nos dias de hoje, os **IDE** atuais fornecem diversos serviços no que toca a resolução de nomes. No entanto, como foi referido anteriomente todos eles carecem de uma flexibilidade e eficiência para lidar com as nuances específicas de cada linguagem.

Geralmente, esses serviços são desenvolvidos manualmente e feitos especificamente para cada linguagem, o que exige um trabalho substancial, tanto no seu desenvolvimento bem como na sua manutenção e evolução.

Não obstante, modelando essas relações é possível gerar um algoritmo de resolução de nomes que pode posteriormente ser usado por estes serviços de edição de forma menos complexa e trabalhosa, como é referido no trabalho de Konat (2013).

2.3.1 Resolução de referências

Relativamente à resolução de referências, a maior parte dos **IDE** já oferece uma gama vasta de soluções com funcionalidades como encontrar a definição de um nome, a definição do seu tipo, as suas referências e as suas implementações se esse for o caso.

No entanto, como foi já referido estas ferramentas só existem para línguagens específicas para as quais foram desenvolvidos esses serviços de forma manual.

2.3.2 Verificação de restrições

Atualmente, os **Ambientes de desenvolvimento integrado** realizam de forma automática algumas verificações estáticas para uma diversa gama de restrições.

Essas verificações ocorrem em tempo real durante a digitação do código e são apresentadas ao utilizador,

diretamente no editor, por meio de marcadores de erro no texto.

As restrições verificadas incluem erros comuns de ligação de nomes, como referências não resolvidas, definições duplicadas, uso antes da definição e definições não utilizadas.

2.3.3 Conclusão de código

No que diz respeito à conclusão de nomes, já existem alguns serviços que preenchem código incompleto com referências válidas no contexto da execução do código.

Nesse sentido, o GitHub Copilot tem vindo a destacar-se nos últimos tempos como referido em Finnie-Ansley (2022), como uma ferramenta de inteligência artificial desenvolvida pelo GitHub em conjunto com a OpenAI, no sentido de auxiliar utilizadores de **Ambientes de desenvolvimento integrado** como Visual Studio Code(VSC), InteliJ(IJ) e Neovim(NV), fornecendo aos desenvolvedores sugestões de código para preenchimento automático.

2.4 Gramáticas de atributos

Segundo Knuth (1968), as gramáticas de atributos são um formalismo que permite especificar a análise semântica de um compilador e modelar algoritmos complexos de travessia.

No contexto de gramáticas de atributos, os programadores não têm a necessidade de desenvolver funções de travessia complexas uma vez que estes mecanismos são gerados por sistemas baseados em gramáticas de atributos.

2.4.1 Zippers

O conceito de Zippers foi inicialmente concebido por Huet (1997), de modo a possíbilitar a representação e navegação uniforme em estruturas de dados, independentemente dos dados que representam.

Num Zipper, existe um foco de atenção relativo às estruturas a analisar que é completamente móvel, permitindo movimentações em todas as direções.

A manipulação de um Zipper é efetuada através de um conjunto de funções predefinidas, que permitem o acesso genérico a todos os nós da árvore para a sua consulta ou alteração.

2.4.2 Gramáticas de atributos baseada em Zippers

No trabalho de Martins (2013), podemos verificar que de facto é possível combinar Zippers e gramáticas de atributos de modo a criar um mecanismo adaptável, eficiente e elegante de manipulação e travessia de árvores, sem a necessidade de desenhar toda a maquinaria necessária para este fim. Foi então criada a biblioteca de programação estratégica (ZAG).

Esta biblioteca será o fundamental para esta tese, uma vez que, o objetivo é estendê-la de acordo com os termos referidos anteriormente.

2.4.3 Programação estratégica

Uma estratégia é uma função de transformação genérica que pode atravessar estruturas de dados heterogéneas, combinando um comportamento uniforme e específico do tipo, como referido no trabalho de Saraiva (2007).

Assim, num ambiente de Programação estratégica apenas os nós a transformar são incluídos na solução, omitindo as produções em que não é necessário efetuar qualquer tipo de alteração.

Recorrendo a este tipo de mecanismos, obtemos um estilo bem mais elegante e sintético de expressar transformações sobre árvores.

Este conceito será de extrema relevância neste projeto, visto que nos serviremos dele e de uma biblioteca que o modela para desenvolver uma nova camada de abstração sobre essa mesma biblioteca.

2.4.4 Análise de nomes com o processador Block

No contexto da biblioteca de programação estratégica (Ztrategic) somos presenteados com o conceito da linguagem baseada em listas (Block), que consiste numa lista possivelmente vazia de itens. Um item pode ser uma declaração de um nome, o uso de um nome ou um bloco aninhado.

Esta linguagem permite ao utilizador recolher informações acerca do uso de nomes de um determinado excerto de código, descrevevendo o comportamento da maioria das linguagens de programação relativamente a declarações e uso de variáveis.

Esta abordagem garante uma análise sistemática das regras de Scope na linguagem Block, incluindo a identificação de definições locais, prevenção de duplicações e verificação do uso adequado de nomes definidos de forma concisa e eficiente.

No entanto, não é abrangente o suficiente e requer a tradução do código fonte em linguagem Block para efetuar as verificações. Essa transformação é exequível mas tem os seus problemas, nomeadamente não

saber apontar em que local estava o erro originalmente.

Portanto, a intenção é empregar o conceito de como o Block opera, mas conduzir essa análise sobre o tipo de dados original da forma mais genérica possível, tornando assim este processo universal e transparente.

Capítulo 3

Planeamento

Após o trabalho de pesquisa efetuado que culminou com a escrita do relatório de pré-dissertação será desenvolvido um protótipo simples, de modo a provar que a implementação é de facto possível e viável. De seguida, será extendida de forma gradual essa mesma implementação almejando tornar a ferramenta o mais "sound and complete" possível. Consequentemente será efetuada uma validação com recurso a vários exemplos, com várias linguagens de modo a provar a eficácia do projeto. Está também planeado a escrita e submissão de um artigo em conferência relativamente à técnica desenvolvida.

3.1 Atividades

Tarefa	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul
Background e **EA**	●	●	●							
Preparação do **RPD**		●	●	●						
Desenvolvimento da técnica					●	●	●	●		
Desenvolvimento da biblioteca					●	●	●	●		
Validação								●		
Escrita									●	●
Submissão de artigo em conferência								●		

Tabela 1: Plano de atividades.

Bibliografia

Intellij idea – the leading java and kotlin ide. https://www.jetbrains.com/idea/. Accessed: 2024-01-20.

Neovim. https://neovim.io/. Accessed: 2024-01-20.

Visual studio code – code editing. redefined. https://code.visualstudio.com/. Accessed: 2024-01-20.

Zipperag: An implementationg of attribute grammars using functional zippers. https://hackage.haskell.org/package/ZipperAG-0.9. Accessed: 2024-01-20.

Hendrik Van Antwerpen. *Scopes as Types*. OOPSLA, 2018.

James Finnie-Ansley. *The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming*. ACM ISBN, 2022.

Gérard Huet. *FUNCTIONAL PEARL*. Cambridge University Press, 1997.

Donald E. Knuth. *Semantics of context-free languages*. Springer, 1968.

Gabriel D. P. Konat. *Declarative Name Binding and Scope Rules*. Springer, 2013.

José Nuno Macedo. *Zipping Strategies and Attribute Grammars*. FLOPS, 2022.

Pedro Martins. *Zipper-Based Attribute Grammars and Their Extensions*. Springer, 2013.

Pierre Neron. *A Theory of Name Resolution*. Springer-Verlag Berlin Heidelberg, 2015.

João Saraiva. *The Fun of Programming with Attribute Grammars*. Detailed summary of lecture as per Art. 8.o Dec.-Lei 239, 2007.