Projeto de Implementação de um Compilador para a Linguagem TPP: Análise Léxica (Trabalho – 1ª parte)

Autor: Alexandre Aparecido Scrocaro Junior
R.A.: 2135485

Professor: Rogério Aparecido Gonçalves
Universidade Tecnológica Federal do Paraná (UTFPR)

Análise Léxica

Resumo

A primeira parte do trabalho consiste na implementação de um analisador léxico - também chamado de *scanner* ou sistema de varredura - para a linguagem TPP, para tanto, foi utilizada a documentação da linguagem disponibilizada pelo professor. A linguagem/ferramenta utilizada foi Python/PLY, além de fazer uso de expressões regulares para analisar os *tokens*.

Especificação da linguagem de programação TPP

- Tipos básicos de dados suportados: **inteiro** e **flutuante**
- Suporte a arranjos uni e bidimensionais **(*arrays*)**
  - Exemplos:
    - tipo: identificador[dim]
    - tipo: identificador[dim][dim]
- Variáveis locais e globais devem ter um dos tipos especificados
- Tipos de funções podem ser omitidos (quando omitidos viram um procedimento e um tipo *void* é devolvido explicitamente
- Linguagem quase fortemente tipificada: nem todos os erros são especificados mas sempre deve ocorrer avisos
- Operadores aritméticos: +, -, * e /
- Operadores lógicos: e (&&), ou (||) e não (!)
- Operador de atribuição: recebe (:=).
- Operadores de comparação: maior (>), maior igual (>=), menor (<), menor igual (<=), igual (=).

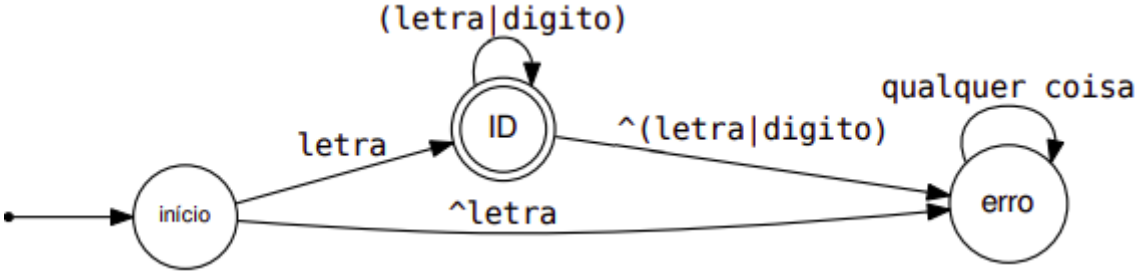
Especificação formal dos autômatos para a formação de cada classe de token da linguagem

Autômato de identificador:

Primeiramente, as classes mais simples:

- letra = [a-zA-Z]+
- digito = [0-9]+

Agora, podemos criar um autômato de um identificador da seguinte forma:

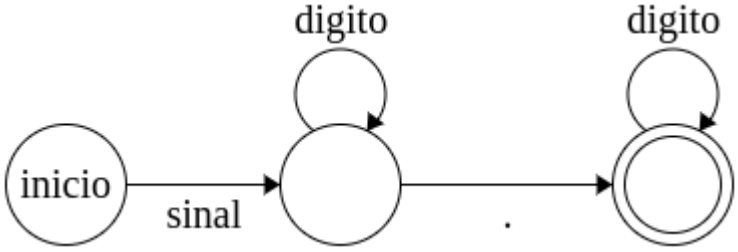


Autômato de número flutuante:

Primeiramente, as classes mais simples:

- sinal = [+-]* (o sinal pode ter nada, é repassado com épsilon)
- digito = [0-9]+

Agora, podemos criar um autômato de um número flutuante da seguinte forma:



Detalhes da implementação da varredura na LP e ferramenta (e/ou bibliotecas) escolhidas pelo projetista

O sistema de varredura, ou analisador léxico, é a fase do compilador na qual se lê o código-fonte como um arquivo de caracteres e o separa em um conjunto de tokens, os quais são reconhecidos através das expressões regulares. A implementação do analisador será explicada a seguir.

Em primeiro lugar, define-se os *tokens*, palavras reservadas e operadores que serão utilizados para reconhecer as facilidades da linguagem TPP, assim como suas expressões regulares. Após isso, foram definidas as funções para reconhecer as classes (ID - que requer atenção especial para não coincidir com nenhuma palavra reservada, notação científica, número de ponto flutuante, número inteiro, comentários, novas linhas e colunas).

Além disso, também são reconhecidos erros de caracteres especiais que a linguagem não contém (como $ e ç) e é mostrada sua posição no código (linha e coluna).

Por último, o código retorna a lista de tokens correspondentes ao arquivo escolhido por quem executa o comando para iniciá-lo, percorrendo todo o código TPP do arquivo selecionado.

A ferramenta utilizada foi o Python PLY, que é uma implementação do lex e yacc e utiliza LR-parsing - que é razoavelmente eficiente. O lex possui a facilidade na tokenização de uma *string* de entrada, por exemplo:

***String* de entrada:**
x = 3 + 42 * (s - t)\

***String* após tokenização:**
'x','=', '3', '+', '42', '*', '(', 's', '-', 't', ')'

***Tokens* obtidos:**
ID, IGUAL, INTEIRO, MAIS, INTEIRO, VEZES, ABRE_PARENTESE, ID, MENOS, ID, FECHA_PARENTESE

Exemplos de saída do sistema de varredura (lista de tokens) para exemplos de entrada (código fonte)

Código para verificar valor 10:

ENTRADA:

```
inteiro principal()
  inteiro: a
  a:=1
  repita
    se a=10
      escreva("valor 10")
    fim
    a++
  ate a=10

  reutn(0)
fim
```

SAIDA:

INTEIRO
ID
ABRE_PARENTESE
FECHA_PARENTESE
INTEIRO
DOIS_PONTOS
ID
ID
ATRIBUICAO
NUM_INTEIRO
REPITA
SE
ID
IGUAL
NUM_INTEIRO
ESCREVA
ABRE_PARENTESE
[6,12]: Caracter inválido ' " '
ID
NUM_INTEIRO
[6,21]: Caracter inválido ' " '
FECHA_PARENTESE
FIM
ID
MAIS
MAIS
ID
ID
IGUAL
NUM_INTEIRO
ID
ABRE_PARENTESE
NUM_INTEIRO
FECHA_PARENTESE
FIM

Código de produto escalar:

ENTRADA:

```
inteiro escalar(inteiro : array1[], inteiro : array2[], inteiro : n)
  inteiro : produto
  produto := 0
  inteiro : i
  i := 0
  se n > 0 entao
    repita
      produto := produto + array1[i] * array2[i]
      i := i + 1
    ate i = n
  retorna (produto)
fim
```

SAIDA:

INTEIRO
ID
ABRE_PARENTESE
INTEIRO
DOIS_PONTOS
ID
ABRE_COLCHETE
FECHA_COLCHETE
VIRGULA
INTEIRO
DOIS_PONTOS
ID
ABRE_COLCHETE
FECHA_COLCHETE
VIRGULA
INTEIRO
DOIS_PONTOS
ID
FECHA_PARENTESE
INTEIRO
DOIS_PONTOS
ID
ID
ATRIBUICAO
NUM_INTEIRO
INTEIRO
DOIS_PONTOS
ID
ID
ATRIBUICAO
NUM_INTEIRO
SE
ID
MAIOR
NUM_INTEIRO
ID
REPITA
ID
ATRIBUICAO
ID
MAIS
ID

ABRE_COLCHETE
ID
FECHA_COLCHETE
MULTIPLICACAO
ID
ABRE_COLCHETE
ID
FECHA_COLCHETE
ID
ATRIBUICAO
ID
MAIS
NUM_INTEIRO
ID
ID
IGUAL
ID
RETORNA
ABRE_PARENTESE
ID
FECHA_PARENTESE
FIM

Implemente uma função que imprima a lista de tokens, não utilize a saída padrão da ferramenta de implementação de Analisadores Léxicos

Para tanto, basta tokenizar o arquivo de entrada e imprimir todos os *tokens* obtidos, que é feito na função a seguir.

```
def imprimir_tokens():
    while True:
        tok = lexer.token()
        if not tok:
            break      # No more input
        # print(tok)

        # print para mostrar o token
        print(tok.type)
```