UNIFACS - UNIVERSIDADE SALVADOR

CIÊNCIA DA COMPUTAÇÃO

Projeto A3

TEORIA DA COMPUTAÇÃO E COMPILADORES

Ferramenta de Tradução

Enrico Falcão Peixoto de Souza- 1272216661

Ignácio Nunes Piva - 1272212685

Lucio Santiago Marques de Queiroz - 12722123508

Salvador
2023

1. Introdução

No universo da computação, os compiladores desempenham um papel fundamental ao permitir que linguagens de programação, criadas para atender necessidades humanas específicas, sejam traduzidas para linguagens compreendidas por máquinas. O desenvolvimento de um compilador vai além de apenas um exercício técnico: é uma experiência que combina teoria e prática, abrangendo áreas como linguagens formais, sintaxe, semântica e teoria da computação. Este trabalho de conclusão de curso na disciplina de Teoria dos Compiladores aborda a construção de uma ferramenta de tradução capaz de converter um programa, desenvolvido em uma linguagem projetada pelo grupo, para uma linguagem de programação amplamente conhecida, onde neste caso será Python.

O desafio envolve o desenvolvimento de uma linguagem-fonte, com regras sintáticas e semânticas definidas pelo grupo, e a criação de um compilador que traduza os programas escritos nessa linguagem para uma linguagem-alvo. A escolha do destino da tradução não é apenas técnica, mas estratégica, considerando as características e a popularidade da linguagem-alvo para execução prática. Este projeto busca não apenas implementar uma solução funcional, mas também explorar os conceitos teóricos que fundamentam o design e a construção de compiladores, promovendo uma compreensão aprofundada do processo de tradução e das decisões envolvidas em cada etapa.

Ao longo deste trabalho, discutiremos o design da linguagem-fonte, os aspectos técnicos e teóricos que guiaram a implementação do compilador e os desafios enfrentados no processo de tradução. Além disso, serão abordados os impactos do projeto na compreensão da complexidade dos compiladores e sua importância no campo da ciência da computação.

2. Documentação

Esse código implementa um tradutor de uma linguagem customizada para Python, utilizando as bibliotecas **Ply** (para análise léxica e sintática) e **Tkinter** (para interface gráfica). O tradutor realiza a análise do código escrito na linguagem customizada, gera uma árvore sintática e converte o código em uma versão equivalente em Python. Vamos examinar o código em detalhes, incluindo a análise léxica (lexer), sintática (parser) e a equivalência do código gerado.

Análise Léxica (Lexer)

A **análise léxica** transforma o código fonte em uma sequência de tokens (elementos lexicais). Cada token representa uma unidade básica da linguagem, como palavras-chave, operadores e identificadores. O lexer usa expressões regulares para identificar os padrões no código.

Definição de tokens

No início do código, é criado um conjunto de **tokens** que a linguagem reconhece, que inclui palavras-chave como `for`, `se`, `leia`, `faixa`, operadores aritméticos (`+`, `-`, `*`, `/`), comparações (`==`, `!=`, `<`, `>`, `<=`, `>=`), delimitadores de estrutura (como parênteses `()`, chaves `{}`, ponto e vírgula `;`), entre outros.

As expressões regulares para cada token são definidas da seguinte forma:

- **Operadores aritméticos**: Definem os tokens para adição (`+`), subtração (`-`), multiplicação (`*`), divisão (`/`), etc.
- **Delimitadores**: Definem tokens para parênteses (`(`, `)`), chaves (`{`, `}`), ponto e vírgula (`;`), etc.
- **Variáveis e valores**: Define tokens para variáveis (`[a-zA-Z_][a-zA-Z0-9_]*`), inteiros (`\d+`), decimais (`\d+\.\d+`), e frases (strings) entre aspas (`"..."` ou `'...'`).

O lexer identifica os tokens no código de entrada e os armazena para serem analisados pelo parser. Aqui, o lexer também implementa o tratamento de **erros** para caracteres ilegais.

Exemplo de entrada:

```
escreva("Digite A: ");
a = leia();
```

O lexer gera tokens como:

```
ESCREVA: 'escreva'
FRASE: 'Digite A: '
PONTOE_VIRGULA: ';'
VARIAVEL: 'a'
IGUAL: '='
LEIA: 'leia'
PARENTESES_ESQUERDO: '('
PARENTESES_DIREITO: ')'
PONTOE_VIRGULA: ';'
```

Análise Sintática (Parser)

A **análise sintática** toma os tokens gerados pela análise léxica e organiza esses tokens em uma **árvore sintática**. O parser define as **regras de produção** que descrevem como os tokens podem ser combinados para formar expressões e sentenças válidas na linguagem.

Definição de regras sintáticas

O parser define várias regras que descrevem como construir expressões e comandos, como variáveis, laços (`for`, `while`), estruturas condicionais (`if-else`), e operações aritméticas.

Algumas regras de produção incluem:

Declaração e atribuição de variáveis:

```
expression : VARIAVEL IGUAL valor FINALEXPRESSAO
expression : VARIAVEL IGUAL expression FINALEXPRESSAO
```

- Essas regras descrevem como uma variável pode ser atribuída a um valor literal ou a outra expressão.

- **Laços de repetição** (`for` e `while`):
    - `for i entre faixa (5)` seria transformado no equivalente Python `for i in range(5):`.
    - `enquanto a <= 10:` seria transformado no equivalente Python `while a <= 10:`.
- **Estruturas condicionais** (`se` / `senao`):
    - A regra para `se a < b: ... senao: ...` transforma o código para a estrutura Python `if a < b: ... else: ...`.
- **Operações aritméticas**:
    - A regra para adição (`+`) entre variáveis seria transformada em uma operação no Python, como `a + b`.

Produção de árvore sintática

O parser, ao processar as expressões, gera uma árvore sintática com tuplas, representando as operações a serem realizadas. Por exemplo, a operação `a = b + c` seria representada como:

```
('atribuir', 'a', ('+', 'b', 'c'))
```

Onde o primeiro valor `atribuir` indica que é uma operação de atribuição e os valores subsequentes indicam a variável e a expressão a ser atribuída.

Tradução para Python

Após a análise sintática, o código é traduzido para Python através da função `traduzir_para_python`. A função recursiva percorre a árvore sintática gerada e converte os comandos da linguagem customizada para sintaxe Python.

Exemplos de conversões:

- **Atribuição**: O comando `a = b + c` é traduzido para:

```
a = b + c
```

- **Laço `for`**: O laço `for i entre faixa (5)` é traduzido para:

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

- **Estrutura `if-else`**: A estrutura `se a < b: ... senao: ...` é traduzida para:

```
if a < b:
    ...
else:
    ...
```

- **Comando `escreva`**: O comando `escreva("Texto")` é traduzido para:

```
print("Texto")
```

- **Comando `leia`**: O comando `leia()` é traduzido para:

```
input()
```

- **Exemplo de Código Traduzido**

Dado o código na linguagem customizada:

```
escreva("Digite A: ");
a = leia();
escreva("Digite B: ");
b = leia();

se a < b:
    c = a + b;
senao:
    c = a - b;

enquanto a <= 10:
    escreva("Menor que 10");

for i entre faixa (5):
    escreva("teste");

escreva("C é igual a");
escreva(c);

d = a + b;
```

A tradução para Python seria:

```
print("Digite A: ")
a = input()
print("Digite B: ")
b = input()

if a < b:
    c = a + b
else:
    c = a - b

while a <= 10:
    print("Menor que 10")

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

Integração com Interface Gráfica (Tkinter)

A interface gráfica é construída com **Tkinter**, permitindo que o usuário insira código na linguagem customizada em um campo de texto. O código é então analisado e traduzido para Python, com a saída sendo exibida em outro campo de texto.

- **Entrada de código**: O usuário digita o código da linguagem customizada.
- **Saída**: O código traduzido para Python é exibido na interface.

Erros de sintaxe são exibidos em janelas separadas com mensagens de erro.

Conclusão

Esse código fornece uma implementação completa de um tradutor de uma linguagem customizada para Python. Ele realiza análise léxica e sintática do código fonte, gera uma árvore sintática e traduz a árvore para uma versão equivalente em Python. Além disso, a interface gráfica permite interação com o usuário para testar o tradutor de maneira prática.