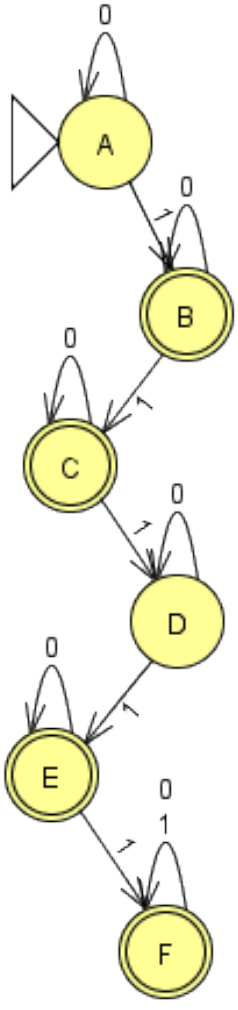


Teoria de Linguagens e Compiladores

Máquinas e Geração de Código

Luiz Eduardo da Silva

Universidade Federal de Alfenas

Agenda

Agenda

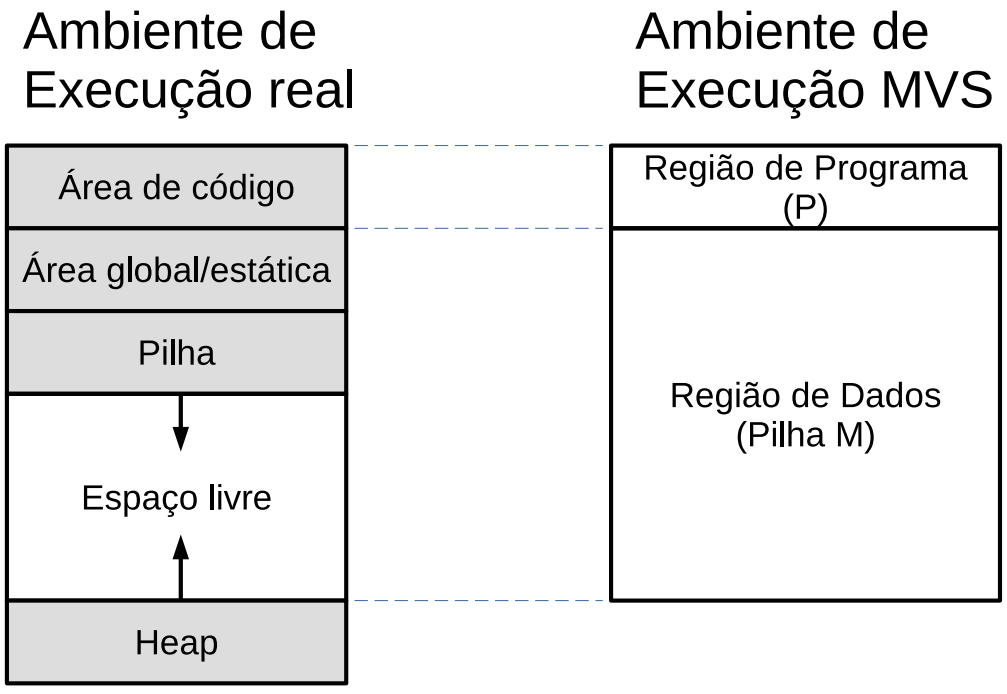
Máquina Virtual

- O objetivo principal do compilador para uma linguagem de programação é construir uma ferramenta de tradução que transforme os códigos numa linguagem de alto nível para instruções numa linguagem de máquina para, assim, possibilitar a sua execução.

- A arquitetura de uma máquina real é complexa, o que torna essa tarefa trabalhosa.
- Para simplificar, definiremos uma máquina virtual mais simples (MVS), mais conveniente para um compilador didático.

Características da MVS

- A Máquina Virtual Simples (MVS) é uma máquina baseada na Máquina de Execução Pascal (MEPA), proposta por Tomás Kowaltowski.
- A memória da MVS é composta de duas regiões:
  1. A região de programa P que conterá as instruções do programa em MVS que a máquina está executando.
  2. A região de pilha de dados M que conterá os valores manipulados pelas instruções MVS.



Características da MVS

As regiões de memória P e M funcionam como vetores com índices numerados de zero até uma tamanho máximo. Além disso, MVS tem três registradores. São eles:

1. O registrador i contém o endereço da próxima instrução a ser executada, $P[i]$. Esse registrador é incrementado automaticamente após a execução de cada instrução. Exceto para as instruções de desvio, que alteram de forma absoluta o valor de i.
2. O registrador s indicará o elemento no topo da pilha de dados, $M[s]$.
3. O registrador de base d que contém o endereço de base no qual a variável está inserida. Esse único registrador é suficiente para generalizar a forma de endereçamento das variáveis locais e globais no programa. Usaremos endereçamento indireto para variáveis locais e globais.

Instruções MVS

#	**Instrução**	**Micro-código**
1	CRCT k	$s \leftarrow s + 1$ $M[s] \leftarrow k$
2	CRVG n	$s \leftarrow s + 1$ $M[s] \leftarrow M[n]$
3	ARZG n	$M[n] \leftarrow M[s]$ $s \leftarrow s - 1$
4	SOMA	$M[s-1] \leftarrow M[s-1] + M[s]$ $s \leftarrow s - 1$
5	SUBT	$M[s-1] \leftarrow M[s-1] - M[s]$ $s \leftarrow s - 1$
6	MULT	$M[s-1] \leftarrow M[s-1] * M[s]$ $s \leftarrow s - 1$
7	DIVI	$M[s-1] \leftarrow M[s-1]/M[s]$ $s \leftarrow s - 1$
8	CMMA	$\underline{\text{SE}}\ M[s-1] > M[s]\ \underline{\text{ENTAO}}\ M[s-1] \leftarrow 1\ \underline{\text{SENAO}}\ M[s-1] \leftarrow 0;$ $s \leftarrow s - 1$
9	CMME	$\underline{\text{SE}}\ M[s-1] < M[s]\ \underline{\text{ENTAO}}\ M[s-1] \leftarrow 1\ \underline{\text{SENAO}}\ M[s-1] \leftarrow 0$ $s \leftarrow s - 1$
10	CMIG	$\underline{\text{SE}}\ M[s-1] = M[s]\ \underline{\text{ENTAO}}\ M[s-1] \leftarrow 1\ \underline{\text{SENAO}}\ M[s-1] \leftarrow 0$ $s \leftarrow s - 1$

Instruções MVS

#	**Instrução**	**Micro-código**
11	DISJ	SE $M[s-1]$ ou $M[s]$ ENTAO $M[s-1] \leftarrow 1$ SENAO $M[s-1] \leftarrow 0$ $s \leftarrow s-1$
12	CONJ	SE $M[s-1]$ e $M[s]$ ENTAO $M[s-1] \leftarrow 1$ SENAO $M[s-1] \leftarrow 0$ $s \leftarrow s-1$
13	NEGA	$M[s] \leftarrow 1 - M[s]$
14	DSVS p	$i \leftarrow p$
15	DSVF p	SE $M[s] = 0$ ENTAO $i \leftarrow p$ SENAO $i \leftarrow i+1$ $s \leftarrow s-1$
16	LEIA	$s \leftarrow s+1$ "$M[s] \leftarrow$ Entrada"
17	ESCR	"Escreve $M[s]$ " $s \leftarrow s-1$
18	NADA	"Não faz nada"
19	INPP	$s \leftarrow -1$ $i \leftarrow 1$ $D \leftarrow 0$;
20	FIMP	"Finaliza a execução"
21	AMEM n	$s \leftarrow s+n$

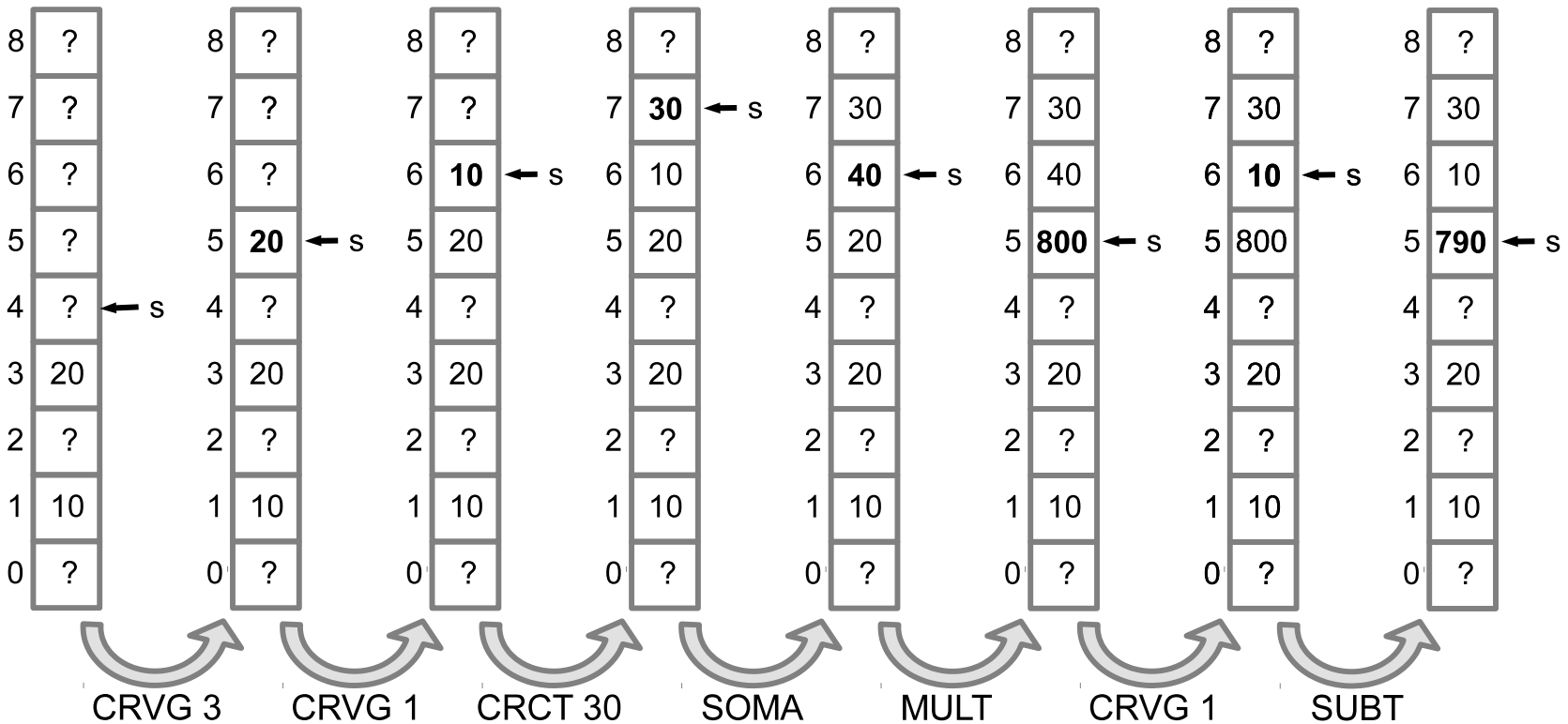
Tradução de Expressões

- Como a máquina MVS é baseada em pilha, as expressões em notação infixa (com o operação entre os operandos) da linguagem fonte devem ser traduzidas para uma sequência de instruções em NPR (Notação Polonesa Reversa, na qual a operação é colocada após os seus operandos).
- Considere a expressão B * (A + 30) - A e suponha que os endereços atribuídos pelo compilador as variáveis A e B são 1 e 3, respectivamente. Considere ainda que o valor da variável A é 10 e que o valor da variável B é 20. A tradução é:

```
1   CRVG  3
2   CRVG  1
3   CRCT  30
4   SOMA
5   MULT
6   CRVG  1
7   SUBT
```

Execução da Expressão

Expressão: B * (A + 30) - A, onde A = 10 e B = 20



Tradução da Atribuição

Instrução	Micro-código
ARZG n	$M[n] \leftarrow M[s]$ $s \leftarrow s - 1$

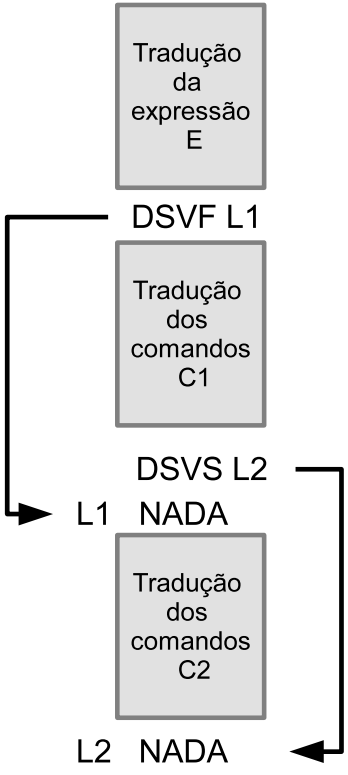
- Considere o comando de atribuição $\underline{\text{A} \leftarrow \text{A} + \text{B}}$, onde os endereços das variáveis A e B são 10 e 12 respectivamente. A tradução MVS para essa atribuição é:

```
1   CRVG  10
2   CRVG  12
3   SOMA
4   ARZG  10
```

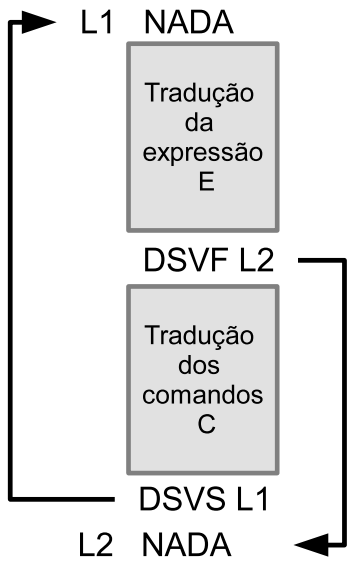
Tradução da Repetição e Seleção

Instrução	Micro-código
DSVS p	$i \leftarrow p$
DSVF p	SE $M[s] = 0$ ENTAO $i \leftarrow p$ SENAO $i \leftarrow i + 1$ $s \leftarrow s - 1$
NADA	"Não faz nada"

se E então C1 senao C2 fimse

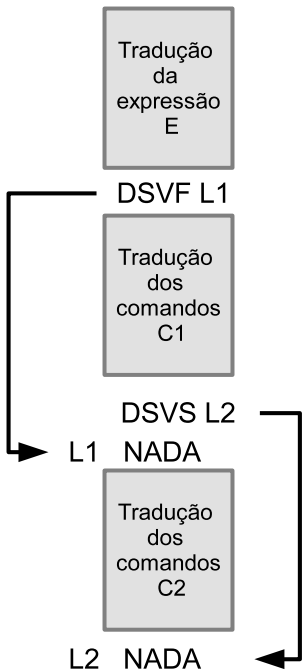


enquanto E faca C fimenquanto



Tradução da Seleção

```
1     A <- V
2     se A
3        entao A <- F
4        senao A <- V
5     fimse
```

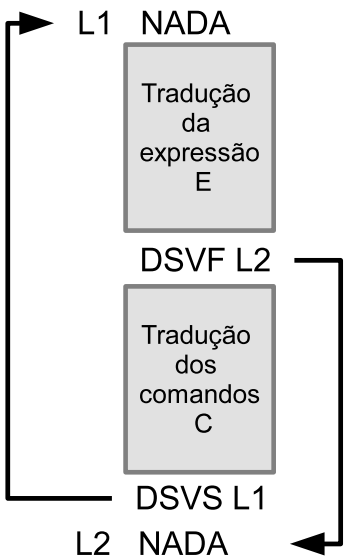


A tradução MVS é:

```
1          CRCT  1
2          ARZG  0
3          CRVG  0
4          DSVF  L1
5          CRCT  0
6          ARZG  0
7          DSVS  L2
8       L1 NADA
9          CRCT  1
10         ARZG  0
11      L2 NADA
```

Tradução da Repetição

```
1     x <- 1
2     enquanto x < 10 faca
3       x <- 2 * x
4     fimenquanto
```



A tradução MVS é:

```
1          CRCT  1
2          ARZG  0
3     L1   NADA
4          CRVG  0
5          CRCT  10
6          CMME
7          DSVF  L2
8          CRCT  2
9          CRVG  0
10         MULT
11         ARZG  0
12         DSVS  L1
13    L2   NADA
```

Tradução de Leitura e Escrita

Instrução	Micro-código
LEIA	$s \leftarrow s + 1$ "$M[s] \leftarrow$ Entrada"
ESCR	"Escreve $M[s]$ " $s \leftarrow s - 1$

```
1       leia A
2       leia B
3       escreva A + B
```

A tradução MVS é:

```
1       LEIA
2       ARZG  0
3       LEIA
4       ARZG  1
5       CRVG  0
6       CRVG  1
7       SOMA
8       ESCR
```

Tradução de Programa

Para tradução de um programa em linguagem Simples completo usaremos as seguintes instruções MVS:

Instrução	**Micro-código**
INPP	$s \leftarrow -1$ $i \leftarrow 1$ $d \leftarrow 0$
FIMP	"Finaliza a execução"
AMEM n	$s \leftarrow s + n$

Onde:

- INPP - instrução MVS que serve para colocar a máquina de execução numa configuração inicial;
- FIMP - instrução que finaliza a execução de um programa.
- AMEM - instrução que aloca memória para as variáveis globais do programa.

Exemplo de tradução

```
1     programa repete
2       inteiro i j
3     inicio
4       i <- 1
5       enquanto i < 10 faca
6         j <- 1
7         enquanto j < 10 faca
8           escreva i + j
9           j <- j + 1
10        fimenquanto
11        i <- i + 1
12      fimenquanto
13    fimprograma
```

```
1         INPP
2         AMEM 2
3         CRCT 1
4         ARZG 0
5     L1  NADA
6         CRVG 0
7         CRCT 10
8         CMME
9         DSVF L2
10        CRCT 1
11        ARZG 1
12    L3  NADA
13        CRVG 1
14        CRCT 10
15        CMME
16        DSVF L4
17        CRVG 0
18        CRVG 1
19        SOMA
20        ESCR
21        CRVG 1
22        CRCT 1
23        SOMA
24        ARZG 1
25        DSVS L3
26    L4  NADA
27        CRVG 0
28        CRCT 1
29        SOMA
30        ARZG 0
31        DSVS L1
32    L2  NADA
33        FIMP
```

Agenda

Introdução

- Podemos associar atributos aos símbolos da gramática (terminais e não terminais) para tradução.
- Nas **definições dirigidas por sintaxe**, estendemos as regras acrescentando ações semânticas que podem ser usados para especificar os valores dos atributos dos símbolos.
- Por exemplo(1):

Regra de Produção	Regra Semântica
$E \rightarrow E_1 + T$	$E.code = E1.code \|\| T.code \|\| '+'$

  - Essa produção tem dois não-terminais E e T (usamos E e E_1 para distinguir o símbolo E do lado esquerdo e direito da regra).
  - Todos os símbolos tem o atributo *code* do tipo cadeia. E a ação semântica que é executada na aplicação da regra produz a tradução pós-fixada da expressão ($||$ representa a concatenação de cadeias).

Introdução

- Podemos associar fragmentos de código como ação semântica nos **esquemas de tradução dirigida por sintaxe** como por exemplo (2):

Regra de Produção	Regra Semântica
$E \rightarrow E_1 + T$	{ printf ("+"); }

  - Por convenção os esquemas de tradução são delimitados por chaves.
  - As definições dirigidas por sintaxe (exemplo 1) são mais legíveis.
  - Os esquemas de tradução dirigidas por sintaxe (exemplo 2) são mais úteis para implementação.
- A abordagem mais geral da tradução dirigida por sintaxe consiste em construir a árvore de derivação, e então calcular os valores dos atributos dos nós da árvore (símbolos da gramática) durante a sua visitação.

Definição Dirigida por Sintaxe

- Uma definição dirigida por sintaxe é uma GLC acrescida de atributos e regras semânticas.
- Se X é um símbolo e a um dos seus atributos, usamos a notação $X.a$.
- Os atributos podem ser **herdados** ou **sintetizados**.
  - O atributo de um símbolo é **sintetizado** se o seu valor é definido a partir dos valores dos símbolos **abaixo** dele na árvore.
  - O atributo de um símbolo é **herdado** se o seu valor é definido a partir de valores de símbolos **acima** dele na árvore.

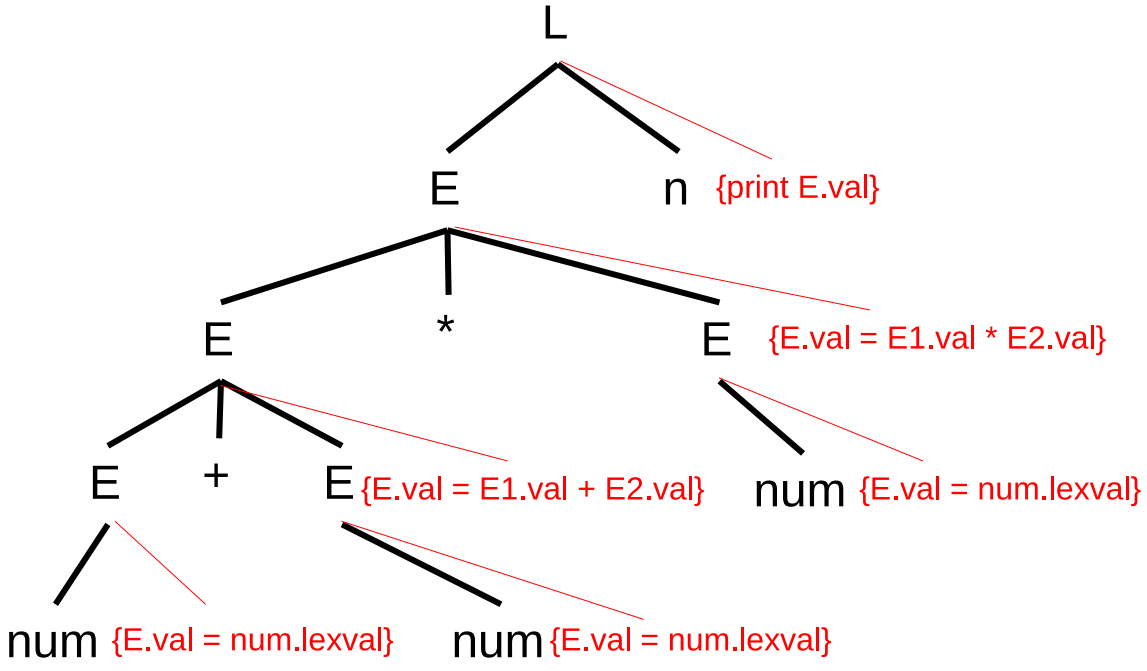
Uma **definição** dirigida por sintaxe para uma calculadora simples:

Regra de Produção	Regra Semântica
$L \rightarrow En$	$L.val = E.val$
$E \rightarrow E_1 + E_2$	$E.val = E_1.val + E_2.val$
$E \rightarrow E_1 * E_2$	$E.val = E_1.val * E_2.val$
$E \rightarrow num$	$E.val = num.lexval$

Tradução Dirigida por Sintaxe

A **tradução** dirigida por sintaxe para uma calculadora simples:

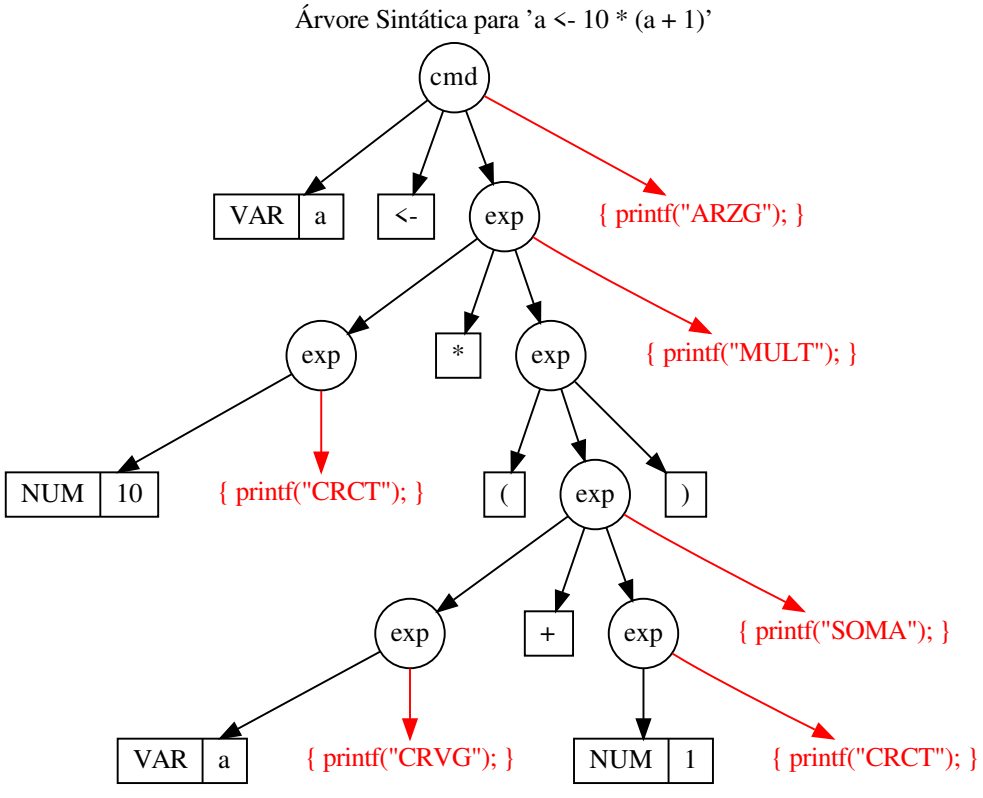
Regra de Produção	Regra Semântica
$L \rightarrow En$	{ printf("%d", $E.val$);}
$E \rightarrow E_1 + E_2$	{ $E.val = E_1.val + E_2.val$; }
$E \rightarrow E_1 * E_2$	{ $E.val = E_1.val * E_2.val$; }
$E \rightarrow num$	{ $E.val = num.lexval$; }



Geração de código MVS (calculadora)

```
cmd :
     expr                  { }
   | VAR ATRIBUI expr
     {printf("\tARZG\t%d\n",$1);}
   ;

expr :
     NUM
     {printf("\tCRCT\t%d\n",$1);}
   | VAR
     {printf("\tCRVG\t%d\n",$1);}
   | expr MAIS expr
     {printf("\tSOMA\n");}
   | expr MENOS expr
     {printf("\tSUBT\n");}
   | expr BARRA expr
     {printf("\tDIVI\n");}
   | expr VEZES expr
     {printf("\tMULT\n");}
   | ABRE expr FECHA    { }
   ;
```



Geração de código