Universidade Federal da Fronteira Sul
Campus Chapecó

Discente: Jacquet Leme e Loude Djema Sime

Doscente: Braulio Adriano de Mello

Trabalho: Construção de Compiladores: Implementação das etapas de compilação.

Le 18/04/2024

1. Introdução

Esse artigo tem como objetivo descrever o desenvolvimento e funcionamento de uma aplicação que executa as etapas de verificação léxica e sintática em um compilador. As etapas citadas são feitas de acordo com arquivos contendo uma gramática regular para formação de variáveis e identificações de tokens e um arquivo contendo o código que será analisado para verificar se pertence a gramatica livre de contexto escolhida para a parte semântica.

2. Referencial Teórico

Para melhor compreensão da descrição do desenvolvimento da aplicação desse artigo se faz necessário o entendimento de alguns conceitos.

- Alfabeto: conjunto finito de símbolos ou caracteres.
- Palavra: cadeia de caracteres.
- Token: palavra reservada.
- Concatenação: junção de dois ou mais símbolos.

Um Autômato Finito é um reconhecedor de Gramaticas Regulares, gramaticas essas que constituem o grau mais simples de linguagem. Segundo Menezes, um Autômato Finito é composto por um conjunto de possíveis estados, um alfabeto, uma função de transição, um estado inicial um conjunto de estados finais [Menezes 2009].

Podemos classificar o Autômato Finito como “Determinístico” ou “Não Determinístico”.

- Autômato Finito Determinístico (AFD): Em cada um dos estados há somente uma transição para outro estado em cada um dos símbolos.
- Autômato Finito Não Determinístico (AFND): Existe pelo menos um estado em que há mais de uma transição para diferentes estados em algum símbolo.

As etapas de compilação podem ser dividas em analise léxica,

analise sintática, analise semântica, geração de código intermediário e, por fim, otimização de código.

Já na segunda etapa, a analise sintática, é verificado se o código está gramaticalmente correto.

Nesta etapa, é utilizado um Analisador LR para fazer a checagem sintática. Os analisadores LR (Left to right) são analisadores redutores eficientes que leem a sentença em análise da esquerda para a direita e produzem uma derivação ao mais a direita ao reverso [Alencar and Sirineo 2001].

3. Desenvolvimento

A linguagem escolhida para a implementação do projeto prático foi Python, sendo o principal motivo para a escolha a facilidade que a linguagem permite manipularmos cadeias de caracteres e listas de itens. Para manipular os arquivos XML foi usado a importação do "xml.etree.ElementTree".

A aplicação é dividida em funções, cada uma com seu papel, indo desde gerar o AFD, até toda a lógica de redução, empilhamento, salto ou aceitação na analise sintática.

3.1. Garantindo o funcionamento do código

Existem algumas restrições e observações que devem ser levadas em conta para garantir o bom funcionamento do código.

Para garantir o bom funcionamento da geração do AFD, no arquivo "syntrax_GR.txt" devemos nos atentar a algumas regras:`

- Apenas uma gramatica ou regra por linha no arquivo fonte.
- O caractere reservado para conjunto vazio é o Epsilon.
- O estado inicial da gramatica será definido pelo "Start Symbol".
- A letra "X" como regra de gramatica é reservada para representar o Estado de erro.
- As produções da regra devem ser separadas por uma barra vertical.
- Os estados gerados a partir de tokens são representados por números.
- O caractere reservado para conjunto vazio e o "ε".

Já no arquivo "input_code.txt", que contém o código que será analisado, é importante que todos os tokens e variáveis fiquem separados por um espaço em branco.

A tabela de Parser LALR deve ser gerada no software GoldParser em formato XML, e deve ficar dentro da pasta "materials", com o nome do arquivo sendo "parser.xml".

3.2. Criação da Tabela de ParseLALR

A criação da tabela de ParseLALR e feito com o auxílio do software GoldParser. Dada uma determinada gramatica, a aplicação em questão gera a tabela de parser, uma vez que a tabela foi gerada podemos exporta-la para um arquivo XML.

Figura 2. Função Scanner

```
def scan(self, input_code):
    tokens = self.tokenize(input_code)
    symbol_table = []
    line_number = 1

    for token in tokens:
        if token.strip() or token == '\n':  #Considerar '\n' como u
            if token == '\n':
                line_number += 1
                continue  # Pula a adição de uma entrada de quebra
            token_type = self.get_word_type(token)
            symbol_table.append({
                'Line': line_number,
                'Identifier': token,
                'Label': token_type
            })
    return symbol_table
```

4. Testes e Resultados

Os testes foram feitos com a seguinte gramatica:

```
input_code = """
_a = b
_a = _i
_a = 5
print ( _d )
while _a = _b { _a = 2 }
if _a +:= 2 { print ( _a ) } else { _a = _a + 1 }
_a = 2 ++ 2
"""
```

Figura 4. Gramatica Livre de Contexto (usada para etapa sintática)

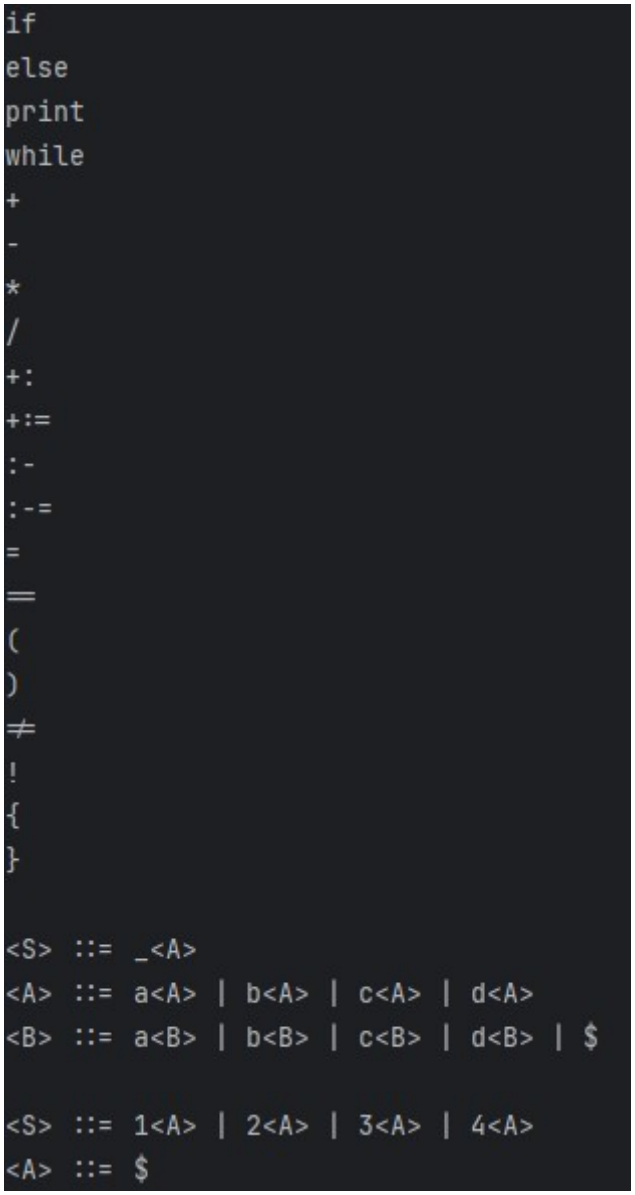

Figura 5. Gramatica Regular.

Para executarmos algum teste basta alterar o que está no arquivo “code_input.txt”, se tudo ocorrer bem ao final da execução receberemos uma mensagem informando que a sentença foi aceita.

Com o código “_a = 2 + 2” por exemplo, obtemos sucesso.
Retornando “OK -> ACCEPTED”

Já com o código “_a = 2 ++ 2”, obtém-se um erro, já que o símbolo “++” não é reconhecido pela gramática.

5. Conclusão

O desenvolvimento de uma aplicação para colocar em prática o conhecimento adquirido ao longo do semestre foi um desafio que ajudou a lapidar a compreensão sobre os

assuntos da disciplina, principalmente por ser necessário combinar tais conhecimentos com técnicas de programação abordadas em semestres anteriores.

Mesmo a aplicação não contemplando todas as etapas do projeto, foi possível compreender os processos por trás das etapas de compilação, suas dependências entre si e maneiras de aborda-las.

A implementação das etapas restantes fica como uma perspectiva para continuidade do trabalho.

Referencias

Alencar, P. A. M. d. and Sirineo, T. S. (2001). *Implementacao de linguagens de programacao: Compiladores*. Sagra-Luzzatto.

Menezes, P. B. (2009). *Linguagens Formais e Automatos: Volume 3 da S^ erie Livros´ Didaticos Inform´ atica UFRGS´* . Bookman Editora.