

Relatório do Trabalho I - Construção de analisador léxico

Ricardo Parizotto¹, Rafael Hengen Ribeiro¹

¹Universidade 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 ' $>$ ' 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*.

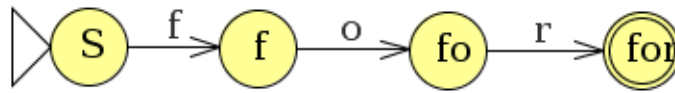


Figura 1. Exemplo de autômato de reconhecimento da linguagem para o token 'for'

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 nenhum 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*

A linguagem *blue* é uma linguagem hipotética. Conta com poucas palavras reservadas e possui tipagem dinâmica. Ela será definida formalmente a seguir.

5.1. tokens da linguagem

A linguagem *blue* 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:

$\langle S \rangle ::= \text{def } \langle \text{fun} \rangle \text{ end}$

$\begin{aligned} * \langle \text{fun} \rangle &::= \langle \text{id} \rangle = \langle \text{exp} \rangle \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{if } \langle \text{exp} \rangle \langle \text{fun} \rangle \text{ end if } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{while } \langle \text{exp} \rangle : \langle \text{fun} \rangle \text{ end while } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{for } \langle \text{id} \rangle \text{ in } \langle \text{id} \rangle : \langle \text{fun} \rangle \text{ end for } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{for } \langle \text{exp} \rangle : \langle \text{fun} \rangle \text{ end for } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{break } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{cont } \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \langle \text{id} \rangle : \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{goto } \langle \text{id} \rangle \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{input } \langle \text{id} \rangle \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \text{ret } \langle \text{exp} \rangle \langle \text{fun} \rangle \\ * \langle \text{fun} \rangle &::= \varepsilon \end{aligned}$

$\text{cons} ::= 0A \mid 1A \mid 2A \mid 3A \mid 4A \mid 5A \mid 6A \mid 7A \mid 8A \mid 9A$

$*A ::= 0A \mid 1A \mid 2A \mid 3A \mid 4A \mid 5A \mid 6A \mid 7A \mid 8A \mid 9A \mid .A \mid \varepsilon$

$*\langle \text{id} \rangle ::= a\langle \text{id} \rangle \mid \dots \mid z\langle \text{id} \rangle \mid 0\langle \text{id} \rangle \mid \dots \mid 9\langle \text{id} \rangle$

$*\langle \text{exp} \rangle ::= \text{cons} \langle B \rangle \mid \langle \text{id} \rangle \langle B \rangle \mid (\text{exp})$

$\langle B \rangle ::= \langle \text{sym} \rangle \langle \text{exp} \rangle \mid \varepsilon$

$\langle \text{sym} \rangle ::= * \mid + \mid - \mid / \mid > \mid < \mid == \mid >= \mid <=$

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.