Exercício 1 - Compiladores

Antônio Augusto Diniz Sousa - 201712040146

07/09/2020

1 Cite as fases constituintes de um compilador, indicando seus respectivos objetivos. Escreva com as suas palavras. Não é necessário citar as fases que são opcionais.

1. Análise Léxica
   É a única parte que entra em contato com o código fonte e que lê os caracteres identificando padrões. Seu objetivo principal é identificar padrões nos caracteres descritos no código fonte e formar *tokens* que serão enviados para a etapa de análise sintática.

2. Análise Sintática
   O objetivo desta fase é analisar se a ordem que os *tokens* foram recebidos do léxico é uma ordem válida (Uma ordem que está devidamente escrita na gramática da linguagem).

3. Análise Semântica
   O objetivo dessa fase é analisar se a semântica do código está certa. Por exemplo, se uma variável que está sendo usada já foi devidamente declarada, se a tipagem do valor que está sendo atribuído é igual à tipagem que foi declarada, etc. Em linhas gerais, ele olha para o contexto geral do código, ~~ao invés de olhar apenas para cada comando~~.

4. Gerador de código
   Após todas as fases de análise, garantindo que o código está dentro de todos os padrões da linguagem, esta fase é responsável por gerar o código visando uma certa máquina como objetivo, preparando para que ele execute devidamente no hardware/software escolhido.

2 No trecho de programa em C a seguir, indique a sequência de *tokens* que seria reconhecida pelo Analisador Léxico:

$$y = fatorial(n);$$

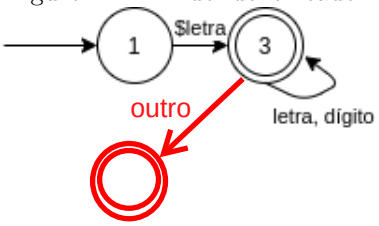
1. identificador, y - <**id, entrada na TS para o símbolo y**>
2. operador, = - <**equal_op**>
3. identificador, fatorial - <**id, entrada na TS para o símbolo fatorial**>
4. símbolo de pontuação, (- <**parenthesisOpen_sp**>
5. identificador, n - <**id, entrada na TS para o símbolo n**>
6. símbolo de pontuação,) - <**parenthesisClose_sp**>
7. símbolo de pontuação, ; - <**semicolon_sp**>

3 As descrições abaixo definem o padrão de formação de *tokens*. Para cada uma delas, mostre uma expressão regular e/ou um AFD (autômato finito determinístico) correspondente.

3.1 Identificadores devem iniciar com $, seguido de pelo menos uma letra, que pode vir seguida de uma sequência de letras e/ou dígitos.

$$digito \rightarrow [0-9]$$
$$letra \rightarrow [A-Za-z]$$
$$id \rightarrow \$letra(letra|digito)*$$

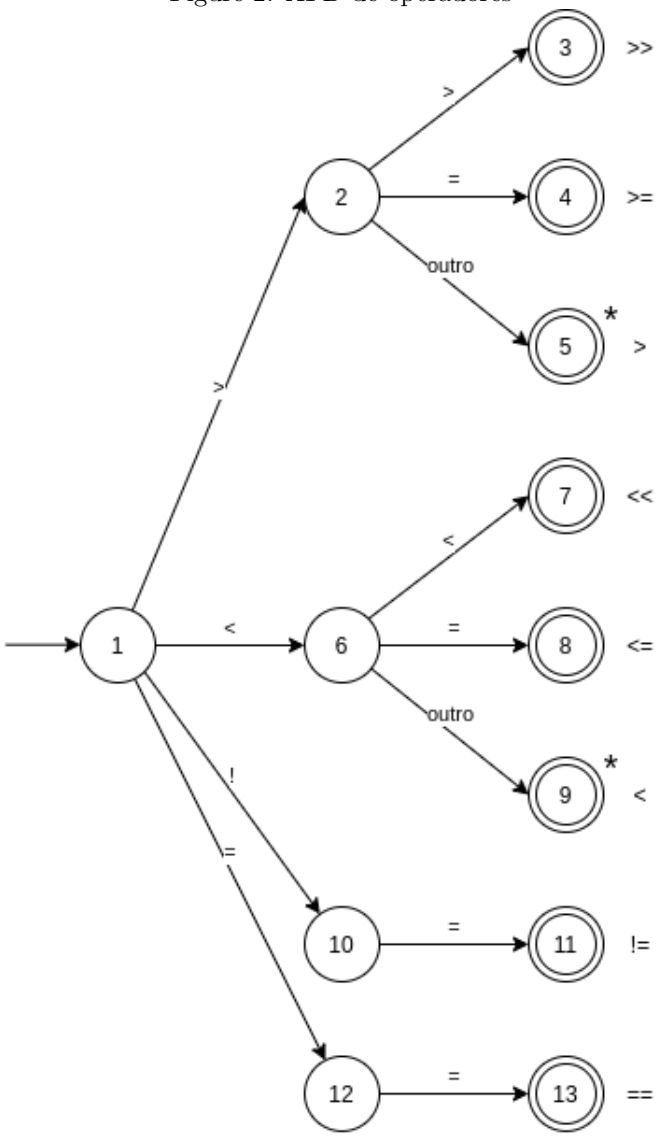
Figure 1: AFD de identificador



Poderia usa a aresta "outro". Nesse caso, o estado 3 não seria final.

3.2 Operadores: <, <<, >, >>, >=, <=, ! = e ==. Neste item não é necessário mostrar a expressão regular.

Figure 2: AFD de operadores



3.3 Constantes do tipo *float* são sequências de pelo menos um dígito seguida por um ponto (‘.’), seguido por pelo menos um dígito e terminadas com a letra ’f’.

$$digito \rightarrow [0-9]$$
$$float \rightarrow digito^{+}.digito^{+}f$$

Ou:

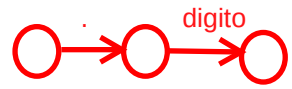
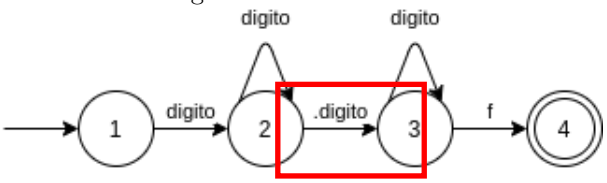


Figure 3: AFD de float



4 Mostre uma implementação (em C/C++, Java ou pseudocódigo) do método *public Token scan ()* que reconheça as construções a seguir.

- Operadores !, ! =, > e >>
- Identificadores conforme descrito no item 3.1

Algorithm 1: Pseudo Código - Analisador léxico resumido

```
Result: Retorna erro léxico ou o token identificado
estado = estadoInicial;
manterCaractere = false;
while Não fim do arquivo do
    if manterCaractere then
        manterCaractere = false;
    else
        caractere = nextChar();
    end
    switch estado do
        case 1 do
            switch caractere do
                case $ do estado = 2;
                case > do estado = 4;
                case ! do estado = 7;
                otherwise do Erro léxico;
            end
        case 2 do
            switch caractere do
                case letra do estado = 3;
                otherwise do Erro léxico;
            end
        case 3 do
            switch caractere do
                case letra ou digito do estado = 3;
                otherwise do Reconhece identificador;
            end
        case 4 do
            switch caractere do
                case > do Reconhece operador >>;
                otherwise do Reconhece operador >;
            end
        case 7 do
            switch caractere do
                case = do Reconhece operador ! =;
                otherwise do Reconhece operador !;
            end
        otherwise do Erro léxico;
    end
end
```

```
1 /* Considerar que sempre que temos "Reconhece ...", estado =
     estadoInicial.  Nos casos que ocorre dentro do otherwise,
     manterCaractere = true                                    */
```

A fim de facilitar o entendimento do pseudo-código, os estados utilizados são correspondentes ao AFD abaixo:

Figure 4: AFD correspondente a lógica implementada no pseudo-código acima