Cap. 5 Tradução Dirigida pela Sintaxe

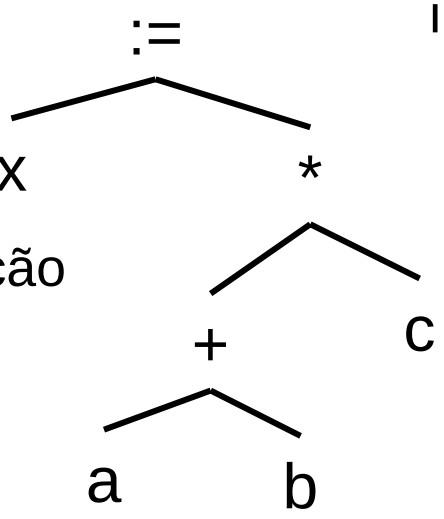
Tradução de um programa fonte em um programa objeto é feita em 2 passos e uma representação intermediária é construída

Essa representação intermediária é tipicamente encontrada em duas formas alternativas:

- **Árvore sintática**
- **ou Seqüência de comandos (código de três endereços)**

Ex. x:= (a+b)*c

Representação em árvore



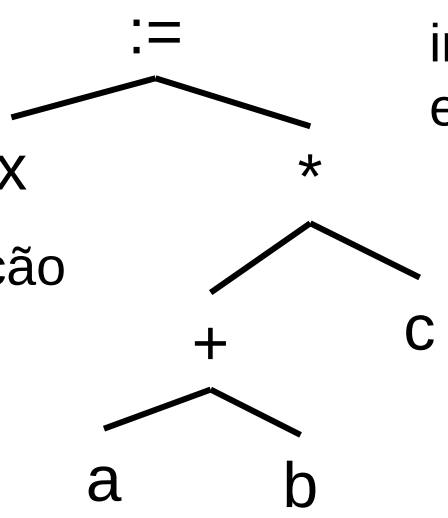
Representação em seqüência de instruções

+	a	b	t1
*	t1	c	t2
:=	x	t1	...

Cap. 5 Tradução Dirigida pela Sintaxe

Ex. x:= (a+b)*c

Representação em árvore



Representação em seqüência de instruções, com código de 3 endereços

+	a	b	t1
*	t1	c	t2
:=	x	t1	...

Para os dois casos de representação
A transformação em código de máquina é SIMPLES.

Iremos abordar inicialmente a representação por código de 3 endereços, chamado de quádruplas

Estruturas envolvidas

Pilha do autômato em movimento na análise do front-end

Gramática

Entrada prog. fonte

TABELA sLR(1)

Tabela de Símbolos

prox

Tabela de Quádruplas

Sintaxe

0. S' → S
1. S → V = E
2. S → I (E)
3. E → E + V
4. E → V
5. V → V = I (E)
6. V → I
7. I → a

	pilha sintática	entrada do prog.fonte
1	{...,...,0}	x=y+z=f(r+t)$
2	{...,...,0}{a,x,4}	=y+z=f(r+t)$
3	{...,...,0}{I,0,3}	
4	{...,...,0}{V,0,2}	
5	{...,...,0}{V,0,2}{=,...,5}	y+z=f(r+t)$
6	{...,...,0}{V,0,2}{=,...,5}{a,y,4}	+z=f(r+t)$
7	{...,...,0}{V,0,2}{=,...,5}{I,1,8}	
8	{...,...,0}{V,0,2}{=,...,5}{V,1,9}	
9	{...,...,0}{V,0,2}{=,...,5}{E,1,7}	
10	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}	z=f(r+t)$
11	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{a,z,4}	=f(r+t)$
12	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{I,2,11}	
13	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	
14	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}	f(r+t)$
15	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{a,f,4}	(r+t)$
16	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}	
17	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}	r+t)$
18	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{a,r,4}	+t)$
19	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{I,4,11}	
20	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{V,4,9}	
21	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}	
22	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}	t)$
23	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{a,t,4}	)$
24	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{I,5,11}	
25	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}	
26	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}	
27	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	
28	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	
29	{...,...,0}{V,0,2}{=,...,5}{E,51,7}	
30	{...,...,0}{S,...,1}	
31	{...,...,0}{S',...,...} a redução da regra 0 encerra a análise	

Tabela sLR(1)

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										R0
2					5					
3					R8	6	R8	R8		R8
4					R7	R7	R7	R7		R7
5		7	9	8					4	
6		10	9	11					4	
7								12		R1
8					R8	13	R8	R8		R8
9					14		R4	R4		R4
10							15	12		
11					R6		R6	R6		R6
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										R2
16					14		R3	R3		R3
17							19	12		
18						13				
19					R5		R5	R5		R5

Tabela de Símbolos – observar que os a
o índice

Índice	nome	entidade
0	x	Variavel
1	y	Variavel
2	z	Variavel
3	f	Funcao
4	r	Variavel
5	t	Variavel
...		
50	t1	Tempora
51	t2	T

1	0 : [ADD 4 5 50]
2	1 : [PARAM 50]
3	2 : [CALL 3]
4	3: [POP 2]
5	4 : [ADD 1 2 51]
6	5 : [STO 51 0]

Cap. 5 Tradução Dirigida pela Sintaxe

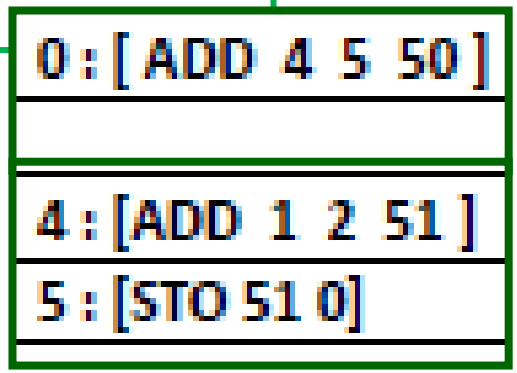
Iremos abordar inicialmente a representação por código de 3 endereços, chamado de quádruplas, seguindo um exemplo de implementação.

Quádruplas (código de 3 endereços)

Quádruplas são geradas em Tabela ou Arquivo

Cada Entrada na Tabela contém 4 Campos para representar uma quádrupla:

operador	operando1	operando2	operando3



Quem preenche Quádrupla é **gera** ou **remenda**

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like)
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
→ V	{$$.indiceSimb = $1.indiceSimb;}
→ V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
→ I	{$$.indiceSimb = $1.indiceSimb;}
→ a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

0 : [ADD 4 5 50]

1 : [PARAM 50]
2 : [CALL 3]
3: [POP 2]

4 : [ADD 1 2 51]

5 : [STO 51 0]

{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}	
{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}	0 : [ADD 4 5 50]
{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	
{.....,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	1 : [PARAM 50] 2 : [CALL 3] 3: [POP 2]
{.....,0}{V,0,2}{=,...,5}{E,51,7}	4 : [ADD 1 2 51]
{.....,0}{S,...,1}	5 : [STO 51 0]

G2

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
4. E → V	{$$.indiceSimb = $1.indiceSimb;}
5. V → V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
6. V → I	{$$.indiceSimb = $1.indiceSimb;}
7. I → a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}	0 : [ADD 4 5 50]
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	1 : [PARAM 50] 2 : [CALL 3] 3: [POP 2]
{...,...,0}{V,0,2}{=,...,5}{E,51,7}	4 : [ADD 1 2 51]
{...,...,0}{S,...,1}	5 : [STO 51 0]

Cap. 5 Tradução Dirigida pela Sintaxe

Entrada na tabela para representar 1 quádrupla:

operador	operando1	operando2	operando3

Para criar uma entrada na tabela, pode ser usada uma rotina chamada `gera (op, op1, op2, op3)`

Uma variável global chamada `prox` é incrementada a cada chamada `gera`, e sempre aponta para a próxima entrada (quádrupla) a ser gerada.

Referência a uma entrada (instrução) pode ser pela posição na tabela

```
void gera (
      Operador codop,int end1,int end2,int end3){
      quadrupla [prox].op = codop;
      quadrupla [prox].operando1 = end1;
      quadrupla [prox].operando2 = end2;
      quadrupla [prox].operando3 = end3;
      prox++;
}
```

	gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
	{$$.indiceSimb = $1.indiceSimb;}

Cap. 5 Tradução Dirigida pela Sintaxe

Precisamos de variáveis temporárias para guardar valores intermediários no cálculo de expressões

uma função temporária `temp()` devolve o endereço da nova temporária criada na tabela de símbolos.

Atenção ! variáveis temporárias têm o mesmo status de variáveis definidas pelo programador.

Elas têm tipo, valor, endereço de memória.

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like)
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
4. E → V	{$$.indiceSimb = $1.indiceSimb;}
5. V → V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
6. V → I	{$$.indiceSimb = $1.indiceSimb;}
7. I → a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}	0 : [ADD 4 5 50]
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	1 : [PARAM 50] 2 : [CALL 3] 3: [POP 2]
{...,...,0}{V,0,2}{=,...,5}{E,51,7}	4 : [ADD 1 2 51]
{...,...,0}{S,...,1}	5 : [STO 51 0]

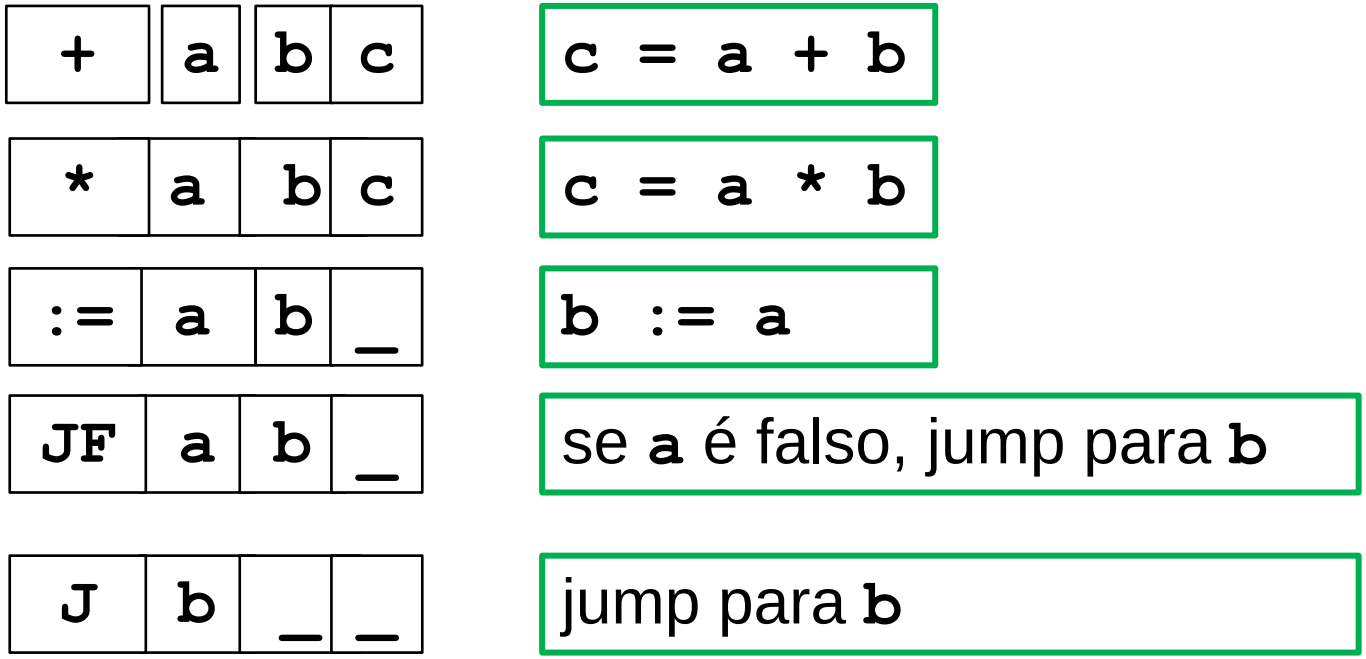
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }

{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}	0 : [ADD 4 5 50]
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	
{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}	1 : [PARAM 50] 2 : [CALL 3] 3: [POP 2]
{...,...,0}{V,0,2}{=,...,5}{E,51,7}	4 : [ADD 1 2 51]
{...,...,0}{S,...,1}	5 : [STO 51 0]

Tabela de Símbolos – observar que os atributos na pilha serão preenchidos com o índice da tabela

Índice	nome	entidade	Valor da Variável ou Endereço de Chamada para Procedimento e Função
0	x	Variavel	
1	y	Variavel	
2	z	Variavel	
3	f	Funcao	
4	r	Variavel	
5	t	Variavel	
...			
50	t1	Temporario	
51	t2	Temporario	

Exemplos de quádruplas (instruções) , numa geração de código intermediário



ATRIBUIÇÃO DE ENDEREÇOS EFETIVOS ÀS VARIÁVEIS USADAS NO PROGRAMA E TEMPORÁRIAS

Variáveis declaradas em um bloco,
ganham espaço apenas na ativação do bloco

O endereço da variável se torna uma fórmula que leva em conta o endereço inicial do espaço de alocação e seu deslocamento

Essas decisões são tomadas na geração de código de máquina, assim, no código intermediário, variáveis de programa e temporárias são referenciadas da tabela de símbolo.

- Endereço de id pode ser obtido pela função `simb (id)`
- Endereço de temporária pode ser obtido por `temp()`

Nos exemplos, a função `simb` foi substituída pela `incluiSimbTab`
Essa função de entrada na tabela de símbolos deve devolver a posição na tabela. Ela serve tanto para busca como para inclusão.

Tabela de Símbolos – observar que os atributos na pilha serão preenchidos com o índice da tabela			
Índice	nome ou valor	entidade: Constante ou Variavel	Valor da Variável ou Constante
0	3	Constante	
1	x	Variavel	
2	4	Constante	
3	y	Variavel	
4	0	Constante	
5			
...			

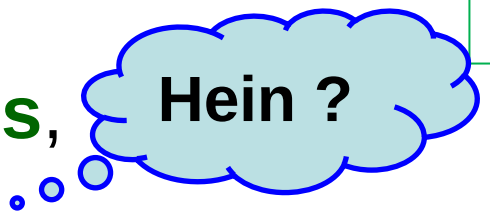
Nos exemplos, a função `simb` foi substituída pela `incluiSimbTab`

```
void incluiSimbTab(YYSTYPE simb,Entidade vOUc){
int retorno;
retorno = buscaSimbTab (simb, vOUc, Alocar);
if (retorno == FRACASSO ) {//simb deve ser incluido
  if (vOUc == Constante)
    tabSimb [topTab].valor = simb.i;//vOUc eh Constante
  else//variável,função,procedimento
    strcpy (tabSimb [topTab].nome, simb.cadeia);
  tabSimb [topTab].entidade = vOUc;
  retorno = topTab;
  topTab++;
} // de FRACASSO,incluido novo simbolo
return (retorno)
```

Implementação da função `temp` , que aloca uma entrada na tabela de símbolo para a variável temporária

```
int temp () {
char [4] nomeTemporaria;
strcpy(nomeTemporaria, "t");//prefixo
strcat(nomeTemporaria, itoa (topTemp-50);// sufixo
strcpy(tabSimb [topTemp].nome, nomeTemporaria);
tabSimb [topTemp].entidade = Temporario;
topTemp++;
return (topTemp-1);
};
```

Atributos sintetizados e herdados, durante a compilação



cada **símbolo da gramática** pode estar **representado** por uma **estrutura** com **campos** para **manter informações**

Uma dessas informações será um atributo da **entidade** do programa **que** corresponde ao **símbolo da gramática,**
que pode estar representando**:**

uma **cadeia** ou
um **tipo** ou
uma **localização de memória** em que se encontra
- um **valor** referenciado por uma **variável/cte**
- uma **instrução**
- um **procedimento** ou **função**

Atributos sintetizados e herdados, durante a compilação

Um valor para um atributo
está em um nó da árvore gramatical
e é definido por uma regra semântica
associada à regra gramatical usada naquele nó

O valor de um atributo Sintetizado é computado dos valores dos atributos dos filhos daquele nó da árvore gramatical

O valor de um atributo Herdado é computado a partir de atributos dos irmãos e pai daquele nó da árvore gramatical

Cálculo de atributos sintetizados, supondo a regra $A \rightarrow X1\ X2\ ...\ Xm$ e um método ascendente de análise sintática, tipo laLR(1), que é o usado em ferramenta Yacc

Na redução, é necessário o seguinte tratamento com os atributos dos símbolos da regra:

Na hora de dar o pop do símbolo Xi, guardar o atributo da unidade removida da pilha

Quando todos os símbolos do lado direito tiverem sido desempilhados, os seus atributos estarão acessíveis, assim, o atributo para o símbolo A do lado esquerdo poderá ser calculado

Atributos comuns associados aos símbolos são os seguintes: endereço, nome, valor, tamanho, localização

Ações Semânticas

As regras semânticas estabelecem dependência entre os atributos

Na avaliação de cada regra semântica é calculado o valor do atributo do símbolo do lado esquerdo, durante a análise de uma dada cadeia

```
exp:NUM {$$ = $1;}
    |exp '+' exp {$$ = $1 + $3;}
    |exp '-' exp {$$ = $1 - $3;}
    |exp '*' exp {$$ = $1 * $3;}
    |exp '/' exp {$$ = $1 / $3;}
    |'-' exp %prec NEG {$$ = -$2;}
    |exp '^' exp {$$ = pow($1,$3);}
    |'(' exp ')' {$$ = $2;}
;
%%
```

Cada regra de produção pode estar associada a um trecho de código (ações) , que pode ser executado quando a regra for reduzida.

Atenção ! Algumas ações se referem à própria compilação e outras à geração de código.

Não existe um limite imposto do que esse código deve fazer e esse código é compilado juntamente com o analisador

Ele pode imprimir mensagem, para a atividade de análise ou tratar as estruturas de dados do compilador

As operações que executam em combinação com as regras são chamadas de ações semânticas, porque elas agem além da sintaxe, sendo muitas relacionadas ao significado do programa

Quando uma ação semântica é efetuada para a regra $A \rightarrow X_1 ... X_n$, os valores semânticos associados a cada símbolo Xi da regra podem ser acessados

Num analisador de baixo para cima, os valores semânticos de $X_1 ... X_n$ são disponíveis no momento em que a regra de produção está para ser reduzida a A e as ações semânticas preparam um valor para A

Num analisador de cima para baixo, um valor para A é disponível quando a regra de produção é expandida e os valores dos atributos de $X_1 ... X_n$ vão sendo preenchidos à medida que vão retornando no caminho da árvore

Quando uma **ação semântica** é efetuada para a **regra** $A \rightarrow X_1 \ldots X_n$, os **valores semânticos associados** a cada **símbolo Xi** da **regra** podem ser acessados

Num analisador de baixo para cima, os **valores semânticos** de $X_1 \ldots X_n$ são disponíveis no momento em que a **regra de produção** está para ser **reduzida** a **A** e **as ações semânticas** preparam um **valor** para **A**

```
/* Grammar follows */
%%
input:  /* empty string */
      |input line
;

line:   '\n'
|exp '\n'{printf("\t%.10g\n",$1);}
;

exp:NUM {$$ = $1;}
     |exp '+' exp {$$ = $1 + $3;}
     |exp '-' exp {$$ = $1 - $3;}
     |exp '*' exp {$$ = $1 * $3;}
     |exp '/' exp {$$ = $1 / $3;}
     |'-' exp %prec NEG {$$ = -$2;}
     |exp '^' exp {$$ = pow($1,$3);}
     |'(' exp ')' {$$ = $2;}
;
%%
```

Ações semânticas

Cap. 5 Tradução Dirigida pela Sintaxe

Técnicas Básicas para geração de quádruplas

Supondo o seguinte:

Tratamento das declarações já feito

Existe uma tabela de símbolos, na qual podem ser encontrados os endereços das entidades: variáveis ou constantes, chamando, p/ex., `incluiSimbTab($1, Variavel`, que devolve o endereço do nome, na tabela.

A função `gera`, gera as quádruplas na tabela de quádruplas e atualiza a global `prox` (endereço da quádrupla)

Os atributos sintetizados, que encontram-se na pilha podem ser endereço, valor, nome de variável, tipo, ...

Em vários momentos, a seguir, vamos usar a notação tipo Yacc, que indica os símbolos pela posição e chama o campo de atributo de $i para os símbolos à direita da regra e de $$ o da esquerda da regra.

Itens importantes que devem ser levados em conta na tradução em compiladores de subida como os laLR(1)

Considerando a regra genérica, a seguir: $X \rightarrow X_1 \ldots X_i \ldots X_n$

A cada transição, é armazenada uma unidade na pilha associada ao símbolo Xi (terminal ou não terminal).
A estrutura dessa unidade contém 3 membros

1) **Xi símbolo da gramática**
2) **atributo associado ao Xi**
3) **estado do autômato**

No momento da transição de redução da regra, a tradução providencia o pop de cada símbolo Xi que está empilhado, começando pelo topo (símbolo Xn mais à direita) até chegar ao X1.
A cada Xi o membro atributo da estrutura da unidade de pilha é guardado na variável $i.

O cálculo do valor de $$, que é uma dentre as várias ações, na redução, deve ser codificado pelo programador do compilador.
Esse código é associado à regra,
e será executado no momento dessa transição, na compilação.
O valor de $$ será guardado no atributo da unidade do símbolo X.
Lembrar que o símbolo X está à esquerda da regra.
Quando a estrutura estiver preenchida, com os campos símbolo, atributo e estado, ela deve ser empilhada.
Por isso o nome redução, pop dos símbolos da direita e push da esquerda

Itens importantes a serem levados em conta nas ferramentas tipo Yacc

Considerando a regra genérica:

$$X \rightarrow X_1 \ldots X_i \ldots X_n$$

```
typedef union {
  char            cadeia [20];
  int             i;
  Logico          b;
  Operador        opcode;
  tipoIntOUBool   tipo;
  int             indiceSimb;
  int             indiceQuadrupla;
  } YYSTYPE;
```

Devemos lembrar que ao especificarmos um compilador, usando o yacc, apenas estamos declarando os seus parâmetros.

Assim, variáveis externas serão necessárias, como as $$ e $i.

As variáveis $i para cada Xi e a $$ para X são do tipo YYSTYPE, Esse tipo é definido como uma union em C.

Uma variável declarada com esse tipo pode assumir vários tipos, sendo os mais comuns: os de cadeia, de endereço de tabela de símbolo, de endereço de quádrupla, de tipo.

LEX e YACC

Itens importantes que devem ser levados em conta

Considerando a regra genérica, a seguir: $X \rightarrow X_1 ... X_i ... X_n$

A variável **`prox`** indica a próxima posição disponível na tabela de quádruplas (código),
e tem uma grande importância nas várias ações semânticas, que auxiliam ou vão fazer parte de desvios de código nas estruturas de controle nos comandos if, for, while, repeat

Técnicas Básicas de Geração de Quádruplas

Regras de declaração	Ações Semânticas na Redução, em Yacc
DV → DV , IDENT	**{entra ($3.cadeia, $1.tipo); $$.tipo = $1.tipo;}**
DV → TIPO IDENT	**{entra ($2.cadeia, $1.tipo); $$.tipo = $1.tipo;}**
TIPO → int	**{$$.tipo = i;}**
TIPO → char	**{$$.tipo = c;}**

As regras:

TIPO → int ou TIPO → char serão reduzidas em primeiro lugar,
e suas ações semânticas guardarão o tipo em $$.tipo,
que será o $1.tipo da regra **DV → TIPO IDENT**
cujas ações semânticas guardarão esse $1.tipo em $$.tipo,
que será o $1.tipo da regra **DV → DV , IDENT**
que assim terá para os próximos **IDENT**'s que aparecerem no programa fonte
o tipo guardado em $1.tipo.

Técnicas Básicas de Geração de Quádruplas

Exemplo para a gramática de uma linguagem de programação simples :

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like)
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
4. E → V	{$$.indiceSimb = $1.indiceSimb;}
5. V → V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
6. V → I	{$$.indiceSimb = $1.indiceSimb;}
7. I → a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

Técnicas Básicas de Geração de Quádruplas

Regras de Expressão

Para as regras **E→T, T → F** e **F →(E),** a situação é semelhante a de **E → V** da gramática anterior:

E → T {$$.indiceSimb = $1.indiceSimb}

T → F {$$.indiceSimb = $1.indiceSimb}

F → (E) {$$.indiceSimb = $2.indiceSimb}

Para as regras **E → E + T** e **T → T * F** , a situação é semelhante a de **E → E + V** da gramática anterior:

E→E + T {$$.indiceSimb = temp ( );
gera(ADD,$1.indiceSimb,$3.indiceSimb,$$.indiceSimb)}

T→T * F {$$.indiceSimb = temp ( );
gera(MULT,$1.indiceSimb,$3.indiceSimb,$$.indiceSimb)}

Sintaxe original	Sintaxe com terminais nomeados por 1 letra p facilitar a implementação	Ações semânticas com uso de $ para atributos (Yacc-like)
0. L' → L	0. L' → L	{encerra();}
1. L → S ; L	1. L → S ; L	
2. L → S	2. L → S	
3. S → v = E	3. S → v = E	{$1. indSimb = incluiSimbTab ($1, Variavel); gera(STO, $3.indSimb,$1.indSimb, NADA)}
4. S → if E then T	4. S → s E t M T	{remenda ($4.indQuadr, JF, $2.indSimb, prox)}
5. S → while E do T	5. S → w N E M d T	{gera (J, $2.ind Quadr, NADA, NADA); remenda($4.indQuadr, JF, $3.indSimb,prox,NADA)}
6. E → v	6. E → v	{$$.indSimb = incluiSimbTab ($1, Variavel)}
7. E → n	7. E → n	{$$.indSimb = incluiSimbTab ($1, Constante)}
	8. M → ε	{$$.indQuadr = prox; prox++;}
	9. N → ε	{$$.indQuadr = prox;}
8. T → { L }	10. T → { L }	

Alterações de regras p acerto de desvios

Terminais associados	Programa fonte edentado na sintaxe original	Programa fonte edentado na sintaxe com os terminais resumidos

Terminais associados
while w
do d
if s
then t
	x = 3;
	y = 4;
	while x do {
	if y then {
	x = 0
	};
	y = 0
	}
		x = 3;
		y = 4;
		w x d {
		s y t {
		x = 0
		};
		y = 0
		}

aAs variáveis `x`, `y` correspondem ao terminal `v` e as constantes numéricas `3`, `4` e `0` correspondem ao terminal `n`

Sintaxe original	Sintaxe com terminais nomeados por 1 letra p facilitar a implementação	Ações semânticas com uso de $ para atributos (Yacc-like)
0. L' → L	0. L' → L	{encerra();}
1. L → S ; L	1. L → S ; L	
2. L → S	2. L → S	
3. S → v = E	3. S → v = E	{$1. indSimb = incluiSimbTab ($1, Variavel); gera(STO, $3.indSimb,$1.indSimb,NADA)}
4. S → if E then T	4. S → s E t M T	{remenda ($4.indQuadr, JF, $2.indSimb, prox)}
5. S → while E do T	5. S → w N E M d T	{gera (J, $2.ind Quadr, NADA, NADA); remenda($4.indQuadr, JF, $3.indSimb,prox,NADA)}
6. E → v	6. E → v	{$$.indSimb = incluiSimbTab ($1, Variavel)}
7. E → n	7. E → n	{$$.indSimb = incluiSimbTab ($1, Constante)}
	8. M → ε	{$$.indQuadr = prox; prox++;}
	9. N → ε	{$$.indQuadr = prox;}
8. T → { L }	10. T → { L }	

Regras que precisam mudanças para ajudar nas ações

COM → IF COND THEN LISTACOM ENDIF

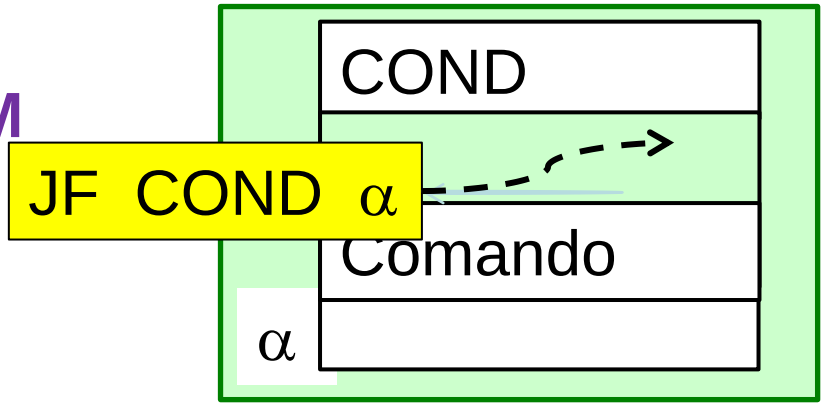
Esta regra vai mudar na especificação para o compilador, por causa da necessidade de se abrir espaço para a quádrupla que será gerada, quando se souber o endereço da instrução, para onde o jump desviará, como, a seguir:

COM → IF COND M THEN LISTACOM ENDIF

M → ε

Com essa mudança, as ações em **M**

```
{
$$.indiceQuadrupla = prox;
prox++;
}
```

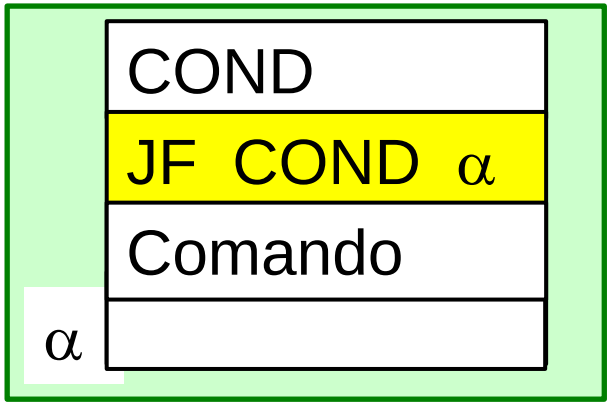


- guardam a posição da quádrupla na tabela. Essa quádrupla receberá a instrução "`JF COND α`", no momento da redução da regra `if`**. Essa posição estará guardada no atributo de M ($3).**

- abrem o espaço, incrementando a variável `prox`.

Regras que precisam mudanças para ajudar nas ações

M → ε {
`$$.indiceQuadrupla = prox;`
`prox++;`
}



Observe que essa ação de **M** vai
- na primeira instrução: guardar a posição da tabela aonde a quádrupla para "**`JF ...`**", a ser gerada no final da análise de **`if`**, deverá ser inserida contando com o procedimento **`remenda`**
- abrir o espaço, ao incrementar a variável **`prox`**, para permitir essa inserção, que pode ser vista abaixo, na ação da regra **`if`**.

COM → IF COND M THEN LISTACOM ENDIF

{**`remenda($3.indiceQuadrupla,JF,$2.indiceSimb,prox, NADA)`**}

`$3.indiceQuadrupla` **(posição da quádrupla a ser remendada)**
`JF` **(jump se falso)**
`$2.indiceSimb` **(variável que guarda a condição false/true)**
`prox` **(endereço da próxima instrução fora do bloco do if)**

Regras que precisam mudanças para ajudar nas ações

COM → IF COND THEN LISTACOM ELSE LISTACOM ENDIF

Esta regra muda, por causa da necessidade de se abrir espaço para as quádruplas a serem remendadas, quando se souber o endereço da instrução, para onde cada jump desviará, como, a seguir:

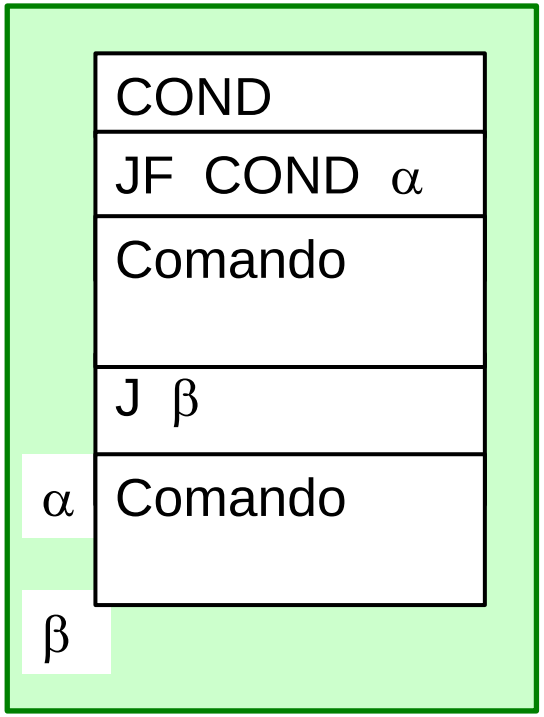
COM → IF COND M THEN LISTACOM M ELSE LISTACOM ENDIF

M → ε

Assim, com **M** suas ações

```
{
$$.indiceQuadrupla = prox;
prox++;
}
```

permitem abrir os dois espaços, para guardar os locais da tabela aonde as quádruplas para "**JF COND ?**" e "**J ?**" serão remendadas no final da análise de **IF**.

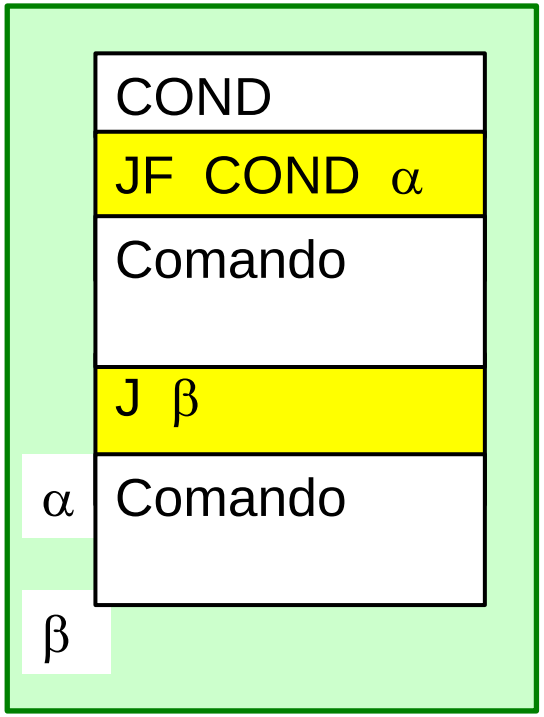


Regras que precisam mudanças para ajudar nas ações

Aqui nessa situação de regra também há a necessidade de empregar a regra M

COM → IF COND M THEN LISTACOM M ELSE LISTACOM ENDIF

```
{
remenda(
$3.indiceQuadrupla,JF,$2.indiceSimb,
$6.indiceQuadrupla+1,NADA);

remenda(
$6.indiceQuadrupla, J, prox, NADA,NADA);
}
```



Regras que precisam mudanças para ajudar nas ações

COM → WHILE COND DO LISTACOM ENDDO

Esta regra vai mudar na especificação para o compilador, por causa da necessidade de:

- guardar a posição da quádrupla antes de COND para que no fim do WHILE seja gerado um jump incondicional para este endereço
- abrir um espaço para a quádrupla, que será gerada quando se souber o endereço da instrução para onde o jump condicional desviará, caso COND seja falsa;

como, a seguir:

COM → WHILE N COND M DO LISTACOM ENDDO

M → ε

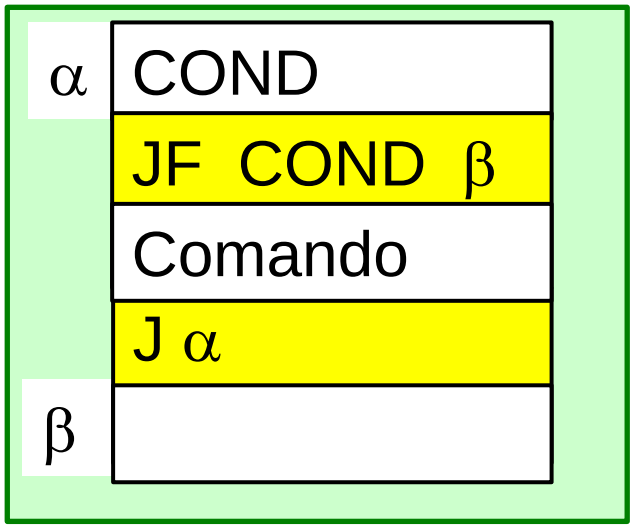
N → ε

Ações na redução de **M → ε**

```
{ $$.indiceQuadrupla = prox;
prox++; }
```

Ações na redução de **N → ε**

```
{$$.indiceQuadrupla = prox;}
```



Regras que precisam mudanças para ajudar nas ações

COM → WHILE N COND M DO LISTACOM ENDDO

M → ε

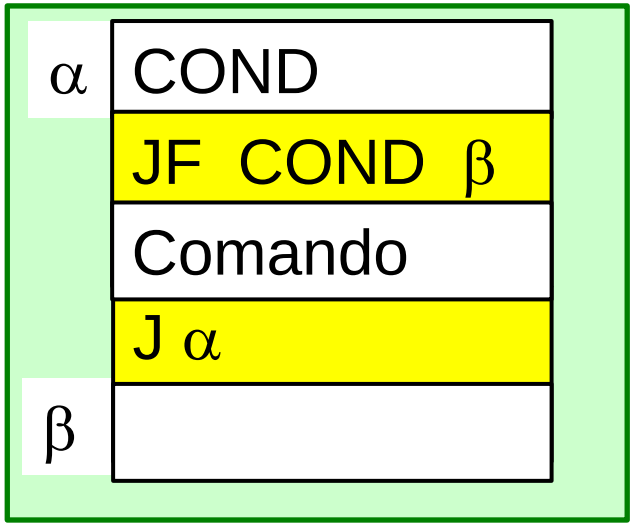
N → ε

Ações na redução de **N → ε**

```
{$$.indiceQuadrupla = prox;}
```

Ações na redução de **M → ε**

```
{ $$.indiceQuadrupla = prox;
prox++; }
```



Ações na redução de **COM → WHILE N COND M DO LISTACOM ENDDO**

```
{
gera(J,$2.indiceQuadrupla,NADA,NADA);

remenda($4.indiceQuadrupla,JF,$3.indiceSimb,prox,NADA);
}
```

Os autômatos de pilha ascendentes vistos no cap. 3,
trataram somente com a informação do símbolo gramatical
(terminal e não terminal) na unidade da pilha.
Vindo de autômato de pilha reconhecido por cadeia vazia e pilha vazia.
Agora, são acrescentadas, à estrutura da unidade de pilha, as informações de estado (obtido pelo LR(1) e seus derivados e atributo.
Assim, a unidade vira uma estrutura (símbolo, atributo, estado)

Transições para os exercícios de chinês a seguir:

1) Se $\delta(p, a) = q$, empilha a estrutura : (a, atributo, q)

2) Se $\delta(p, a) = R_i$ e regra i é $B \rightarrow \gamma$

então

reduz $|\gamma|$ (nº de pop's)

verifica na tabela de estados, o estado do topo com B

δ(topo, B) = q,

empilha a estrutura (B, atributo, q) ($$ é o atributo do lado esq.)

3) Quando chegar a R_0, como resultado da transição,

então encerra a análise.

G3

Sintaxe original	Sintaxe com terminais nomeados por 1 letra p facilitar a implementação	Ações semânticas com uso de $ para atributos (Yacc-like)
0. L' → L	0. L' → L	{encerra();}
1. L → S ; L	1. L → S ; L	
2. L → S	2. L → S	
3. S → v = E	3. S → v = E	{$1. indSimb = incluiSimbTab ($1, Variavel); gera(STO, $3.indSimb,$1.indSimb,NADA)}
4. S → if E then T	4. S → s E t M T	{remenda ($4.indQuadr, JF, $2.indSimb, prox)}
5. S → while E do T	5. S → w N E M d T	{gera (J, $2.ind Quadr, NADA, NADA); remenda($4.indQuadr, JF, $3.indSimb,prox,NADA)}
6. E → v	6. E → v	{$$.indSimb = incluiSimbTab ($1, Variavel)}
7. E → n	7. E → n	{$$.indSimb = incluiSimbTab ($1, Constante)}
	8. M → ε	{$$.indQuadr = prox; prox++;}
	9. N → ε	{$$.indQuadr = prox;}
8. T → { L }	10. T → { L }	

Terminais associados	Programa fonte edentado na sintaxe original	Programa fonte edentado na sintaxe com os terminais resumidos

Terminais associados
while w
do d
if s
then t
	x = 3;
	y = 4;
	while x do {
	if y then {
	x = 0
	};
	y = 0
	}
		x = 3;
		y = 4;
		w x d {
		s y t {
		x = 0
		};
		y = 0
		}

aAs variáveis **x**, **y** correspondem ao terminal v e as constantes numéricas **3**, **4** e **0** correspondem ao terminal n

tabela laLR(1) para a G3

	L	S	E	M	N	T	;	v	=	s	w	n	{	}	t	d	$	
0	1	2						3		4	5							0
1																	R_0	1
2							6							R_2			R_2	2
3									7									3
4			8					9				10						4
5					11			R_9				R_9						5
6	12	2						3		4	5							6
7			13					9				10						7
8															14			8
9							R_6							R_6	R_6	R_6	R_6	9
10							R_7							R_7	R_7	R_7	R_7	10
11			15					9				10						11
12														R_1			R_1	12
13							R_3							R_3			R_3	13
14				16									R_8					14
15				17												R_8		15
16						18							19					16
17																20		17
18							R_4							R_4			R_4	18
19	21	2						3		4	5							19
20						22							19					20
21														23				21
22							R_5							R_5			R_5	22
23							R_{10}							R_{10}			R_{10}	23
	L	S	E	M	N	T	;	v	=	s	w	n	{	}	t	d	$	

```
int incluiSimbTab(
YYSTYPE simb,Entidade vOUc){
int retorno;
/*se busca encontrar, retorna posição da tabela*/
retorno = buscaSimbTab (simb, vOUc, Alocar);
if (retorno == FRACASSO ) {//simb deve ser incluido
  if (vOUc == Constante)
    tabSimb [topTab].valor = simb.i;//Constante
  else//variável
    strcpy (tabSimb [topTab].nome, simb.cadeia);
    tabSimb [topTab].entidade = vOUc;
  retorno = topTab; // de FRACASSO,incluido novo simbolo
  topTab++;
  }
return (retorno)
}
```

```
x = 3;
y = 4;
while x do {
  if y then {
    x = 0
    };
  y = 0
  }
```

tabela

Índice	nome/valor	entidade: Cte/Variavel	Valor
0	3	Constante	
1	x	Variavel	
2	4	Constante	
3	y	Variavel	
4	0	Constante	
5			
...			

G3

```
x = 3;y = 4; w x d{s y t{x = 0}; y = 0}$
```

```
x = 3;
y = 4;
w x d {
   s y t {
      x = 0
      };
   y = 0
   }
```

	pilha sintática	entrada do prog.fonte	prox	i:[op op1 op2 op3
1	{...,...,0}	x=3 ; y=4 ; . . .	0	
2	{...,...,0}{v,x,3}	=3 ; y=4 ; . . .		
3	{...,...,0}{ v,x,3}{=,...,7}	3 ; y=4 ; . . .		
4	{...,...,0}{ v,x,3}{=,...,7}{n,3,10}	;y=4; w...		
5	{...,...,0}{ v,x,3}{=,...,7}{E,0,13}			
6	{...,...,0}{ S,...,2}		1	0:[STO,0,1]
7	{...,...,0}{ S,...,2}{;,...,6}	y=4; w...		
8	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}	=4;w...		
9	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}	4;w x d		
10	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}{n,4,10}	;wxd		
11	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}{E,2,13}			
12	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}		2	1:[STO,2,3]
13	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}	wxd{		
14	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5} estranho ? aqui vai haver a redução da regra 9. N → ε , em que o valor do prox é guardado no atributo de N.	xd{		
15	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}			
16	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{v,x,9}	d{syt{x=0};y=0}$		
17	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15} estranho ? aqui vai haver a redução da regra 8. M → ε , em que o valor do prox é guardado no atributo de M e prox é incrementado para abrir espaço p remendar o JF nesta quádrupla.			
18	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15}{M,2,17}		3	
19	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15}{M,2,17}{d,...,20} retomada na l. 44	{syt{x=0};y=0}$		
20	{{,...,19}	syt{x=0};y=0}$		
21	{{,...,19}{s,...,4}{v,y,9}	t{x=0};y=0}$		
22	{{,...,19}{s,...,4}{E,3,8}			
23	{{,...,19}{s,...,4}{E,3,8}{t,...,14} estranho ? aqui vai haver a redução da regra 8. M → ε , em que o valor do prox é guardado no atributo de M e prox é incrementado para abrir espaço p remendar o JF nesta quádrupla.	{x=0};y=0}$		
24	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}		4	

	pilha sintática `x = 3;y = 4; w x d{s y t{x = 0}; y = 0}$`	entrada do prog.fonte	prox	i:[op op1 op2 op3]
1	{...,...,0}	x=3 ; y=4 ; . . .	0	
2	{...,...,0}{v,x,3}	=3 ; y=4 ; . . .		
3	{...,...,0}{ v,x,3}{=,...,7}	3 ; y=4 ; . . .		
4	{...,...,0}{ v,x,3}{=,...,7}{n,3,10}	;y=4; w...		
5	{...,...,0}{ v,x,3}{=,...,7}{E,0,13}			
6	{...,...,0}{ S,...,2}		1	0:[STO,0,1]
7	{...,...,0}{ S,...,2}{;,...,6}	y=4; w...		
8	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}	=4;w...		
9	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}	4;w x d		
10	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}{n,4,10}	;wxd		
11	{...,...,0}{ S,...,2}{;,...,6}{v,y,3}{=,...,7}{E,2,13}			
12	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}		2	1:[STO,2,3]
13	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}	wxd{		
14	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5} estranho ? aqui vai haver a redução da regra 9. N → ε , em que o valor do prox é guardado no atributo de N.	xd{		
15	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}			
16	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{v,x,9}	d{syt{x=0};y=0}$		
17	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15} estranho ? aqui vai haver a redução da regra 8. M → ε , em que o valor do prox é guardado no atributo de M e prox é incrementado para abrir espaço p remendar o JF nesta quádrupla.			
18	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15}{M,2,17}		3	
19	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15}{M,2,17}{d,...,20} retomada na l. 44	{syt{x=0};y=0}$		
20	{{,...,19}	syt{x=0};y=0}$		
21	{{,...,19}{s,...,4}{v,y,9}	t{x=0};y=0}$		
22	{{,...,19}{s,...,4}{E,3,8}			
23	{{,...,19}{s,...,4}{E,3,8}{t,...,14} estranho ? aqui vai haver a redução da regra 8. M → ε , em que o valor do prox é guardado no atributo de M e prox é incrementado para abrir espaço p remendar o JF nesta quádrupla.	{x=0};y=0}$		
24	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}		4	

Transições para os exercícios de chinês a seguir:

1) Se $\delta(p, a) = q$, empilha a estrutura : (a, atributo, q)
2) Se $\delta(p, a) = R_i$ e regra i é $B \rightarrow \gamma$
   então
   reduz $|\gamma|$ (nº de pop's)
   verifica na tabela de estados, o estado do topo com B
   δ(topo, B) = q,
   empilha a estrutura (B, atributo, q) ($$ é o atributo do lado esq.)
3) Quando chegar a R_0, como resultado da transição,
   então encerra a análise.

G3

25	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}	x=0};y=0}$		
26	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{v,x,3}	=0};y=0}$		
27	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{v,x,3}{=,...,7}	0};y=0}$		
28	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{v,x,3}{=,...,7}{n,0,10}	};y=0}$		
29	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{v,x,3}{=,...,7}{E,4,13}			
30	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{S,...,2}		5	4:[STO, 4, 1]
31	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{L,...,21}			
32	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{{,...,19}{L,...,21}{},...,23}	;y=0}$		
33	{{,...,19}{s,...,4}{E,3,8}{t,...,14}{M,3,16}{T,...,18} aqui vai haver a redução da regra 4. S → s E t M T com os efeitos das ações semânticas, na coluna de quádruplas.			
34	{{,...,19}{S,...,2}			3:[JF,3,5,NADA]
35	{{,...,19}{S,...,2}{;,...,6}	y=0}$		
36	{{,...,19}{S,...,2}{;,...,6}{v,y,3}	=0}$		
37	{{,...,19}{S,...,2}{;,...,6}{v,y,3}{=,...,7}	0}$		
38	{{,...,19}{S,...,2}{;,...,6}{v,y,3}{=,...,7}{n,0,10}	}$		
39	{{,...,19}{S,...,2}{;,...,6}{v,y,3}{=,...,7}{E,4,13}			
40	{{,...,19}{S,...,2}{;,...,6}{S,...,2}		6	5:[STO,4,3,NADA}
41	{{,...,19}{S,...,2}{;,...,6}{L,...,12}			
42	{{,...,19}{{L,...,21}			
43	{{,...,19}{L,...,21}{},...,23} aqui haverá a redução da regra 10. T → { L }, cujo T será empilhado continuando o que tinha na pilha desde a linha 19, ver abaixo, deixando a regra while completa.	$		
44	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{w,...,5}{N,2,11}{E,1,15}{M,2,17}{d,...,20}{T,...,22} aqui vai haver a redução da regra 5. S → w N E M d T com os efeitos das ações semânticas, na coluna de quádruplas e do prox.			
45	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{S,...,2}		7	6:[J,2,NADA,NADA] 2:[JF,1,7,NADA]
46	{...,...,0}{ S,...,2}{;,...,6}{S,..., 2}{;,...,6}{L,...,12}			
47	{...,...,0}{ S,...,2}{;,...,6}{L,...,12}			
48	{...,...,0}{ L,...,1}			
49	{...,...,0} {L',...,...} reduz a regra 0, encerra a análise			

G2

Estado 0

S'→.S	S	1
S→.V=E	V	2
S→.I(E)	I	3
V→.V=I(E)	V	2
V→.I	I	3
I→.a	a	4

follow

S	$
E	$) +
V	= $) +
I	(= $) +

Estado 1

S'→S.	π	0

Estado 2

S→V.=E	=	5
V→V.=I(E)	=	5

Estado 3

S→I.(E)	(	6
V→I.	π	6

Estado 4

I→a.	π	7

Estado 5

S→V=.E	E	7
V→V=.I(E)	I	8
E→.E+V	E	7
E→.V	V	9
V→.V=I(E)	V	9
V→.I	I	8
I→.a	a	4

Estado 6

S→I(.E)	E	10
E→.E+V	E	10
E→.V	V	9
V→.V=I(E)	V	9
V→.I	I	11
I→.a	a	4

Estado 7

S→V=E.	π	1
E→E.+V	+	12

Estado 8

V→V=I.(E)	(	13
V→I.	π	6

Estado 9

E→V.	π	4
V→V.=I(E)	=	14

Estado 10

S→I(E.)	)	15
E→E.+V	+	12

Estado 11

V→I.	π	6

Estado 12

E→E+.V	V	16
V→.V=I(E)	V	16
V→.I	I	11
I→.a	a	4

Estado 13

V→V=I(.E)	E	17
E→.E+V	E	17
E→.V	V	9
V→.V=I(E)	V	9
V→.I	I	11
I→.a	a	4

Estado 14

V→V=.I(E)	I	18
I→.a	a	4

Estado 15

S→I(E).	π	2

Estado 16

E→E+V.	π	3
V→V.=I(E)	=	14

Estado 17

V→V=I(E.)	)	19
E→E.+V	+	12

Estado 18

V→V=I.(E)	(	13

Estado 19

V→V=I(E).	π	5

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										π0
2					5					
3					π6	6	π6	π6		π6
4					π7	π7	π7	π7		π7
5		7	9	8					4	
6		10	9	11					4	
7								12		π1
8					π6	13	π6	π6		π6
9					14		π4	π4		π4
10							15	12		
11					π6		π6	π6		π6
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										π2
16					14		π3	π3		π3
17							19	12		
18						13				
19					π5		π5	π5		π5

GRAMÁTICA

0. S'→S
1. S→V=E
2. S→I(E)
3. E→E+V
4. E→V
5. V→V=I(E)
6. V→I
7. I→a

Gramática G2 sLR(1)

G2

Gramática G2 sLR(1)

GRAMÁTICA

0. $S' \to S$
1. $S \to V = E$
2. $S \to I(E)$
3. $E \to E + V$
4. $E \to V$
5. $V \to V = I(E)$
6. $V \to I$
7. $I \to a$

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										π0
2					5					
3					π6	6	π6	π6		π6
4					π7	π7	π7	π7		π7
5		7	9	8					4	
6		10	9	11					4	
7								12		π1
8					π6	13	π6	π6		π6
9					14		π4	π4		π4
10							15	12		
11					π6		π6	π6		π6
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										π2
16					14		π3	π3		π3
17							19	12		
18						13				
19					π5		π5	π5		π5

G2

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like)
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
4. E → V	{$$.indiceSimb = $1.indiceSimb;}
5. V → V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
6. V → I	{$$.indiceSimb = $1.indiceSimb;}
7. I → a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

Tabela sLR(1)

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										R_0
2					5					
3					R_8	6	R_8	R_8		R_8
4					R_7	R_7	R_7	R_7		R_7
5		7	9	8					4	
6		10	9	11					4	
7								12		R_1
8					R_8	13	R_8	R_8		R_8
9					14		R_4	R_4		R_4
10							15	12		
11					R_8		R_8	R_8		R_8
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										R_2
16					14		R_3	R_3		R_3
17							19	12		
18						13				
19					R_5		R_5	R_6		R_5

Gramática G2

Sintaxe	Ações semânticas com uso de $ para atributos (Yacc-like)
0. S' → S	{encerra();}
1. S → V = E	{gera(STO, $3.indiceSimb,$1.indiceSimb,NADA)}
2. S → I (E)	{tabSimb[$1.indiceSimb].entidade = Procedimento; gera (PARAM,$3.indiceSimb,NADA,NADA); gera(CALL, $1.indiceSimb, NADA, NADA)}
3. E → E + V	{$$.indiceSimb = temp(); gera (ADD, $1.indiceSimb, $3.indiceSimb, $$.indiceSimb) }
4. E → V	{$$.indiceSimb = $1.indiceSimb;}
5. V → V = I (E)	{tabSimb[$3.indiceSimb].entidade = Funcao; gera (PARAM,$5.indiceSimb,NADA,NADA); gera(CALL, $3.indiceSimb, NADA, NADA); gera(POP, $1.indiceSimb, NADA, NADA) ; $$.indiceSimb=$1.indiceSimb;}
6. V → I	{$$.indiceSimb = $1.indiceSimb;}
7. I → a	{$$.indiceSimb = incluiSimbTab ($1, Variavel)}

Gramática G2

Tabela de símbolos, preenchida conforme o aparecimento dos identificadores na análise de: `x=y+z=f(r+t)$`

Tabela de Símbolos – observar que os atributos na pilha serão preenchidos com o índice da tabela

Índice	nome	entidade	Valor da Variável ou Endereço de Chamada para Procedimento e Função
0	x	Variavel	
1	y	Variavel	
2	z	Variavel	
3	f	Funcao	
4	r	Variavel	
5	t	Variavel	
...			
50	t1	Temporario	
51	t2	Temporario	

pilha prox, e quádruplas na análise de: x=y+z=f(r+t)$

G2

Transições para os exercícios de chinês a seguir:

1) Se $\delta(p, a) = q$, empilha a estrutura : (a, atributo, q)

2) Se $\delta(p, a) = R_i$ e regra i é $B \rightarrow \gamma$

então

reduz $|\gamma|$ (nº de pop's)

verifica na tabela de estados, o estado do topo com B

δ(topo, B) = q,

empilha a estrutura (B, atributo, q) ($$ é o atributo do lado esq.)

3) Quando chegar a R_0, como resultado da transição, então encerra a análise.

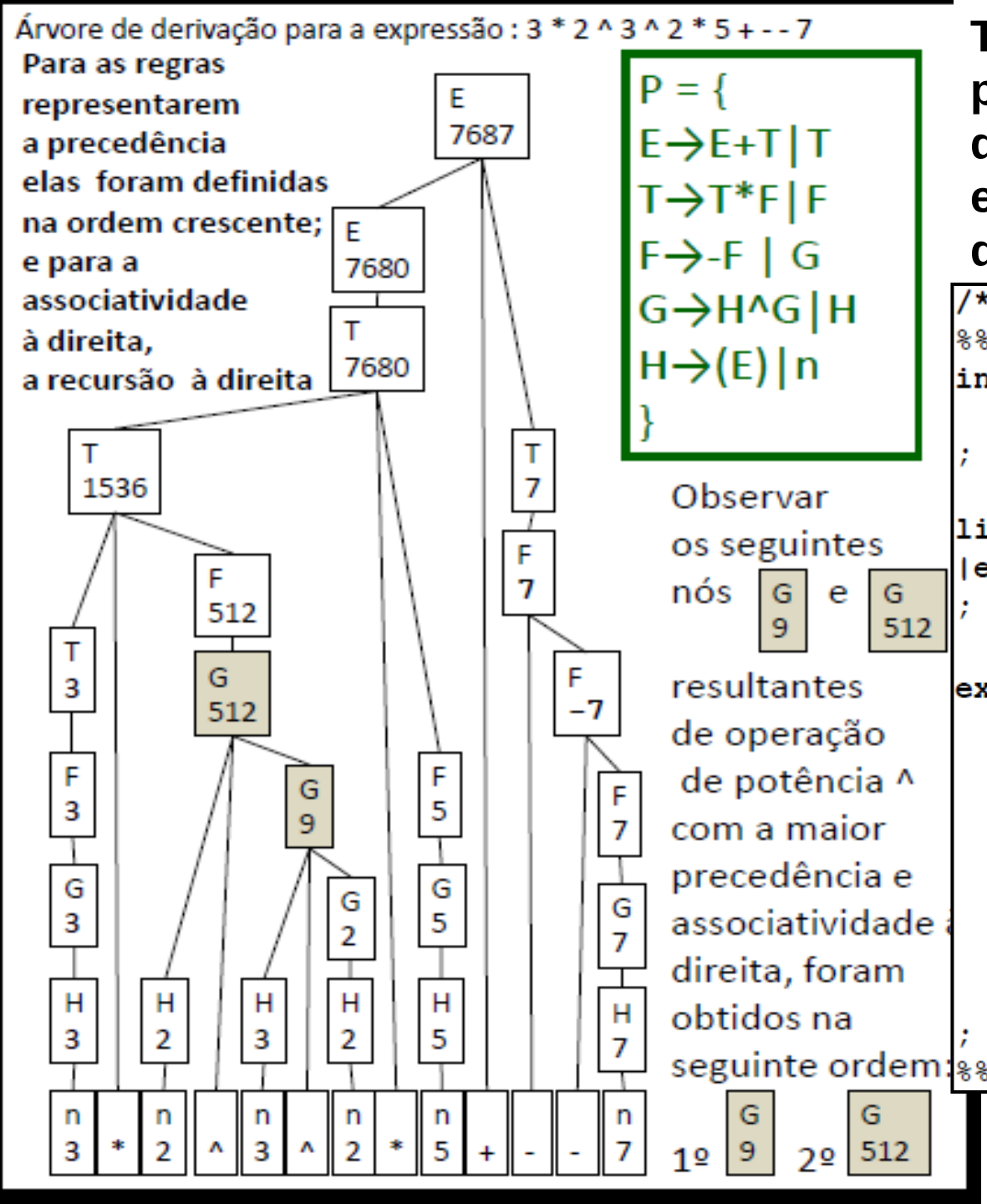
	pilha sintática	entrada do prog.fonte	prox	i:[op op1 op2 op3]
1	{...,...,0}	x=y+z=f(r+t)$	0	
2	{...,...,0}{a,x,4}	=y+z=f(r+t)$		
3	{...,...,0}{I,0,3}			
4	{...,...,0}{V,0,2}			
5	{...,...,0}{V,0,2}{=,...,5}	y+z=f(r+t)$		
6	{...,...,0}{V,0,2}{=,...,5}{a,y,4}	+z=f(r+t)$		
7	{...,...,0}{V,0,2}{=,...,5}{I,1,8}			
8	{...,...,0}{V,0,2}{=,...,5}{V,1,9}			
9	{...,...,0}{V,0,2}{=,...,5}{E,1,7}			
10	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}	z=f(r+t)$		
11	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{a,z,4}	=f(r+t)$		
12	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{I,2,11}			
13	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}			
14	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}	f(r+t)$		
15	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{a,f,4}	(r+t)$		
16	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}			
17	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}	r+t)$		
18	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{a,r,4}	+t)$		
19	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{I,4,11}			
20	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{V,4,9}			
21	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}			
22	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}	t)$		
23	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{a,t,4}	)$		
24	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{I,5,11}			
25	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,4,17}{+,...,12}{V,5,16}			
26	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}		1	0 : [ADD 4 5 50]
27	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}{=,...,14}{I,3,18}{(,...,13}{E,50,17}{),...,19}	$		
28	{...,...,0}{V,0,2}{=,...,5}{E,1,7}{+,...,12}{V,2,16}		2 3 4	1 : [PARAM 50] 2 : [CALL 3] 3: [POP 2]
29	{...,...,0}{V,0,2}{=,...,5}{E,51,7}		5	4 : [ADD 1 2 51]
30	{...,...,0}{S,...,1}		6	5 : [STO 51 0]
31	{...,...,0}{S',...,...} a redução da regra 0 encerra a análise			

Gr. G2

Exemplo para a gramática da linguagem da calculadora calc:

```
/*
Calculadora de notação
infix - calc
*/

%{
#define YYSTYPE double
#include <math.h>
%}

/* BISON Declarations */
%token NUM
%left '-' '+'
%left '*' '/'
/*negation unary minus */
%left NEG
'^'/*exponentiation*/
%right '^'/*exponentiation*/
```

```
/* Grammar follows */
%%
input:  /* empty string */
      |input line
;

line:   '\n'
|exp '\n'{printf("\t%.10g\n",$1);}
;

exp:NUM {$$ = $1;}
    |exp '+' exp {$$ = $1 + $3;}
    |exp '-' exp {$$ = $1 - $3;}
    |exp '*' exp {$$ = $1 * $3;}
    |exp '/' exp {$$ = $1 / $3;}
    |'-' exp %prec NEG {$$ = -$2;}
    |exp '^' exp {$$ = pow($1,$3);}
    |'(' exp ')' {$$ = $2;}
;
%%
```



Transformação da gramática para tirar a ambiguidade e definir a precedência e associatividade diretamente pelas regras.

```
/* Grammar follows */
%%
input:  /* empty string */
      |input line
;

line:   '\n'
|exp '\n'{printf("\t%.10g\n",$1);}
;

exp:NUM {$$ = $1;}
    |exp '+' exp {$$ = $1 + $3;}
    |exp '-' exp {$$ = $1 - $3;}
    |exp '*' exp {$$ = $1 * $3;}
    |exp '/' exp {$$ = $1 / $3;}
    |'-' exp %prec NEG {$$ = -$2;}
    |exp '^' exp {$$ = pow($1,$3);}
    |'(' exp ')' {$$ = $2;}
;
%%
```

Regras da Gramatica e ações semânticas

```
S    → Stm; S
S    → Stm
Stm → id = E
Stm → print ( E )
E    → E + T
     | E - T
     | T
T    → T * F
      | T / F
      | F
F     → ( E )
      | _V
      | _N
```

Léxico `LpDaG6L.txt` deve ser renomeado para `LpDaG6.l`

```
%{
#include <stdio.h>
#include "LpDaG6.tab.h"
%}
newline             \n
tab             \t
alfa            [A-Za-z_]
digit           [0-9]
number          ({digit}+)
realnumber      {digit}*"."{digit}+
char            '[^"\n]'{1,4}
string          \"[^"]*\"
ident           {alfa}({alfa}|{digit})*
comment                 "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
separator       (newline|tab|" ")
invalid         [ !@#$%&"?""|"~ ]
```

LpDaG6L.txt deve ser renomeado para LpDaG6.l

```
%%
"="             { return( _ATRIB    );      };
$0              { return( _EOF      );      };
"("             { return( _ABREPAR  );      };
")"             { return( _FECHAPAR );      };
"+"             { return( _MAIS     );      };
"-"             { return( _MENOS    );      };
"*"             { return( _MULT     );      };
"/"             { return( _DIVID    );      };
{number}        { yylval.i= atoi(yytext);return( _N );};
print           { return( _PRINT    );      };
{ident}         { strcpy (yylval.cadeia, yytext);return( _V );};
{separator}     |
{comment}       ;
{newline}       {
                xxlines ++;
                xxcols = 1;
                return(_L);

                };
{invalid}               { return(_ERRO}; }
```

`LpDaG6Y.txt` deve ser renomeado para `LpDaG6.y`

Sintático e
ações semânticas

```
%{
#include <stdio.h>
#include <string.h>
#include <math.h>

/* tipo de yylval, $$, $1, $2, ... */

typedef union {
       char          cadeia [20];
       int           i;
       Operador      opcode;
       int           indSimb;
       int           indQuadrupla;
       } YYSTYPE;

#define          NADA              9999
#define          FRACASSO          9998
#define          ACHOUDIFVAR       9997
```

`LpDaG6Y.txt` deve ser renomeado para `LpDaG6.y`

```
typedef  enum {      ADD,
                SUB,
                MUL,
                DIV,
                STO,
                PRNT,
                } Operador;

struct Quadrupla {
        Operador          op;
        int               operando1;
        int               operando2;
        int               operando3;
        } quadrupla [ 100 ];
int prox = 0; /* índice da prim. quádrupla disponível,
na tabela*/

typedef  enum {
                Variavel,
                Constante,
                Temporaria,
                Qualquer
                        } VarOUcteOUtempOUqqr;
```

```
typedef enum  {
              Sintatico,
              Incluir
              }SintaticoOUIncluir;
typedef enum {
              Alocar,
              Referenciar
              }AlocarOuReferenciar;
/* TabSimb: 1as 50 entradas p/simbolos do fonte
e últimas 50 p/ var. temps */
typedef union {
       char nomeVar [20];
       int  valorCte;
       } TipoReferencia;
struct TabSimb {
       TipoReferencia strOUnum;
       VarOuCteOuTempOuQqr varOUcte;
       } tabSimb [100];
int    indSimb,
indTemp;
int    topTab = 0, //inicio simbs
       topTemp = 50;//inicio temps
```

```
%%token _ATRIB _EOF _ABREPAR _FECHAPAR
_MAIS _MENOS _MULT _DIVID _N _PRINT _V
_ERRO
%% /* Regras da Gramatica e acoes semanticas */
S    : Stm; S
S    : Stm
Stm : _V _ATRIB E { $1.indSimb = incluiSimbTab($1,Variavel);
                    gera(STO,$3.indSimb,$1.indiceSimb,NADA);}
Stm : _PRINT _ABREPAR E _FECHAPAR {gera(PRINT, $3, NADA, NADA);}
E    : E _MAIS  T {$$.indSimb = temp();
                   gera (ADD, $1.indSimb, $3.indSimb, $$.indSimb);}
     | E _MENOS T {$$.indSimb = temp();
                   gera (SUB, $1.indSimb, $3.indSimb, $$.indSimb);}
     | T          {$$.indSimb = $1.indSimb; }
T    : T _MULT F  {$$.indSimb = temp();
                   gera (MULT,$1.indSimb, $3.indSimb, $$.indSimb);}
      | T _DIVID F {$$.indSimb = temp();
                    gera (DIV,$1.indSimb, $3.indSimb, $$.indSimb);}
      | F          { $$.indSimb = $1.indSimb; }
F     : _ABREPAR E _FECHAPAR {$$.indSimb = $2.indSimb;}
      | _V         { $$.indSimb = incluiSimbTab($1, Variavel);}

      | _N         { $$.indSimb = incluiSimbTab($1, Constante);}

%%
```

```
void finaliza() {
  imprimeQuad ( ); // ausente p/ razao de espaco
  imprimeTabSimb ( ); // ausente p/ razao de espaco
  imprimeFonte ( ); // ausente p/ razao de espaco
  criaObjeto( ); // ausente p/ razao de espaco
};
```


para LpDaG6.y

```
int buscaSimbTab (
YYSTYPE simb, VarOuCteOuTempOuQqr vOUc, AlocarOuReferenciar aOUr){
int k;
for (k = 0;k < topTab; k++) {
  switch (vOUc) {
    case Variavel:
     if (strcmp(simb.cadeia, tabSimb [k].strOUnum.nomeVar) == 0)
       return (k);
     break;
    case Constante:
     if (simb.i == tabSimb [k].strOUnum.valorCte)
       return(k);
     break;
   default: break;
  }; // do switch
}; // do for
if (aOUr == Alocar)||(vOUc == Constante) return (FRACASSO);
  else {//referencia de variável não encontrada, aborta !
    if (vOUc == Variavel){
       printf("\n simbolo ausente %s \n",simb.cadeia);
       exit(0);
    }; // do if Variavel
   }; // do else
}; // da função
```

LpDaG6Y.txt Pg 6
para LpDaG6.y

```
void incluiSimbTab(YYSTYPE simb,VarOuCteOuTempOuQqr vOUc){
int retorno;
retorno = buscaSimbTab (simb, vOUc, Alocar);
 if ( retorno == FRACASSO ) {//simb deve ser incluido
  if (vOUc == Variavel)
    strcpy (tabSimb [topTab].strOUnum.nomeVar, simb.cadeia);
  else
    tabSimb [topTab].strOUnum.valorCte=simb.i;//vOUc eh Constante
 tabSimb [topTab].varOUcte = vOUc;
 retorno = topTab;
 topTab++;
 } // de FRACASSO, incluido novo simbolo (var ou cte)
return ( retorno );
} // da funcao
int temp () {
char [4] nomeTemporaria;
strcpy(nomeTemporaria, "t");//prefixo
strcat(nomeTemporaria, itoa (topTemp-50);// sufixo
strcpy(tabSimb [topTemp].strOUnum.nomeVar, nomeTemporaria);
tabSimb [topTemp].varOUcte = Temporaria;
topTemp++;
return (topTemp-1);
};
```

LpDaG6Y.txt Pg 7
para LpDaG6.y

```
void gera (Operador codop,int end1,int end2,int end3){
        quadrupla [prox].op = codop;
        quadrupla [prox].operando1 = end1;
        quadrupla [prox].operando2 = end2;
        quadrupla [prox].operando3 = end3;
        prox++;}

void yyerror(const char *str)
{
        printf(stderr,"error: %s\n",str);
}
int yywrap()
{
        return 1;
}
main()
{
        yyparse();
}
```

Continua LpDaG6
comandos para geração e execução


```
...$ flex LpDaG6.l
...$ bison -d LpDaG6.y
...$ gcc lex.yy.c LpDaG6.tab.c -o LpDaG6
...$ LpDaG6
...$x = 5
...$print(x)
...$y = x + 1
...$print(y)
...$z = x*y
...$print(z)
...$print (z/2)
...$print(z-y)
...$x = 2
...$w = (z - y)/x
...$print(w)
```