Relatório do Trabalho I - Construção de analisador léxico

Ricardo Parizotto[1], **Rafael Hengen Ribeiro**[1]

[1]Universidade Federal da Fronteira Sul
Chapecó - Santa Catarina - Brasil

ricardo.dparizotto@gmail.com

rafaelhr.ribeiro@gmail.com

1. Resumo

Neste trabalho é apresentado a implementação de um algoritmo capaz de ler uma tabela com um autômato reconhecedor de tokens e fazer a análise léxica de um código fonte através das regras de produção da tabela. Para mostrar sua eficiência, criamos uma linguagem de programação hipotética, "blue", e construímos o autômato reconhecedor.

2. Introdução

A análise léxica é o primeiro passo da compilação. Um analisador léxico tem como função ler os caracteres de um código fonte e identificar os *tokens* que pertencem a linguagem. A maior parte dos erros não são identificados nesta fase da compilação, porém ela é importante para gerar símbolos para a próxima etapa: a análise sintática.

3. Análise léxica

De acordo com [Johnson and Zelenski 2008, p. 1], a análise léxica ou *scanning* é o processo em que os caracteres que compõem o código-fonte são lidos da esquerda para a direita e agrupados em *tokens*.

3.1. Tokens

Os *tokens* compõem o conjunto de todos os símbolos pertencentes a uma linguagem. Incluem: palavras reservadas, símbolos especiais, constantes e identificadores.

4. O analisador

O analisador carrega um arquivo de entrada **.csv** com o autômato reconhecedor de tokens da linguagem, onde para todo estado, ao ler um símbolo, há instrução de como ir para outro estado.

O estado inicial é identificado por $' > S'$ e todos os estados finais são identificados pelo caractere $'\#'$ vindo antes do nome do estado. O caractere separador para cada transição é o caractere $(espaço)$. Por isso, o estado de $erro$ não está implícito, devemos indicar na tabela quando a transição leva a um estado de erro.

O analisador espera como entrada um arquivo do código fonte da linguagem $blue$.

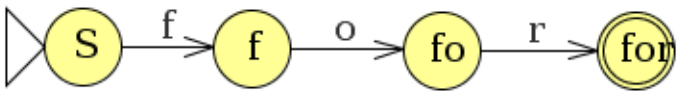

Figura 1. Exemplo de autômato de reconhecimento da linguagem para o token 'for'

4.1. Descrição da implementação

A implementação do *Scanner* léxico foi realizada na linguagem C++. Para representarmos o autômatos utilizamos funções da biblioteca **map**, garantindo maior flexibilidade e fácil adaptação à outras linguagens.

O algoritmo para reconhecer os tokens é muito simples. Para cada caractere lido do arquivo fonte é conferido se ele não é separador e está em um estado final. Caso isso ocorrer, o estado atual reconheceu um token da linguagem e este é inserido na fita de saída. Caso contrário, é analisado se o símbolo lido não leva a um estado de erro e, por fim, se nenhuma dos testes realizados for verdadeiro, o estado atual é agora atualizado para o estado que o caractere lido leva.

O arquivo de saída é o nome do arquivo de entrada concatenado com ".lextbl".

5. A linguagem 'blue'

5.1. tokens da linguagem

A linguagem hipotética criada para a apresentação do trabalho, possui as seguintes palavras reservadas:

- def
- input
- if
- end
- else
- while
- for
- in
- goto
- cont
- break
- ret

Os identificadores da linguagem podem ser qualquer sequência de caracteres minúsculos e números naturais concatenados que seja diferente de toda palavra reservada da linguagem.

O conjunto de símbolos especiais da linguagem é formado pelo conjunto $\{=, *, -, +, /, :, >, <\}$.

As constantes são números naturais (0,...,9).

5.2. Sintaxe

A linguagem pode ser definida pela seguinte GLC:

```
>S ::= def <fun> end

*<fun>::=   <id>=<exp><fun>
*<fun>::=   if <exp> <fun> end if <fun>
*<fun>::=   while <exp> : <fun> end while <fun>
*<fun>::=   for <id> in <id> : <fun> end for <fun>
*<fun>::=   for <exp> : <fun> end for <fun>
*<fun>::=   break <fun>
*<fun>::=   cont <fun>
*<fun>::=   <id>:<fun>
*<fun>::=   goto <id> <fun>
*<fun>::=   input <id> <fun>
*<fun>::=   ret <exp> <fun>
*<fun>::=   ε

cons ::= 0A | 1A | 2A | 3A | 4A | 5A | 6A | 7A | 8A | 9A

A:: = 0A | 1A | 2A | 3A | 4A | 5A | 6A | 7A | 8A | 9A | .A | ε

*<id> ::= a<id> | ... | z<id> | 0<id> | ... | 9<id>

*<exp> ::= cons<B> | <id><B> | (exp)

<B> ::= <sym><exp>

<sym>:: = * | + | - | / | > | < | == | >= | <=
```

6. Conclusão

A parte mais desafiadora foi organizar a tabela de maneira compreensível e adaptável para possíveis mudanças. O código foi projetado para facilitar, posteriormente, a implementação da análise sintática, que será outra classe que terá acesso à saída gerada pela classe do analisador léxico.

Referências

Johnson, M. and Zelenski, J. (2008). Lexical analysis. *Disponível em: http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/03-Lexical-Analysis.pdf*. Acesso em: 10/09/2015.