INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Campus
Irecê

Átila Gama Silva

Witch Cooking: Formatação de Código via Linguagem de Consulta do *Tree-Sitter*

Irecê

2023

Átila Gama Silva

Witch Cooking: Formatação de Código via Linguagem de Consulta do *Tree-Sitter*

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Bahia – Campus Irecê, como requisito parcial para obtenção do diploma de Técnico(a) em Análise e Desenvolvimento de Sistemas, mediante a orientação do professor Rafael Xavier.

Irecê

2023

Resumo

A formatação de código é fundamental no desenvolvimento de software, permitindo estabelecer aspectos desejados como a padronização e legibilidade do código, que impactam positivamente o ciclo de vida do software. Em ambientes de desenvolvimento modernos, é comum o uso de ferramentas para automatizar a formatação de código. No entanto, essas ferramentas apresentam limitações no número de linguagens suportadas, nas opções de configuração e não permitem a definição de regras de formatação personalizadas. Visando superar ou reduzir as limitações frequentemente encontradas nas ferramentas convencionais de formatação, neste trabalho é apresentado o *Witch Cooking*, um software protótipo que tem como objetivos: (I) abranger uma gama de linguagens de programação; além de (II) permitir que o usuário defina suas próprias regras de formatação.

Palavras-chave: formatação de código; *Tree-Sitter*; *prettyprint*; customização.

Sumário

1 Introdução

A flexibilidade presente na sintaxe de linguagens de programação permite que diferentes arranjos dum mesmo código compartilhem um valor sintático equivalente. Essa característica possibilita a formatação do código de acordo com aspectos desejados, como a legibilidade, que é fundamental no ciclo de vida do software (BUSE; WEIMER, 2009; OLIVEIRA et al., 2020). Consequentemente, a formatação é frequentemente utilizada para estabelecer um nível satisfatório de legibilidade em bases de código.

A formatação de código pode ser realizada manualmente pelo programador, embora esse processo possa ser demorado, especialmente em grandes bases de código, exigindo do programador um tempo que poderia ser empregado em outra tarefa. Além disso, a formatação manual também pode ser falha e inconsistente devido à suscetibilidade humana ao erro, podendo ser agravada quando há múltiplos programadores em uma base de código.

Para evitar os problemas inerentes da formatação manual, é comum a utilização de softwares que automatizam a formatação de forma determinística, tornando o código padronizado e consistente. No entanto, é comum que essas ferramentas de formatação sejam limitadas a uma linguagem ou a uma família de linguagens de programação. Além disso, algumas delas são *opinionated*, assim restringindo as possiblidades de customização do usuário. Finalmente, é também comum que essas ferramentas de formatação não permitam que o usuário defina regras de formatação personalizadas.

Em ambientes de desenvolvimento modernos, além da utilização de ferramentas que automatizam a formatação de código, é também comum a utilização do *Tree-Sitter* (TREE-SITTER..., 2023): um sistema de análise sintática de código aberto que foi disponibilizado ao público geral no GitHub primeiramente em 2019, tendo sido inicialmente desenvolvido por Max Brunsfeld. Desde seu lançamento, o *Tree-Sitter* tem ganhado popularidade na comunidade de desenvolvedores devido dentre outros motivos à: (I) sua capacidade de suportar várias linguagens de programação; além de (II) sua linguagem de consulta, a qual permite a realização de buscas complexas na árvore analisada de um código fonte.

Visando superar ou reduzir as limitações previamente mencionadas, quais são frequentemente presentes nas ferramentas convencionais de formatação, este trabalho tem como objetivo geral desenvolver um software protótipo para a formatação de código. Nomeada antecipadamente de *Witch Cooking* (SILVA, 2023) pelo autor, a ferramenta a ser desenvolvida tem como objetivos: (I) abranger uma gama de linguagens de programação; além de (II) permitir que o usuário defina suas próprias regras de formatação.

Para alcançar o objetivo geral proposto, este trabalho tem como objetivos específicos: (I) apresentar e contextualizar ferramentas de formatação de código conceituadas, além de abordar seus métodos de formatação; (II) conceituar, contextualizar e abordar as tecnologias do sistema *Tree-Sitter*; e, por fim, (III) utilizar o *Tree-Sitter* para desenvolver e atender às aspirações do *Witch Cooking*.

2 A Formatação de Código-Fonte

2.1 Introdução à Formatação de Código-Fonte: Conceitos e Importância

Desde os primórdios da computação, métodos foram desenvolvidos para garantir que a saída impressa fosse formatada de maneira esteticamente agradável (HARRIS, 1956 apud YELLAND, 2015, p. 1). Esses métodos ganharam popularidade sob o termo *prettyprinting*, que se refere à formatação visual de diversos tipos de conteúdo. No contexto do desenvolvimento de software, o *prettyprinting* é conhecido como formatação de código: uma prática histórica e comum que envolve a adoção de convenções estilísticas para estruturar o código-fonte. Existem diversas terminologias utilizadas para se referir às ferramentas que realizam a formatação do código-fonte, incluindo: (I) formatadores de código, ou *code formatters* em inglês; (II) *prettyprinters*; ou ainda (III) *beautifiers*.

Inicialmente, a formatação de código era amplamente restrita a operações simples, como a quebra de linhas e a inserção de espaços em branco (YELLAND, 2015, p. 1). No entanto, ao longo do tempo, houve a necessidade de adaptação à crescente complexidade e tamanho do código-fonte, bem como às demandas contemporâneas do ambiente de desenvolvimento. Consequentemente, existem atualmente ferramentas mais avançadas capazes de, por exemplo: (I) realizar operações sofisticadas com base na análise semântica, superando, assim, a abordagem tradicional restrita à análise sintática; além de (II) formatar o código para tornar "a revisão de código mais rápida, produzindo os menores *diff*s possíveis" (BLACK. . . , 2023, tradução nossa).

A formatação de código é uma prática historicamente comum no desenvolvimento de software, que envolve estabelecer convenções estilísticas para estruturar o código-fonte. Desde seu surgimento, a formatação de código tem sido aprimorada para acompanhar a complexidade e o tamanho do código-fonte, bem como as demandas contemporâneas do ambiente de desenvolvimento. Inicialmente, a formatação de código se limitava a operações simples, como quebra de linhas e inserção de espaços em branco. No entanto, atualmente existem ferramentas mais sofisticadas, como o *rustfmt* (RUSTFMT. . . , 2023) e o *clang-format* (CLANG. . . , 2023), capazes de realizar operações avançadas com base na análise semântica.

De acordo com Buse e Weimer (2009, p. 1), a *readability*[1] de um programa está relacionada à sua manutenibilidade e, portanto, é um fator-chave na qualidade geral do software. Nesse contexto, a formatação de código moderna desempenha um papel fundamental, visando principalmente: (I) arranjar o código de forma padronizada e consistente; (II) beneficiar a legibilidade e compreensão do código; e, por fim, (III) facilitar a manutenção do código.

[1] Segundo Oliveira et al. (2020), na engenharia de software, os termos *readability*, *legibility*, *understandability* e *comprehensibility* têm significados sobrepostos.

3 O Sistema *Tree-Sitter*

O *Tree-Sitter* é um sistema multilingual de análise sintática para ferramentas de programação inicialmente desenvolvido como um projeto secundário por Max Brunsfeld. Como relatado pelo próprio autor (TREE-SITTER. . . , 2017), o *Tree-Sitter* surgiu como uma tentativa de solucionar problemas presentes nas ferramentas de análise sintática da época. Mais especificamente, o sistema tinha como objetivos: (I) ser utilizado no ambiente de desenvolvimento para produzir árvores de sintaxe a partir da análise de códigos escritos em várias linguagens; (II) implementar a análise incremental, permitindo a atualização da árvore de sintaxe em tempo real; (III) expor através da árvore de sintaxe os nós representando suas construções gramaticais no código (*e.g.*, classes, funções, declarações, etc.), diferentemente das ferramentas contemporâneas, que utilizavam uma abordagem simplística baseada em expressões regulares; e, por fim, (IV) ser livre de dependências, assim beneficiando sua adoção e aplicabilidade.

Além das funcionalidades previamente mencionadas presentes no *Tree-Sitter*, o sistema também conta uma ferramenta de linha de comando que pode ser utilizada para gerar *parsers* para uma linguagem a partir de sua gramática. A gramática é definida via a linguagem de programação JavaScript, a qual: (I) foi eleita 15 vezes seguidas pela *Stack Overflow Developer Survey* (STACK. . . , 2022) como a linguagem de programação mais comumente usada; além de (II) ser amplamente considerada pela comunidade de programadores como uma das linguagens mais fáceis de aprender e programar (11. . . , 2023; GOEL, 2023; JAVASCRIPT. . . , 2023). A ferramenta de geração de *parsers* também disponibiliza funções preestabelecidas para permitir a criação de gramáticas com diferentes níveis de complexidade. Não é surpreendente que, devido a essas características e facilidades presentes na criação de *parsers*, exista uma variedade de linguagens de programação e formatos de arquivos – variando de linguagens com sintaxes complexas, como C++ e Perl, a formatos de arquivos mais específicos, como *.vhs* e *.rasi* –, os quais têm parsers gerados pelo *Tree-Sitter* e são suportados pelo sistema.

Na análise de código, é comum realizar tarefas que envolvem a busca de padrões em árvores sintáticas. Para isso, o *Tree-Sitter* oferece uma pequena linguagem de consulta declarativa que é capaz de expressar esses padrões por meio de *S-expression* e buscar correspondências. A linguagem de consulta suporta operadores que permitem: (I) a captura de nós; (II) a quantificação de nós, análoga às expressões regulares; (III) o agrupamento de nós; (IV) as alternâncias de nós; (V) o uso de *wildcards*; e (VI) a ordenação de nós. Adicionalmente, é possível definir propriedades nos nós da árvore sintática usando a linguagem de consulta do *Tree-Sitter*. Também é permitido o uso de predicados – funções arbitrárias para filtrar nós ou realizar verificações mais complexas durante a busca de padrões – sejam eles *builtins* ou estendidos por meio de uma API.

Um exemplo da relevância e utilidade da linguagem de consulta é o plugin *nvim-treesitter* (NVIM. . . , 2023), que é frequentemente utilizado no editor *Neovim* (HYPEREXTENSIBLE. . . , 2023). Esse plugin utiliza a linguagem de consulta para definir diferentes recursos, tais como: (I) *folds*, que permitem agrupar blocos de código; (II) highlights, que realçam a sintaxe do código;

(III) indentações, que definem a estrutura do código; (IV) injeções, que permitem adicionar novas sintaxes a arquivos existentes; além de (V) captura de nós correspondentes a construções gramaticais (*e.g.*, funções, classes, métodos, etc.), os quais são frequentemente utilizados em rotinas de programação tais como a remoção e navegação do código.

Em suma, o sistema *Tree-Sitter* se mostrou uma solução inovadora e eficiente para a análise sintática de códigos em diversas linguagens de programação, com uma abordagem diferenciada e sofisticada que possibilita a atualização em tempo real das árvores sintáticas e a identificação precisa das construções gramaticais presentes no código. Além disso, a ferramenta de linha de comando disponível no sistema facilita a geração de parsers a partir de gramáticas definidas em JavaScript, o que torna o processo mais acessível e personalizável para os programadores. Finalmente, a linguagem de consulta declarativa oferecida pelo *Tree-Sitter* se mostrou uma importante *feature*, sendo utilizada em diversos plugins de editores de código para realizar tarefas variadas e sofisticadas, contribuindo significativamente no ambiente de desenvolvimento.

4 O *Witch Cooking*

O *Witch Cooking* surgiu das necessidades do autor de: (I) formatar código utilizando estilos não convencionais; (II) formatar linguagens não-populares, as quais eram desprovidas de formatadores mais sofisticados.

Referências

11 Most In-Demand Programming Languages. Berkeley Extension. Disponível em: <https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages>. Acesso em: 5 mai. 2023.

BLACK: The uncompromising Python code formatter. Black. Disponível em: <https://black.readthedocs.io/>. Acesso em: 8 mai. 2023.

BUSE, Raymond P. L.; WEIMER, Westley R. Learning a Metric for Code Readability. **IEEE Transactions on Software Engineering**, IEEE, v. 36, n. 4, p. 546–558, 2009. DOI: 10.1109/TSE.2009.70.

TREE-SITTER: a new parsing system for programming tools - GitHub Universe 2017. 1 vídeo (43 min). GitHub. 2017. Disponível em: <https://youtu.be/a1rC79DHpmY>. Acesso em: 30 abr. 2023.

GOEL, Aman. **How to Learn JavaScript in 2023 | 8 Best Ways For Beginners**. Disponível em: <https://hackr.io/blog/how-to-learn-javascript>. Acesso em: 4 mai. 2023.

HARRIS, R. W. Keyboard Standardization. **Western Union Technical Review**, v. 10, n. 1, p. 37–42, 1956.

CLANG 17.0.0git documentation. LLVM. Disponível em: <https://clang.llvm.org/docs/ClangFormat.html>. Acesso em: 15 mai. 2023.

HYPEREXTENSIBLE Vim-based text editor. Neovim. Disponível em: <https://neovim.io/>. Acesso em: 7 mai. 2023.

NVIM Treesitter configurations and abstraction layer. nvim-treesitter. Disponível em: <https://github.com/nvim-treesitter/nvim-treesitter>. Acesso em: 7 mai. 2023.

OLIVEIRA, Delano et al. Evaluating Code Readability and Legibility: An Examination of Human-centric Studies. In: IEEE. 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). [S.l.: s.n.], 2020. DOI: 10.1109/icsme46990.2020.00041.

RUSTFMT. rust-lang. Disponível em: <https://rust-lang.github.io/rustfmt>. Acesso em: 15 mai. 2023.

SILVA, Átila Gama. **Witch Cooking**: Experimental multilingual code formatter based on Tree-Sitter's query. Disponível em: <https://github.com/atchim/witch-cooking>. Acesso em: 27 abr. 2023.

STACK Overflow Developer Survey. Stack Overflow. 2022. Disponível em: <https://survey.stackoverflow.co/2022>. Acesso em: 1 mai. 2023.

TREE-SITTER: a parsing system for programming tools. Tree-Sitter. Disponível em: <https://tree-sitter.github.io/>. Acesso em: 6 abr. 2023.

JAVASCRIPT Tutorial. W3Schools. Disponível em: <https://www.w3schools.com/js>. Acesso em: 5 mai. 2023.

YELLAND, Phillip M. A New Approach to Optimal Code Formatting. **Google Research**, 2015. Disponível em: <https://research.google.com/pubs/archive/44667.pdf>. Acesso em: 7 abr. 2023.

Glossário

B

Builtin Termo usado na programação para descrever funções ou comandos que fazem parte do próprio sistema ou ambiente de programação, em oposição a funções ou comandos definidos pelo próprio usuário. Por exemplo, em Python, *print* é um comando *builtin* porque faz parte do próprio ambiente de programação, enquanto uma função definida pelo usuário seria algo criado pelo próprio programador para atender às suas necessidades específicas. Esses comandos ou funções *builtin* estão geralmente disponíveis para o usuário em tempo de execução, sem precisar serem definidos ou importados explicitamente no código.

D

Diff No desenvolvimento de software, *diff* (ou diferença) se refere à comparação entre duas versões de um arquivo ou conjunto de arquivos. Especificamente, o *diff* mostra as alterações feitas entre as versões, destacando as linhas adicionadas, removidas ou modificadas. O *diff* é comumente utilizado para visualizar e revisar as mudanças feitas no código-fonte durante o processo de desenvolvimento colaborativo.

F

Feature Característica distintiva de um item de software (por exemplo, desempenho, portabilidade ou funcionalidade).

Fold No contexto de editores de texto e IDEs, *fold* geralmente se refere a um recurso que permite esconder (ou "dobrar") um bloco de código ou texto para tornar mais fácil a navegação e a visualização do conteúdo.

O

Opinionated Termo utilizado no desenvolvimento de software para se referir a um conjunto de práticas preestabelecidas sobre como abordar uma determinada tarefa, podendo não permitir customizações ou desvio das abordagens.

P

Parser Software que analisa a estrutura sintática de uma sequência de símbolos, geralmente de acordo com uma gramática formal. Ele é comumente utilizado na compilação de linguagens de programação para verificar se o código fonte está escrito corretamente de acordo com as regras da linguagem. O *parser* recebe como entrada o código fonte

e produz uma representação interna da estrutura sintática do código em uma forma que possa ser manipulada pelo compilador ou outro programa.

Prettyprinting *prettyprinting* é a aplicação de diversas convenções estilísticas de formatação a arquivos de texto, como código-fonte, marcação e conteúdos similares. Essas convenções de formatação podem incluir o uso de estilos de indentação, cores e tipos de fonte diferentes para destacar elementos sintáticos do código-fonte, ou ajustes de tamanho, para tornar o conteúdo mais fácil de ser lido e compreendido por pessoas. *Prettyprinters* para código-fonte são às vezes chamados de formatadores de código ou *beautifiers*.

S

S-expression (sexp) Na programação de computadores, uma *S-expression* (ou expressão simbólica, abreviada como *sexpr* ou *sexp*) é uma expressão em uma notação de mesmo nome para dados em lista aninhada (estruturados em árvore). As *S-expressions* foram inventadas e popularizadas pela linguagem de programação LISP, que as utiliza tanto para código-fonte quanto para dados.

W

Wildcard Termo usado em várias áreas da computação para se referir a um caractere ou símbolo especial que pode ser usado para representar qualquer outro caractere ou conjunto de caracteres em uma determinada operação. Em geral, o *wildcard* é utilizado para permitir a busca ou correspondência de padrões mais flexíveis e abrangentes em textos, arquivos ou expressões regulares. O uso mais comum de *wildcards* é em expressões regulares, onde um caractere especial é usado para representar um conjunto de caracteres desconhecidos ou variáveis em uma string.