Compilador Na Linguagem TPP

Gabriel David Sacca[1]

[1] Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 87301 – Campo Mourão – PR – Brazil

{gabrielsacca}@alunos.utfpr.edu.br

Abstract. *This report is divided into 6 major and main sessions, the first of which introduces the proposed theme, the second presents a language that will be used and what are the advantages and advantages of using this specific language, a third discussion session on a lexical part of the project, its importance in the whole, showing its code, how it was created and what rates it returns, a fourth session presents a syntactic part, also shows its code and its visualizations and organization, a fifth part presents the semantics and its organizations and lastly we have a code generation executed with a combination of all the other parts. At the end of the project, a summary will also be displayed, considering final considerations, portraying as difficulties and resources to understand and create a compiler.*

Resumo. *Este relatório esta dividido em 6 grandes e principais sessões, a primeira delas da uma introdução ao tema proposto, a segunda apresenta a linguagem que será utilizada e quais são as vantagens e desvantagens de se utilizar esta linguagem em especifico, a terceira sessão fala sobre a parte léxica do projeto, sua importância no todo, mostrando seu código, como ele foi criado e quais as saídas que ele irá retornar, a quarta sessão apresenta a parte sintática, também mostrando seu código e suas saídas e organização, a quinta parte apresenta a semântica e suas organizações e por ultimo temos a geração de código realizada com a combinação de todas as outras partes. Ao fim do projeto terá também um resumo colocando as considerações finais, retratando as dificuldades e facilidades de se entender e construir um compilador.*

1. Introdução

Este relatório consiste na descrição da implementação de um trabalho feito na aula de Compiladores ministrado pelo professor Rogério Aparecido Gonçalves. O trabalho foi divido em 4 partes, sendo elas as analises léxicas, sintáticas, semânticas e a parte de geração de código. Por fins educacionais a linguagem TPP foi utilizada, e no final deste projeto teremos um compilador que consegue entender um código em TPP e gerar um executável para o mesmo, apresentando erros ou falhas se houver.

2. Linguagem TPP

Esta linguagem foi desenvolvida com fins educacionais para a disciplina de um compilador, e nela temos algumas das estruturas mais conhecidas na areá, como laços, funções, declarações de tipos de dados e afins, todas essas estruturas são mais simples e menos complexas para que facilitem no desenvolver do trabalho. Neste capitulo irá ser apresentado todos estes tipos de estruturas, todos os possíveis Tokens e seus respectivos significados para a linguagem:

- **Estrutura de Condição:** Começando pelas condições, temos nesta linguagem o Token SE, SENAO e ENTAO que respectivamente funcionam como o if, elif e else da linguagem pyton;
- **Laços de Repetição:** Os laços são representados nessa linguagem com os Tokens REPITA e ATE, eles funcionam como o do while da linguagem C, python e outras, e todo código dentro deste laço é executado enquanto a condição do até não seja quebrada;
- **Funções:** Já as funções nesta linguagem podem ou não possuir retorno de dados, parâmetros e todas possuem escopos separados de variáveis;
- **Tipos de dados:** Por ultimo a linguagem possui quatro tipos distintos de dados inteiro, flutuante, vetores e matrizes.

Com estas simples estruturas a linguagem é perfeita para o aprendizado da criação de um compilador, pois se fosse utilizar linguagens conhecidas como C, Python, Java e outras, a complexidade das linguagens tornaria inviável o desenvolvimento do compilador em apenas 1 semestre.

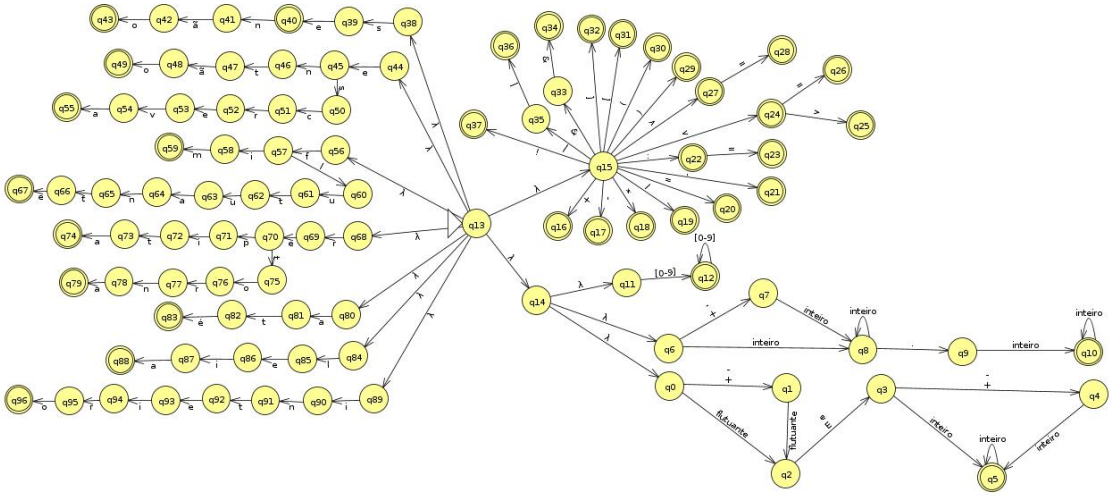

Figura 1. Imagem do Automato e todos os seus possíveis caminhos

3. Léxica

Um compilador é dividido em 4 principais etapas, como já citado na sessão 1, e nesta sessão ira ser mostrado como foi implementar a parte léxica do trabalho mostrando e exemplificando o automato necessário para gerar a expressão regular do código, a ferramenta para criação de um analisador léxico do Python denominada Python, como utilizalo e quais suas vantagens e desvantagens, e por ultimo alguns exemplos de entradas e saídas de código que o nosso programa léxico gera com diferentes entradas.

3.1. Especificação dos autômatos

Um automato nada mais é do que uma maquina que dada uma entrada, irá ou não te retornar uma saída com aceito ou não aceito pela linguagem. Neste relatório pode ser observado na figura 1, o automato que gera exatamente as expressões regulares que são necessários para entender todos os Tokens da linguagem. Para a criação da maquina eu

a dividi em 3 principais sessões, sendo a primeira de reconhecimento de palavras reservadas, a segunda para reconhecer Tokens de símbolos da linguagem e a terceira para reconhecer as possíveis formas de representações numéricas.

Começando pelo reconhecimento dos Tokens de símbolos eles são bem simples e intuitivos, pois cada caminho pode te levar para 1 ou 2 possíveis caracteres e a soma dos dois forma uma palavra reconhecida pela linguagem. Um exemplo de Token reconhecido pela linguagem foi , neste automato temos um estado inicial que leva para um dos e um estado final que leva para o segundo , fazendo assim com que a linguagem considere um único um carácter invalido, mas uma sequencia de dois é aceita pela mesma.

Para reconhecer os tipos de dados que na linguagem se limitam a inteiros, flutuantes e números representados com notação cientifica, foram necessários 3 outros sub autômatos:

```python
import sys
import ply.lex as lex

reserved = {
    'se': 'SE',
    'então': 'ENTAO',
    'senão': 'SENAO',
    'fim': 'FIM',
    'repita': 'REPITA',
    'flutuante': 'FLUTUANTE',
    'retorna': 'RETORNA',
    'até': 'ATE',
    'leia': 'LEIA',
    'escreva': 'ESCREVA',
    'inteiro': 'INTEIRO'
}

tokens = [
    'SOMA',
    'SUBTRACAO',
    'MULTIPLICACAO',
    'DIVISAO',
    'IGUALDADE',
    'VIRGULA',
    'ATRIBUICAO',
    'MENOR',
    'MAIOR',
    'MENOR_IGUAL',
    'MAIOR_IGUAL',
    'ABRE_PARENTESES',
    'FECHA_PARENTESES',
    'DOIS_PONTOS',
    'ABRE_COLCHETE',
    'FECHA_COLCHETE',
    'E_LOGICO',
    'OU_LOGICO',
    'NEGACAO',
    'DIFERENTE',
    'NOTACAO_CIENTIFICA',
    'ID',
    'NUMERO_INTEIRO',
    'NUMERO_FLUTUANTE'
] + list(reserved.values())
```

Figura 2. Imagem da declaração das palavras reservadas e dos tokens

- **Inteiro:** Para reconhecer inteiros foi utilizado dois estados, um que reconhece um primeiro simbolo de 0 até 9 e outro que fica repetindo um estado final enquanto

houverem números de 0 até 9, se não houver nada além de números o automato irá parar e reconhecer que aquela sequencia de símbolos é um número inteiro;

- **Flutuante:** Os flutuantes são um pouco mais complexos, o primeiro estado da maquina pode levar para dois outros caminhos, um para antes de chegar nos números inteiros reconhecer sinais de positivo e negativo, e outro sem os sinais caso a pessoa não queira identifica-los, após passar por esta etapa de decisão você chega em um estado onde os inteiros podem ser repetidos e assim que encontrar um ponto o automato vai para o próximo estado, onde novamente aceita uma sequencia de inteiros até acabar a palavra, e se nenhum carácter diferente de inteiro for encontrado ela será aceita como flutuante pela linguagem;
- **Notação cientifica:** E por ultimo a notação cientifica, que é a mais complexa dos 3 reconhecedores de tipos de dados. No inicio ela trabalha de forma semelhante aos números flutuantes, pois necessita de dois caminhos, um com sinal e outro sem sinal, para reconhecer um número flutuante. Se este número for reconhecido e logo em seguida os caracteres E ou e forem detectados, o automato vai para um novo estado intermediário, e se novamente um flutuante com ou sem sinal forem encontrados, os símbolos são aceitos como notação cientifica pela linguagem.

```
45  t_ignore = ' \t\n'
46  t_SOMA = r'\+'
47  t_SUBTRACAO = r'-'
48  t_MULTIPLICACAO = r'\*'
49  t_DIVISAO = r'/'
50  t_IGUALDADE = r'='
51  t_VIRGULA = r','
52  t_ATRIBUICAO = r':='
53  t_MENOR = r'<'
54  t_MAIOR = r'>'
55  t_MENOR_IGUAL = r'<='
56  t_MAIOR_IGUAL = r'>='
57  t_ABRE_PARENTESES = r'\('
58  t_FECHA_PARENTESES = r'\)'
59  t_DOIS_PONTOS = r':'
60  t_ABRE_COLCHETE = r'\['
61  t_FECHA_COLCHETE= r'\]'
62  t_E_LOGICO = r'&&'
63  t_OU_LOGICO = r'\|\|'
64  t_NEGACAO = r'!'
65  t_DIFERENTE = r'<>'
66  t_NOTACAO_CIENTIFICA = r'((\+|-)?[\d+]+\.?[\d+]*)(e|E)(\+|-)?[\d+]+'
67  t_NUMERO_FLUTUANTE = r'(\+|-)?[\d+]+\.[\d+]*'
68  t_NUMERO_INTEIRO = r'(-|\+)?\d+'
69
70  def t_ID(t):
71      r'[A-Za-z_][\w_]*'
72      t.type = reserved.get(t.value, 'ID')
73      return t
74
75  def t_error(t):
76      print('Illegal character: %s' % t.value[0])
77      t.lexer.skip(1)
78
79  def t_COMMENT(t):
80      r'\{((.|\n)*?)\}'
```

Figura 3. Imagem da declaração das palavras reservadas e dos Tokens

Já para reconhecer as palavras reservadas encontradas nesta linguagem é simples,

basta criar estados intermediários para as letras da palavra desejada, sendo o ultimo estado a ultima letra dela, e por consequência a palavra já será aceita pela linguagem. Todavia existem palavras que se inciam com as mesmas letras como exemplo o Token SE e o Token SENAO, estas palavras irão começar pelo mesmo caminho até o SE e depois vão se dividir em dois caminhos diferentes, um que aceita a palavra SE e outro que verifica se ainda possui mais caracteres para validar.

3.2. PLY

O PLY [Beazley] é uma ferramenta do Python para auxiliar na criação de um analisador léxico. Ele possui ferramentas que auxiliam na criação e desenvolvimento do código, e além de explicar cada ferramenta do PLY, nesta sessão ira ser mostrado também o código criado para gerar a linguagem léxica no geral.

a primeira etapa do PLY consiste em declarar todas as palavras reservadas e os Tokens como pode ser visto na figura 2, elas são divididas em 2 grandes estruturas, a primeira delas possui o nome de reserved e nela você deve colocar todas as palavras reservadas da linguagem, sendo a primeira a palavra em si, e logo em seguida o Token que essa palavra vai gerar. A segunda estrutura com nome de Tokens possui todos os possíveis Tokens que essa linguagem pode gerar.

```
82  arquivo = open(sys.argv[1])
83  lex.lex()
84
85  lex.input(arquivo.read())
86  while True:
87      tok =  lex.token()
88      if not tok: break
89      if(len(sys.argv) >= 3):
90          if(sys.argv[2] == "C"):
91              print(tok.type,",", tok.value)
92          elif(sys.argv[2] == "S"):
93              print(tok.type)
94      else:
95          print(tok.type,",", tok.value)
```

Figura 4. Imagem da implementação das saídas

já na figura 3 podemos observar as expressões regulares que serão necessários para reconhecer os Tokens previamente listados, então cada um dos Tokens possui uma expressão regular para si, que deve conter informações de como é o comportamento padrão do mesmo na linguagem escolhida. Além disso temos também algumas expressões

padrões do PLY, a primeira delas é o **t_ignore**, e todos os caracteres que estiverem dentro de sua expressão serão ignorados para a analise léxica da linguagem, neste código os espaços em branco, tabulações e quebras de linha foram ignorados como pode ser analisado na linha 45. já a partir da linha 75 temos as funções para exceções como o **t_error**, que é chamado caso algum caractere não seja reconhecido pela linguagem e retorna uma mensagem para o usuário, e a ultima função é o **_COMMENT** que encontra e ignora qualquer comentário.

```
1   inteiro: n
2   flutuante: a[10]
3
4   inteiro fatorial(inteiro: n)
5       inteiro: fat
6       se n > 0 então {não calcula se n > 0}
7           fat := 1
8           repita
9               fat := fat * n
10              n := n - 1
11          até n = 0
12          retorna(fat) {retorna o valor do fatorial de n}
13      senão
14          retorna(0)
15      fim
16  fim
17
18  inteiro principal()
19      leia(n)
20      escreva(fatorial(n))
21      retorna(0)
22  fim
23
24
```

Figura 5. Exemplo de código em TPP

e a última parte do código são as saídas, você pode escolher vários tipos de saídas possíveis, e neste projeto eu escolhi dois tipos, saída apenas com os Tokens encontrados e saídas com Tokens e os valores destes. Para uma saída com Tokens e valores é necessário digitar C como ultimo argumento de entrada, e sem valores digite S.

3.3. Exemplos de entrada e saída

Para testar se tudo esta funcionando perfeitamente alguns exemplos de código em TPP foram passados para teste, um destes exemplos foi o fat, um código que calcula o fatorial de um número que pode ser observado na figura 5. Se a parte léxica estiver funcionando corretamente, então uma lista de Tokens correspondentes aos apresentados deve ser impressa no terminal como os da figura 6, que mostra um trecho inicial da saída esperada, com os Tokens correspondentes.

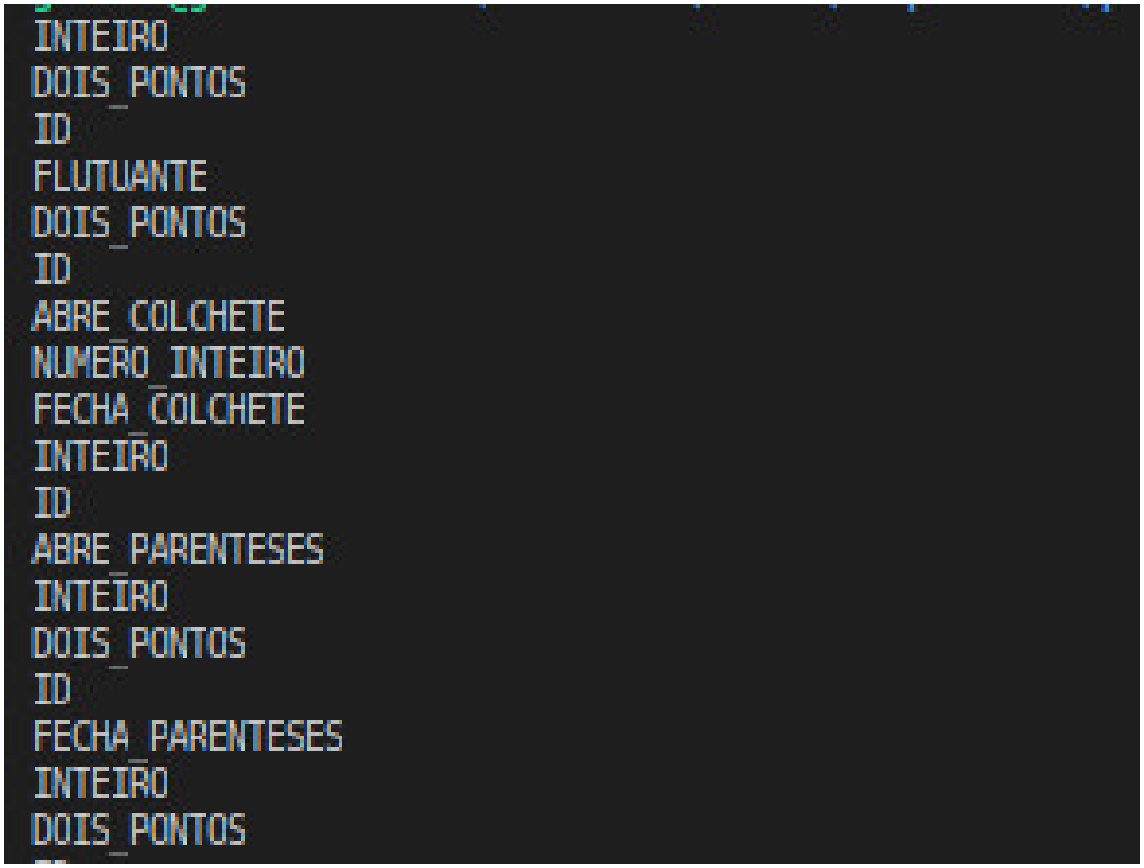

Figura 6. Exemplo de código em TPP

4. Resultados

Desta forma, se torna possível perceber que a parte léxica é muito importante para o funcionamento total de um compilador, ela gera Tokens que serão utilizados nas demais áreas, todavia não é a única, e no decorrer do projeto as próximas áreas irão utilizar o que já foi desenvolvido nesta.

Referências

Beazley, D. M. Ply (python lex-yacc) documentation.