# Compiladores II

## Projeto de um Compilador

### *A linguagem LALG*[1]

Especificação da Sintaxe da Linguagem LALG

Glalg = {N,T,P,S}

N = {<programa>, <corpo>, <dc>, <comando>, <comandos>, <dc_v>, <mais_dc>, <dc_p>, <variaveis>, <tipo_var>, <mais_var>, <parametros>, <corpo_p>, <lista_par>, <mais_par>, <dc_loc>, <mais_dcloc>, <lista_arg>, <argumentos>, <pfalsa>, <condicao>, <expressao>, <relacao>, <termo>, <outros_termos>, <op_ad>, <op_un>, <fator>, <mais_fatores>, <op_mul>}

T = {ident, numero_int, numero_real, (, ), *, /, +, -, <>, >=, >, <, if, then, $, while, do, write, read, ;, else, begin, end, :, , , }

P:

<programa> ::= program ident <corpo> .

<corpo> ::= <dc> begin <comandos> end

<dc> ::= <dc_v> <mais_dc> | <dc_p> <mais_dc> | λ

<mais_dc> ::= ; <dc> | λ

<dc_v> ::= var <variaveis> : <tipo_var>

<tipo_var> ::= real | integer

<variaveis> ::= ident <mais_var>

<mais_var> ::= , <variaveis> | λ

<dc_p> ::= procedure ident <parametros> <corpo_p>

<parametros> ::= (<lista_par>) | λ

<lista_par> ::= <variaveis> : <tipo_var> <mais_par>

<mais_par> ::= ; <lista_par> | λ

<corpo_p> ::= <dc_loc> begin <comandos> end

<dc_loc> ::= <dc_v> <mais_dcloc> | λ

<mais_dcloc> ::= ; <dc_loc> | λ

---

[1] Usada na disciplina de Compiladores da Profa. Maria das Graças Volpe Nunes do ICMC-USP-São Carlos

<lista_arg> ::= (<argumentos>) | λ

<argumentos> ::= ident <mais_ident>

<mais_ident> ::= ; <argumentos> | λ

<pfalsa> ::= else <comandos> | λ

<comandos> ::= <comando> <mais_comandos>

<mais_comandos> ::= ; <comandos> | λ

<comando> ::= read (<variaveis>) |
write (<variaveis>) |
while <condicao> do <comandos> $ |
if <condicao> then <comandos> <pfalsa> $ |
ident <restoIdent>

<restoIdent> ::= := <expressao> |
<lista_arg>

<condicao> ::= <expressao> <relacao> <expressao>

<relacao> ::= = | <> | >= | <= | > | <

<expressao> ::= <termo> <outros_termos>

<op_un> ::= + | - | λ

<outros_termos> ::= <op_ad> <termo> <outros_termos> | λ

<op_ad> ::= + | -

<termo> ::= <op_un> <fator> <mais_fatores>

<mais_fatores> ::= <op_mul> <fator> <mais_fatores> | λ

<op_mul> ::= * | /

<fator> ::= ident | numero_int | numero_real | (<expressao>)

Exemplos de programas LALG

```
program nome1
/* exemplo 1 */
var a,a1,b : integer;
begin
read(a,b);
a1 := a + b;
while a1 > a do
  write(a1);
  a1 := a1 – 1
$;
if a <> b then write(a)$
end.
```

```
program nome2
{exemplo2}
var a: real;
var b:integer;
procedure nomep(x:real)
var a,c:integer;
begin
read(c,a);
if a < x + c then
   a:= c+x;
   write(a)
else c:=a+x
$
end
begin{programa principal}
read(b);
nomep(b)
end.
```

***Observações:***

- Comentários na LALG: entre { } ou /* */

- Identificadores e números são itens léxicos da forma:
    - ident: seqüência de letras e dígitos, começando por letra
    - número inteiro: seqüência de dígitos (0 a 9)
    - número real: sequencia de um ou mais digitos seguido de um ponto decimal seguido de um ou mais digitos.

- palavras reservadas – são os tokens usados para fins especificos, ou seja, que são previamente definidos na linguagem.

- simbolos simples e duplos – são aqueles também definidos na linguagem (<, $, >, etc. como exemplo de simples, e := como exemplo de duplo).

Classes de tokens da LALG:

C1. *Identificadores* - seqüência de letras e dígitos, começando por letra.

C2. *Palavras Reservadas* - tabela de símbolos Reservados - TSR.

C3. *Inteiros sem sinal* - seqüência de dígitos.

C4. *Símbolos Especiais* - seqüência de 1 ou 2 caracteres distintos de letra, dígito ou espaço, que pertença à TSR.

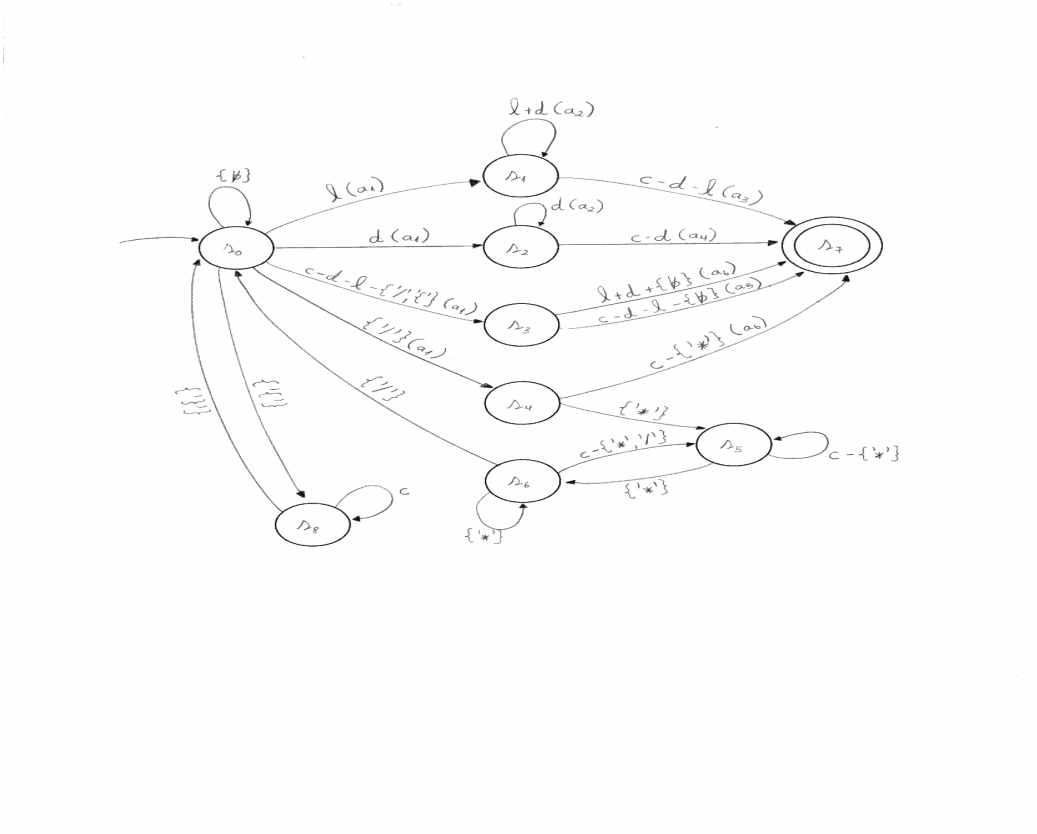
C5. *Comentários* - aparecem entre os delimitadores /* */ ou { }.

C6. Números Reais

Sejam:

c = todos os caracteres
l = {'A' ... 'Z'} ∪ {'a' ... 'z'} letra
d = {'0' ... '9'} dígito
l + d = letra ou dígito; c - l - d = caractere ≠ letra ou dígito

## Autômato Finito da LALG



Ações Semânticas
$a_1$ - inicializa cadeia com 1º. caractere
$a_2$ - concatena demais caracteres à cadeia
$a_3$ - reconhece palavras reservadas e identificadores
$a_4$ - reconhece número; calcula valor
$a_5$ - reconhece símbolos duplos
$a_6$ - reconhece símbolos especiais
$a_3$ - $a_6$ : "devolvem" caractere delimitador
$a_3$, $a_5$, $a_6$ : consultam Tabela de símbolos reservados (T.S.R.)

Erros Léxicos

- Ocorrem internamente ao scanner (analisador léxico). Exemplo: símbolo ilegal (@, etc.)

## Atividades a serem desenvolvidas:

1. Implementar um analisador léxico, baseado no autômato, para reconhecer/identificar os tokens da LALG. Ps: Modificar o autômato acima para lidar com números reais e também conforme sua necessidade.

Entrada: um programa escrito na linguagem LALG (programa fonte)
Saida:

a) uma lista contendo os pares <token, classe a qual pertence>, na sequência que aparecem no programa fonte;

b) já deve ter sido projetada a tabela de símbolos e nela inseridos os identificadores encontrados pelo léxico.

2. Implementar um analisador sintático descendente recursivo para a LALG
   Entrada: a lista de pares montada em 1.
   Saída:
   a) a análise sintática concluída ou erros, se houverem, nas respectivas linhas, indicando o/os ponto/s nos quais foram identificados problemas.
   b) a tabela de símbolos acrescida com os tipos dos identificadores, os escopos aos quais estão vinculados, a quantidade e os tipos de parâmetros vinculados a cada subrotina, além da ordem nas quais apareceram na declaração dos parâmetros. Obs: identificadores são variáveis, nomes de subrotinas e parâmetros. Você deverá diferencia-los de acordo com sua categoria na tabela de símbolos, e trata-los como tal. Por exemplo, nome de subrotina não tem tipo, uma vez que a LALG só trabalha com procedimentos; parâmetros, são sempre locais às suas subrotinas.

3. Implementar um verificador de tipos para a LALG, seguindo o esquema de tradução dirigida pela sintaxe.
   Na verificação de tipos deverá ser checado se tipos usados em expressões, atribuições são compatíveis entre si (compatíveis, aqui nos nosso caso é se são iguais). Essa verificação é estática e é feita através de consulta à tabela de símbolos procurando pelos tipos das variáveis armazenadas naquele escopo ou globais. Sempre lembrando que o escopo local tem prioridade sobre o global no caso de dupla declaração. Deverá ser feito também a checagem da quantidades e compatibilidades de tipos entre parâmetros formais e reais.

4. Apresentar a tabela de símbolos com todas as informações sobre identificadores.

5. Gerar um código intermediário baseado na MEPA para o programa fonte baseado na tradução dirigida pela sintaxe que você propôs.

6. Implementar a máquina de execução hipotética para o código MEPA gerado.