UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE - UERN
FACULDADE DE CIÊNCIAS EXATAS E NATURAIS - FANAT
DOCENTE: Sebastião Emidio Alves Filho.
DISCENTES: Allany dos Santos Rodrigues, Fabio Bentes Tavares de Melo Junior, João Victor da Costa Gomes, Reinaldo Rogger Santos da Silva, Victor Manoel Soares da Silva.
DISCIPLINA: Compiladores e Paradigmas de Programação

TRABALHO TRANSPILADOR - 3° AVALIAÇÃO

15 de Dezembro de 2024
MOSSORÓ - RN

Sumário

1. INTRODUÇÃO

1.1. Introdução

Este trabalho propõe a implementação de um transpilador que converte programas escritos em Python para Ruby, com o objetivo de facilitar a migração e a integração de sistemas desenvolvidos nessas linguagens. A escolha dessas duas linguagens se justifica pela sua ampla utilização e características complementares, além do potencial de otimização e aprendizagem que esse processo oferece. A implementação do transpilador envolve o estudo de técnicas de análise sintática, semântica e otimização de código, além de proporcionar uma oportunidade para explorar as diferenças e semelhanças entre essas linguagens de alto nível. O trabalho busca, assim, contribuir para a compreensão e aplicação de conceitos fundamentais na área de compiladores e transpiladores, ao mesmo tempo que oferece uma ferramenta prática para desenvolvedores que precisam interagir com Python e Ruby em seus projetos.

1.2. JUSTIFICATIVA

A criação de um transpilador que converte programas escritos em Python para Ruby é relevante e importante por diversas razões. Primeiramente, ambas as linguagens são amplamente utilizadas em diferentes domínios, e facilitar a conversão entre elas pode aumentar a flexibilidade no desenvolvimento de software, permitindo que códigos escritos em Python sejam rapidamente adaptados para ambientes onde Ruby seja a linguagem preferida, como em muitas plataformas web e frameworks como Ruby on Rails.

Com um transpilador, é possível migrar ou integrar sistemas existentes de forma mais ágil, sem a necessidade de reescrever grandes trechos de código, economizando tempo e esforço. Isso proporciona uma maneira eficiente de adaptar softwares entre ambientes onde Python e Ruby são usados de forma intercambiável.

Além disso, a implementação de um transpilador oferece uma valiosa oportunidade para aprofundar o entendimento sobre as diferenças entre linguagens de alto nível e como seus ecossistemas operam. O processo envolve técnicas de análise sintática e semântica, além de otimização de código, que são fundamentais no estudo de compiladores e transpiladores. Dessa forma, o desenvolvimento dessa ferramenta contribui para o avanço do conhecimento na área de engenharia de software e pode ser um recurso importante para desenvolvedores que precisam trabalhar com ambas as linguagens, ampliando suas habilidades e capacidade de adaptação a diferentes tecnologias.

2. A LINGUAGEM DE ORIGEM

2.1. Descrição da Linguagem de Origem

Python é uma linguagem de programação de alto nível, criada por Guido van Rossum e lançada em 1991. Seu principal objetivo é ser simples e legível, com uma sintaxe clara que facilita a escrita e leitura do código. A linguagem é interpretada, o que permite um desenvolvimento mais rápido, e adota tipagem dinâmica, dispensando a declaração explícita de tipos de variáveis.

Python é multiparadigma, ou seja, suporta programação orientada a objetos, funcional e imperativa. Sua rica biblioteca padrão e um vasto ecossistema de bibliotecas externas cobrem áreas como desenvolvimento web, ciência de dados e automação. Além disso, é

uma linguagem portátil, funcionando em diferentes sistemas operacionais sem a necessidade de ajustes no código.

A sintaxe de Python é organizada por indentação, o que elimina a necessidade de chaves ou palavras-chave para definir blocos de código, promovendo a clareza. Apesar de ser fácil de usar, Python pode ser mais lento que linguagens compiladas como C, e a tipagem dinâmica pode gerar problemas de desempenho em algumas situações. A implementação padrão, o CPython, também possui o GIL (Global Interpreter Lock), que limita a execução concorrente de threads.

3. A LINGUAGEM DE DESTINO

3.1. Descrição da Linguagem de Destino

Ruby é uma linguagem de programação de alto nível, dinâmica e orientada a objetos, criada por Yukihiro Matsumoto em 1995. Seu design foca na simplicidade e produtividade, com uma sintaxe natural que permite aos desenvolvedores escrever código de maneira concisa e elegante.

Em Ruby, quase tudo é um objeto, incluindo tipos primitivos como números e strings. A linguagem é dinâmica, com tipagem determinada em tempo de execução, o que proporciona flexibilidade, mas pode gerar erros difíceis de identificar antecipadamente.

Ruby é multiparadigma, suportando programação orientada a objetos, funcional e imperativa. Oferece recursos como blocos de código, lambdas e métodos integrados para manipulação eficiente de coleções como arrays, hashes e strings.

A linguagem possui uma rica biblioteca padrão e uma comunidade ativa, com muitas gemas (bibliotecas de terceiros) que expandem suas capacidades. Ruby é famosa por seu framework **Ruby on Rails**, amplamente utilizado no desenvolvimento de aplicações web devido à sua filosofia de "convenção sobre configuração", que acelera a criação de aplicativos.

4. ANÁLISE LÉXICA

4.1. Tokens Suportados

```
# tokens.py
tokens = (
    'IDENTIFIER', 'EQUALS', 'NUMBER', 'FLOAT', 'STRING', 'COMMENT',
    'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN',
    'TRUE', 'FALSE', 'AND', 'OR', 'PRINT',
    'IF', 'ELSE', 'FOR', 'WHILE', 'DEF', 'COLON', 'IN',
    'COMMA',
    'GT', 'LT', 'GE', 'LE', 'EQ', 'NE',
    'PLUS_EQUALS', 'MINUS_EQUALS', 'TIMES_EQUALS', 'DIVIDE_EQUALS'

)
```

4.2. Literais e Tipos de Dados Suportados

Tipos de Dados Reconhecidos

Os tipos de dados básicos reconhecidos pelo lexer e parser são:

1. **Numéricos:**
    - NUMBER: números inteiros (e.g., 42).
    - FLOAT: números de ponto flutuante (e.g., 3.14).
2. **Booleanos:**
    - TRUE e FALSE: reconhecidos e convertidos para true e false no código Ruby.
3. **Strings:**
    - STRING: suporta strings com aspas simples ou duplas, incluindo caracteres escapados (definição detalhada no lexer.py).

Literais Suportados

Os operadores e palavras reservadas definidos em tokens.py e utilizados no lexer.py incluem:

- **Operadores aritméticos:** PLUS, MINUS, TIMES, DIVIDE.
- **Operadores de comparação:** GT (>), LT(<),GE(>=),LE(<=),EQ(==),NE` (!=).
- **Operadores compostos:** PLUS_EQUALS (+=), MINUS_EQUALS (-=), etc.
- **Delimitadores:** COLON (:), COMMA (,).

4.3. Palavras Reservadas

As palavras reservadas, listadas em tokens.py, não podem ser usadas como identificadores. Elas incluem:

- **Controle de fluxo:**
    - IF, ELSE, WHILE, FOR.
- **Funções:**
    - DEF, IN.
- **Operadores lógicos:**
    - AND, OR.
- **Exibição:**
    - PRINT.

Essas palavras são tratadas como tokens no lexer.py com funções específicas que garantem que sejam corretamente interpretadas.

5. GRAMÁTICA UTILIZADA NO RECONHECIMENTO DOS COMANDOS

5.1. Definição da Gramática

A gramática é definida no parser.py usando o módulo ply.yacc. Aqui estão as principais definições:

- **Regras Sintáticas Gerais:**
  - A gramática começa com a produção inicial program, que pode conter várias statements (declarações).
  - Cada statement pode ser uma atribuição, um laço, um condicional, ou mesmo um comentário.
- **Regras para Operadores:**
  - Precedência é definida no topo do arquivo (precedence), especificando a ordem de resolução de operadores, com suporte para operadores unários (UMINUS).
- **Exemplos de Produções:**

Atribuição:
statement : IDENTIFIER EQUALS expression

  - **Exemplo: x = 42 → x = 42 (em Ruby).**

Operadores compostos:
statement : IDENTIFIER PLUS_EQUALS expression

  - **Exemplo: x += 1 → x += 1.**

Estruturas de controle:
statement : WHILE expression COLON statements
Exemplo:
```
while x < 10:
   x += 1
```
Se torna:
```
while x < 10
  x += 1
end
```
Condicionais:
statement : IF expression COLON statements ELSE COLON statements
Exemplo:
```
if x > 10:
   print(x)
else:
   print(0)
```

Se torna:
```
if x > 10
  puts x
else
  puts 0
end
```
5.2. Exemplos de Regras Sintáticas

A gramática cobre uma ampla gama de construções Python, traduzindo-as para Ruby. Exemplos incluem:

Funções:

statement : DEF IDENTIFIER LPAREN params RPAREN COLON statements

Exemplo:

```
def soma(a, b):
    return a + b
```

Se torna:

```
def soma(a, b)
  return a + b
end
```

Laços for:

statement : FOR IDENTIFIER IN expression COLON statements

Exemplo:

```
for i in range(5):
    print(i)
```

Se torna:

```
for i in range(5)
  puts i
end
```