

Witch Cooking

Formatação Multilíngue e Personalizada de Código-Fonte via o Sistema
Tree-Sitter

Átila Gama Silva

3 de dezembro de 2023



INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Campus
Irecê

Motivações

- ▶ Dificuldades ao estudar estilos de formatação em diversas linguagens
 - ▶ Estilos convencionais
 - ▶ Recorrência a diferentes *prettyprinters*
 - ▶ Diferentes configurações e níveis de suporte
 - ▶ Estilos não convencionais
 - ▶ Aplicação manual inevitável
 - ▶ Consumo de tempo e esforço

Problemática

- ▶ Em geral, os formatadores de código
 - ▶ São restritos a
 - ▶ Uma linguagem específica
 - ▶ Uma família de linguagens de programação
 - ▶ São limitados nas configurações de estilização
 - ▶ Proporcionam pouca personalização

Objetivos Gerais

- ▶ Desenvolver um software de linha de comando
- ▶ De natureza prototípica
- ▶ Para a formatação de código-fonte
- ▶ Tendo como objetivos
 - ▶ Ser multilíngue
 - ▶ Proporcionar a formatação personalizada
 - ▶ Via a linguagem de consulta do *Tree-Sitter*

Objetivos Específicos

- ▶ Desenvolver um algoritmo de formatação
 - ▶ Fundamentado no *Tree-Sitter*
- ▶ Definir configurações de estilização para o predicado `set !`
- ▶ Estender os predicados da linguagem de consulta
 - ▶ Proporcionando predicados basais para a formatação

Resultados Esperados

- ▶ Abranger qualquer linguagem suportada pelo *Tree-Sitter*
- ▶ Possibilitar procedimentos básicos de formatação
 - ▶ Através dos predicados desenvolvidos

Limitações

- ▶ Desenvolvimento de predicados limitado a procedimentos básicos de formatação
- ▶ Ausência de mecanismos sofisticados de formatação
 - ▶ E.g., formatação condicional

A Formatação de Código-Fonte

- ▶ **Oppen (1980)** apresentou um algoritmo de formatação multilíngue
 - ▶ Baseado em anotações delimitando blocos
 - ▶ Feitas por uma ferramenta intermediária
- ▶ **Yelland (2015)** descreveu um algoritmo de otimização de layout do código
 - ▶ Relativo a uma noção intuitiva de custo de layout
 - ▶ Onde empregou-se os *combinators*
 - ▶ Funções geradoras descrevendo layouts alternativos para o código

O Sistema *Tree-Sitter*

- ▶ Sistema multilíngue de análise sintática
- ▶ Oferece uma linguagem de consulta declarativa
 - ▶ Expressa padrões da árvore sintática
 - ▶ Por meio de *S-expressions*
 - ▶ Busca correspondências
 - ▶ Proporciona o uso — e extensão — de predicados
 - ▶ Funções arbitrárias que filtram nós e realizam verificações complexas

Materiais

- ▶ Ecosystema Rust
- ▶ Sistema/biblioteca *Tree-Sitter* ([TREE-SITTER...](#), 2023)
- ▶ Ecosystema Neovim

Métodos

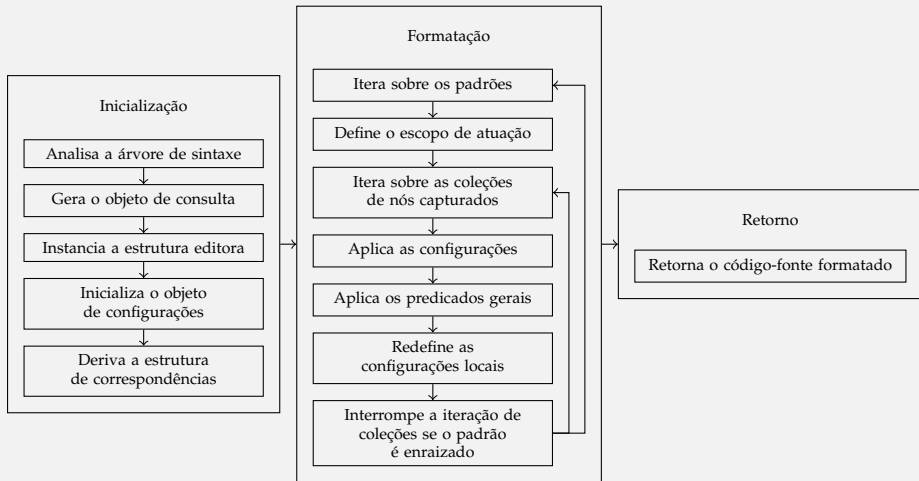
- ▶ Pesquisa experimental
 - ▶ Explorando a aplicação do *Tree-Sitter*
 - ▶ Como base para o algoritmo de formatação
- ▶ Estudo de caso
 - ▶ Analisando a eficácia do software desenvolvido

Usagem

► `cook [-l LANG] -q QUERY [SRC]`

O Algoritmo de Formatação

Fluxograma do Algoritmo de Formatação



As Diretrizes de Formatação

- ▶ Especificadas no arquivo submetido via `-q QUERY`
- ▶ Para realizar um procedimento de formatação, é necessário
 - ▶ Definir um padrão de correspondência
 - ▶ Delimitando o escopo de operação
 - ▶ Capturar nós que
 - ▶ Sejam alvos do procedimento
 - ▶ Auxiliarão nas operações
 - ▶ Opcionalmente, aplicar configurações — via o predicado `set!`
 - ▶ Aplicar os predicados estendidos pelo *Witch Cooking*

As Configurações

- ▶ `indent-rule`
 - ▶ Define a regra de indentação para um nó
 - ▶ Seu valor pode ser
 - ▶ Um inteiro não negativo — sem sinal
 - ▶ Um inteiro positivo — com sinal
 - ▶ Um inteiro negativo
- ▶ `indent-style`
 - ▶ Define a string usada para indentar
 - ▶ Pode ser de escopo local ou global
 - ▶ Não aplicável a nós

Os Predicados

space!

Função Com Bloco Aglomerado

```
1  fn x_plus_y() -> u32 {
2    let x = 5; let y = 11;
3
4    x + y
5  }
```

Consulta para Desaglomerar Bloco

```
1  (function_item
2    body: (block (__) @item . (__) @next)
3    (#space! "\n" 2 @item @next))
```

Função Com Bloco Desaglomerado

```
1  fn x_plus_y() -> u32 {
2    let x = 5;
3    let y = 11;
4
5    x + y
6  }
```


Os Predicados

indent!

Função Com Bloco Aglomerado

```
1  fn x_plus_y() -> u32 {
2    let x = 5; let y = 11;
3
4    x + y
5  }
```

Consulta para Desaglomerar Bloco Com Indentação

```
1  (#set! indent-style " ")
2
3  (function_item
4    body: (block (__) @item . (__) @next)
5    (#space! "\n" 2 @item @next)
6    (#set! @next indent-rule "+1")
7    (#indent! @next))
```

Função Com Bloco Desaglomerado e Indentação Apropriada

```
1  fn x_plus_y() -> u32 {
2    let x = 5;
3    let y = 11;
4
5    x + y
6  }
```

Os Predicados

indent-offset!

Função Compactada

```
1  fn foo() {bar() }
```

Consulta para Formatar uma Função Conforme o Estilo 1TBS

```
1  (#set! indent-style "    ")
2
3  ( (function_item
4    body: (block (__) @item "}" @close)) @fn
5    (#set! @item indent-rule "+1")
6    (#indent-offset! @close @fn)
7    (#indent! @item @close))
```

Função Formatada Conforme o Estilo 1TBS

```
1  fn foo() {
2      bar()
3  }
```

A Sincronização de Nós

Funções Aninhadas

```
1  fn foo() {fn bar() {baz()}}
```

Consulta para Formatar Funções Conforme uma Variante do *1TBS*

```
1  (#set! indent-style " ")
2
3  ( (function_item
4    body: (block (_) @item "}" @close)) @fn
5    (#set! @item indent-rule "+1")
6    (#indent-offset! @close @fn)
7    (#indent! @item @close))
```

Funções Aninhadas Conforme uma Variante do *1TBS*

```
1  fn foo() {
2    fn bar() {
3      baz()
4    }
5  }
```

Funções Aninhadas Mal Formatadas

```
1  fn foo() {
2    fn bar() {
3      baz()
4    }
```

Conclusão

- ▶ O objetivo geral deste trabalho foi atendido — desenvolver o *Witch Cooking*
- ▶ O software desenvolvido atendeu aos objetivos de
 - ▶ Abranger uma gama de linguagens de programação
 - ▶ Proporcionar a formatação personalizada via a linguagem de consulta do *Tree-Sitter*
 - ▶ Não funcional para cenários realistas




Contribuições

- ▶ O *Witch Cooking*
 - ▶ Formata diversas linguagens através do *Tree-Sitter*
 - ▶ Proporciona predicados para dirigir a formatação
 - ▶ Oferece maior controle ao usuário
 - ▶ Exige conhecimento
 - ▶ Da sintaxe em questão
 - ▶ Da linguagem de consulta

Trabalhos Futuros

- ▶ Desenvolver um algoritmo dedicado à sincronização de nós
- ▶ Aprimorar o predicado `indent !`
- ▶ Dinamizar a formatação
 - ▶ Formatar conforme *CPL*
 - ▶ Cálculo layout otimizado
- ▶ Disponibilizar diretrizes de formatação

Referências

-  OPPEN, Derek C. Prettyprinting. **ACM Transactions on Programming Languages and Systems**, v. 2, n. 4, p. 465–483, out. 1980. DOI: 10.1145/357114.357115.
-  TREE-SITTER: a parsing system for programming tools. Versão 0.20.8. Tree-Sitter. Disponível em: <<https://tree-sitter.github.io/>>. Acesso em: 6 abr. 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.