Projeto AV3 - Python para Rust

Aspectos teóricos computação

Universidade de Fortaleza - Unifor

Ciência da Computação - CCT

Autor: Thiago Fernandes

Orientador: Samuel Barrocas

1. Introdução

O projeto desenvolvido envolve a criação de um transpilador que converte código fonte de Python para Rust. Inicialmente, realizamos o processo de traduzir o código de uma linguagem de programação para outra, mantendo a funcionalidade original. Dessa forma, a presente implementação é capaz de mapear o parsing, gerar sua sintaxe e gramática para realizar a "troca" e traduzir uma parsing simples em python para rust mantendo as mesmas funcionalidades. O projeto usa a biblioteca parsing para gerar os parsings iniciais, como também para a construção da AST e geração do código em rust.

2. Tecnologias Utilizadas

Linguagens de Programação: Python e Rust.

Bibliotecas Python:

- • pyparsing: Utilizada para criar o parser que transforma o código Python em tokens.
- • ast (Abstract Syntax Tree): Para representar a estrutura sintática do código de forma hierárquica (Que no caso, blocos iniciais para a conversão são inicializados antes para mapear em blocos os parsers de python e com o code_generator, traduzir em rust).

3. Estrutura do Projeto

O projeto está organizado da seguinte forma:

• src: Diretório principal contendo o código fonte.

• custom_ast: Contém os arquivos relacionados à AST e à geração de código.

- • transpiler_ast.py: Define as classes para os nós da AST.
- • python_parser.py: Contém a lógica para o parsing do código Python e construção da AST.
- • code_generator.py: Contém a lógica para gerar o código Rust a partir da AST.

• main.py: Arquivo principal para executar o processo de transpilação.

• test: Contém exemplos de código Python e Rust para testes.

• requirements.txt: Lista as dependências do projeto.

4. Desenvolvimento e Complexidade

4.1 Parsing e Construção da AST

O primeiro passo no processo de transpilação é o parsing do código Python. A gramática inclui elementos básicos como inteiros, identificadores e literais de string, além de construções mais complexas como expressões e blocos de código.

A construção da AST envolve a conversão dos tokens gerados pelo parser em nós de uma árvore. Cada tipo de construção de linguagem (atribuição, impressão, condição, função) é representado por um nó específico na AST. O código abaixo exemplifica a definição de um nó de atribuição:

```
def parse_assignment(tokens):
 expression = " ".join(str(x) for x in tokens[2]) if isinstance(tokens[2], (list, ParseResults)) else str(tokens[2])
 return AssignmentNode(tokens[0], ExpressionNode(expression))
```

4.2 Geração de Código

Uma vez construída a AST, o próximo passo é a geração de código Rust. Isso é feito atravessando a AST e gerando o código equivalente em Rust para cada nó. Por exemplo, um nó de atribuição em Python é convertido para uma declaração de variável em Rust:

```
def generate_rust(node):
    if isinstance(node, AssignmentNode):
        return f"let {node.identifier} = {generate_rust(node.expression)};"
```

4.3 Complexidade

A complexidade deste projeto pode ser analisada em termos de:

- • Parsing: A complexidade de parsing com pyparsing é geralmente linear em relação ao tamanho do código de entrada, O(n), onde n é o número de tokens no código.
- • Construção da AST: A construção da AST envolve a criação de uma estrutura de árvore, o que também é O(n) em termos de complexidade, assumindo que a árvore não é muito desequilibrada.
- • Geração de Código: A geração de código a partir da AST envolve atravessar a árvore, o que é uma operação linear, O(n), em relação ao número de nós na árvore.

5. Dificuldades Encontradas

Durante o desenvolvimento do projeto, diversas dificuldades foram encontradas e superadas:

- • Definição da Gramática: Criar uma gramática precisa para Python que fosse suficientemente completa para os exemplos de teste, mas não tão complexa que fosse difícil de implementar e debugar.
- • Conversão de Tokens para AST: Garantir que os tokens fossem corretamente convertidos em nós da AST e que a estrutura da AST representasse fielmente a estrutura do código Python.
- • Geração de Código: Assegurar que a geração de código Rust fosse correta e eficiente, mantendo a lógica do código original.

6. Conclusão

O projeto de transpilação de Python para Rust foi uma empreitada desafiadora que envolveu o uso de diversas tecnologias e conceitos avançados de compilação e parsing. O resultado é uma ferramenta capaz de converter código Python para Rust, permitindo aproveitar as vantagens de ambas as linguagens. Este projeto não apenas reforçou o entendimento de parsing e geração de código, mas também destacou a importância da organização e modularização em projetos de software complexos.

7. Referências

- • pyparsing Documentation: https://pyparsing-docs.readthedocs.io/en/latest/
- • Python AST Module: https://docs.python.org/3/library/ast.html
- • Rust Programming Language: https://www.rust-lang.org/

8. Código Fonte

https://github.com/ThiagoDev202/transpiler-project