Introdução a Construção de Compiladores

- Parte 2
  Análise Sintática

F.A. Vanini

IC – Unicamp

Klais Soluções

Análise Sintática

A **análise sintática** ou ***parsing*** de uma sentença **w**, numa gramática **G**, consiste em verificar se **w** $\in$ **L(G)** e em caso positivo, determinar a **estrutura sintática** de **w** em **G**.

A estrutura sintática de **w** pode ser descrita por uma **árvore de derivação** de **w** em G ou por uma **derivação canônica** de **w** em **G**.

Existem duas grandes famílias de métodos de análise sintática:

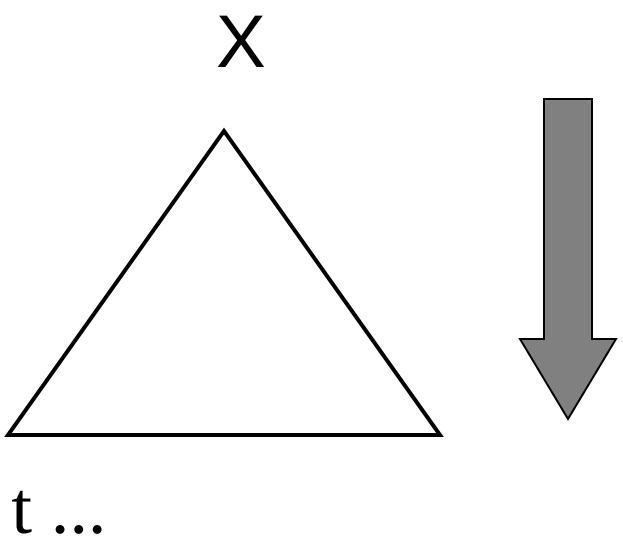
- análise sintática **descendente**
- análise sintática **ascendente**

Análise Sintática Descendente I

Um **análisador sintático descendente** procura construir a árvore de derivação da sentença(ou a derivação mais à esquerda correspondente) partindo da raiz para as folhas, lendo um símbolo por vez da sentença de entrada (da esquerda para a direita).

Nesse processo, a cada instante tem-se um símbolo X, não terminal, que corresponde à raiz da sub-árvore sendo construída e um símbolo terminal t lido da sentença de entrada. A partir do par (X,t) de símbolos, o analisador determina o conjunto C(X,t) de regras aplícáveis. O analisador deve tentar cada regra possível e pelo menos uma deve levar à sub-árvore desejada.

Análise Sintática Descendente II

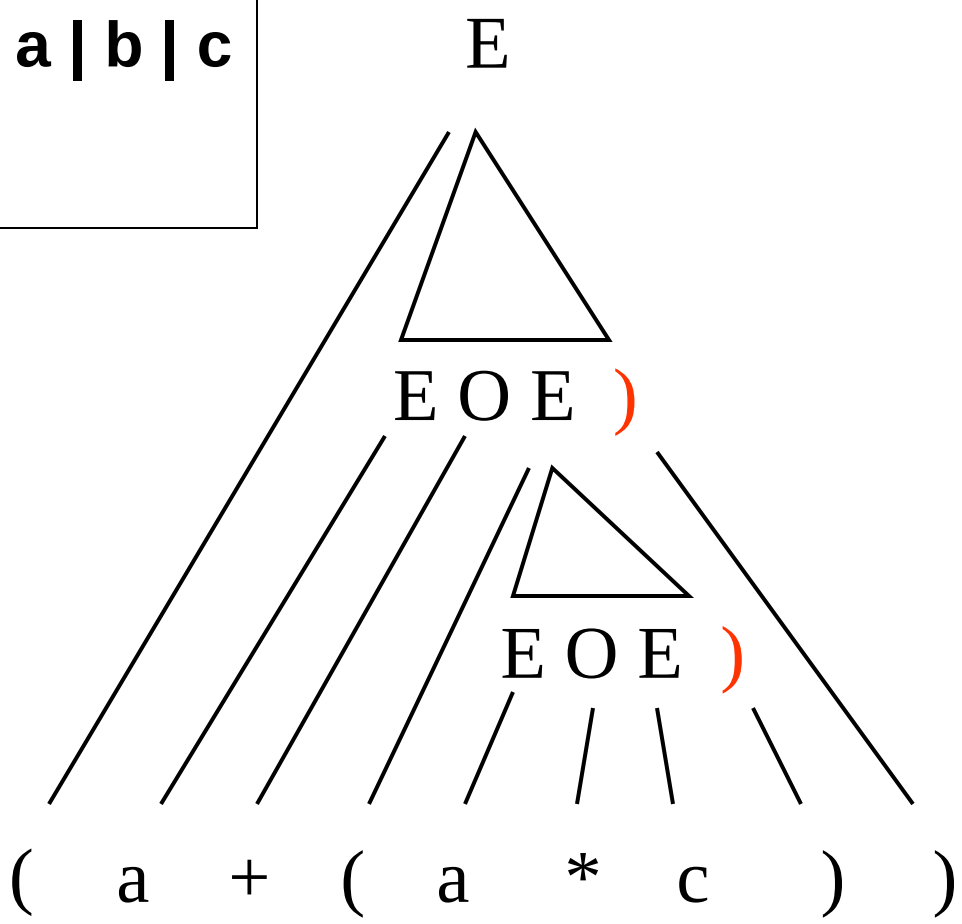


C(X,t) = conjunto de regras de P da forma X $\rightarrow \alpha$ tais que $\alpha \Rightarrow^{*} t\ \beta$

Análise Sintática Descendente III

E → (E O E) | a | b | c

O → * | +



Algoritmo Genérico

```
// reconhece a sentença derivada a partir de X, não terminal
// a variável global t contém o último símbolo lido
função booleana reconhece (X ) {
   C = conj. de regras de P da forma X  → α tais que  α ⇒* t β;
   se C é vazio então { retorne falso; }
   b = verdadeiro;
   para toda regra em C, da forma X  → α faça {
      para todo símbolo Y em  α faça {
         se Y é terminal então {
           b = b && (Y==t);  t = próximo símbolo
         } senão b = b && reconhece(Y)
      }
   }
  retorne b;
}
```

Análise Sintática Descendente sem Retrocesso

Para que a análise sintática possa ser feita sem a necessidade de retrocesso, é necessário impor restrições à gramáticas que garantam que a cada passo do processo, o conjunto C tenha apenas uma regra (em outras palavras, o par (X,t) indica uma única regra como aplicável).

As restrições impostas à gramática limitam a classe das linguagens tratáveis por este método de análise sintática. Isso no entanto não impede que o mesmo seja usado em muitas situações práticas (existem muitos compiladores comerciais que utilizam variantes desse método).

Restrições à Gramática

1- Para cada forma sentencial α

- *Inicia(α):* o conjunto de terminais que podem iniciar uma sentença derivada a partir de α.
- *Segue*(α): conjunto de terminais que podem aparecer seguindo uma sentença derivada a partir de α.

2 - Para cada regra R $\in$ P, da forma $A \rightarrow \alpha$, definimos *Prediz*(R) como:

Se $\alpha \Rightarrow^* \lambda$ então Prediz(R) = Inicia(α) $\cup$ Segue(α)

senão Prediz(R) = Inicia(α).

A restrição:

Se $A \rightarrow \alpha_1 \mid \alpha_2 .. \mid \alpha_k$, então os conjuntos *Prediz* ($A \rightarrow \alpha_i$) devem ser disjuntos entre si.

Um Exemplo I

G= { N,T,P,S }

N = { S, A, C,L, W, R, E }

T = { a, :=, if, then, else, while, do }

P = { $S \rightarrow A \mid C \mid L$, $A \rightarrow a := E$,

$C \rightarrow$ if E then S else S | if E then S,

$L \rightarrow W \mid R$, $W \rightarrow$ while E do S,

$R \rightarrow$ do S while E, $E \rightarrow a \mid (E + E)$

}

Um Exemplo II

Regra	Prediz
$S \rightarrow A$	{ a }
$S \rightarrow C$	{ if }
$S \rightarrow L$	{ while, repeat }
$C \rightarrow$ if E then S else S	{ if } *viola a regra*
$C \rightarrow$ if E then S	{ if } "
$L \rightarrow W$	{ while }
$L \rightarrow R$	{ repeat }
$E \rightarrow a$	{ a }
$E \rightarrow$ (E+E)	{ (}

Outro Exemplo I

Gramática “clássica” para expressões aritméticas, que leva em conta a “precedência” convencional entre os operadores “+” e “*” (para verificar, basta construir a árvore sintática de expressões como “a+b*c”, “a*b+c, etc.)

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a \mid b \mid c$

Outro Exemplo II

Regra	Prediz
$E \rightarrow E+T$	{ (, a, b, c } (viola)
$E \rightarrow T$	{ (, a, b, c } “
$T \rightarrow T*F$	{ (, a, b, c } “
$T \rightarrow F$	{ (, a, b, c } “
$F \rightarrow (E)$	{ (}
$F \rightarrow a$	{ a }
$F \rightarrow b$	{ b }
$F \rightarrow c$	{ c }

Outro Exemplo III

Nessa gramática, uma expressão é definida como uma sequência de “soma de termos”, definida pelas regras

$E \rightarrow E\mathbf{+}T \mid T$ que levam a derivações do tipo

$E \Rightarrow^{+} T$

$E \Rightarrow^{+} T \mathbf{+} T$

$E \Rightarrow^{+} T \mathbf{+} T \mathbf{+} T \ldots \mathbf{+} T$

Essas regras podem ser reescritas como

$E \rightarrow T\ T'$

$T' \rightarrow + E \mid \lambda$

Que geram o mesmo tipo de sequência e obedecem às restrições.

Outro Exemplo I

A mesma transformação é aplicável às regras
$T \rightarrow T^{*} F \mid F$ que levam a derivações do tipo

$T \Rightarrow^{+} F$
$T \Rightarrow^{+} F * F$
$T \Rightarrow^{+} F * F * F \ldots * F$

Essas regras podem ser reescritas como

$T \rightarrow F\ F'$
$F' \rightarrow * T \mid \lambda$

Que geram o mesmo tipo de sequencia e obedecem às restrições.

Outro Exemplo III

A “gramática transformada” seria a seguinte:

$E \rightarrow T\ T'$
$T' \rightarrow +\ E \mid \lambda$
$T \rightarrow F\ F'$
$F' \rightarrow *\ T \mid \lambda$
$F \rightarrow (\ E\ ) \mid a \mid b \mid c$

Essa gramática é equivalente à original (gera a mesma linguagem) e obedece às restrições necessárias à análise descendente sem retrocesso.

Notação Estendida I

A “gramática transformada” do exemplo anterior introduz novos não terminais que implicam em “ramos” adicionais na árvore de derivação das sentenças. Durante a análise sintática, isso implica em passos adicionais.

Para evitar a criação desses símbolos não terminais e tornar a gramática mais compacta, costuma-se usar uma notação estendida, usando nas regras as construções abaixo:

- [α] para indicar que a sequência α pode ocorrer ou não (zero ou uma vez)
- { α } para indicar que a sequência α pode ocorrer zero ou mais vezes

Notação Estendida II

A “gramática transformada” do exemplo anterior, usando a “notação estendida” ficaria da seguinte forma:

$E \rightarrow T \{ + E \}$
$T \rightarrow F \{ * F \}$
$F \rightarrow (E) \mid a$

Construção do Analisador I

1 – para cada símbolo não terminal da gramática construímos um procedimento responsável pela análise sintática do trecho da sentença de entrada derivado desse símbolo não terminal.

2 – quando um procedimento associado a um símbolo não terminal é chamado, o “próximo símbolo de entrada” já deve ter sido lido.

Construção do Analisador II

3 – para cada símbolo não terminal A, que aparece numa regra da forma $A \rightarrow \alpha_1 \mid \alpha_2 .. \mid \alpha_k$, construímos o procedimento A, com a seguinte estrutura:

```
procedimento A() {
   caso símbolo lido esteja em
   Prediz (A → α1) : " reconhece α1 ";
   Prediz (A → α2) : " reconhece α2 ";
   Prediz (A → αk) : " reconhece αk ";
   outros:  erro
}
```

Construção do Analisador III

4 – para cada sentença $\alpha = X_1X_2..X_k$, que aparece no lado direito de uma regra, a operação "reconhece α" é traduzida para

```
{
   "reconhece X1 ";
   "reconhece X2 ";
   ...
   "reconhece Xk ";
}
```

Construção do Analisador IV

5 – se X é um símbolo terminal, a operação "reconhece X" é traduzida para

```
" se simbolo (lido==X) {
     leia prox. Simbolo;
   } senão erro "
```

6 – se X é um símbolo não terminal, "reconhece X" é traduzida para a chamada ao procedimento correspondente.

Se $\alpha = \lambda$ "reconhece α" é traduzido para o comando vazio .

Construção do Analisador V

7 – se X é uma frase do tipo [α], “reconhece X” é traduzida para

```
se (símbolo lido ∈ inicia(α) ) {
   “reconhece α”
 }
```

8 - se X é uma frase do tipo { α }, “reconhece X” é traduzida para

```
enquanto(símbolo lido ∈ inicia(α))
{
 “reconhece α”
}
```

Um Exemplo I

Para a nossa gramática de expressões:

$E \rightarrow T \{ + E \}$

$T \rightarrow F \{ * F \}$

$F \rightarrow (E) \mid a$

```
procedimento E() {
    T;
    enquanto (simbolo lido == "+" ) {
        leia próximo simbolo;
        E;
    }
}
```

Um Exemplo II

```
procedimento T() {
     F;
     enquanto (simbolo lido == "*" ) {
         leia próximo simbolo;
         T
     }
  }

 procedimento F() {
    caso simbolo lido seja
       ´(´ : {
                leia proximo simbolo; E;
                se simbolo lido == ´)´ leia próximo simbolo
                senão erro
             }
       ´a´, ´b´,´c´: leia proximo simbolo;
    outro: erro
 }
```

Análise Ascendente I

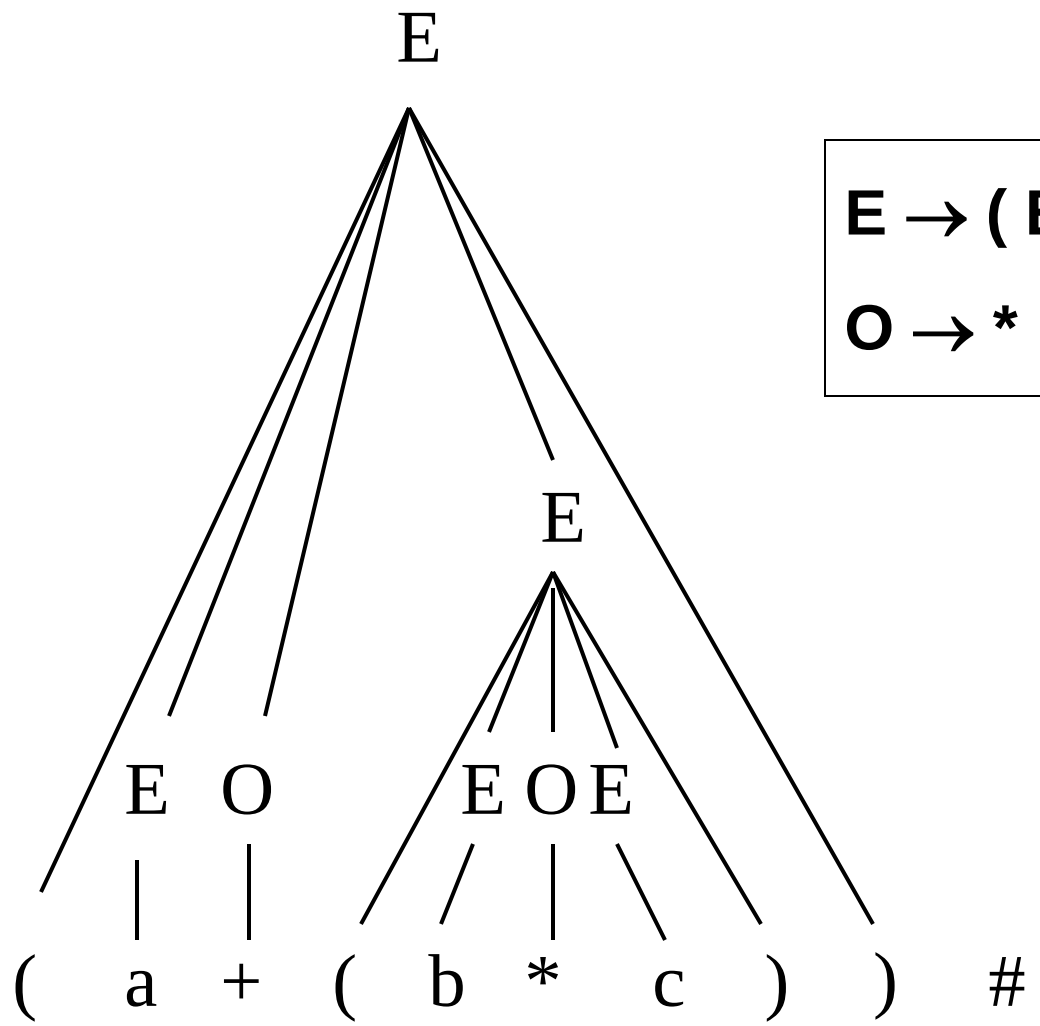
Os algoritmos de análise ascendente procuram, da esquerda para a direita, trechos da sentença sendo analisada, trechos que possam corresponder a um lado direito de produção para substituir pelo não terminal correspondente.

Essa operação é chamada de *redução* e o lado direito encontrado forma sentencial sendo analisada, é chamado de *redutendo*.

O processo se encerra quando toda a forma sentencial é reduzida para o símbolo inicial da gramática.

A árvore sintática é construída das folhas para a raíz, daí o nome *ascendente*.

Análise Ascendente II



E → (E O E) | a | b | c

O → * | +

Algoritmo Genérico

- G = { N, T, P, S } é alterada para G = { N', T', P', S'} onde N' = N U {S'} T' = T U { # } e
  S'= S U { S' $\rightarrow$ #S # }

```
procedimento análise_ascendente () {
  empilhe ( # );
  leia próximo símbolo;
  repita {
     caso símbolo lido e o topo da pilha indiquem {
       "avançar" : { empilhe simbolo lido; leia prox. Simbolo }
       " reduzir"  : {
                          "identificar lado direito e  substituir
                          pelo não terminal à esq. da regra
                          correspondente"
                     }
        outra coisa:  "erro"
     }
  } até que conteúdo da pilha seja igual a #S#
}
```

Precedência Simples I

Para uma gramática G = { N, T, P, S} definimos as relações $\lessdot$, = e $\gtrdot$, entre os símbolos de $N \cup T$, da seguinte forma:

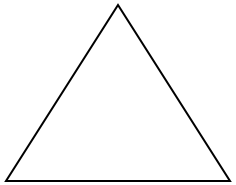
- X = Y se existe em P uma regra da forma $Z \rightarrow \alpha XY\beta$
- X $\lessdot$ Y se existe Z tal que X = Z e $Z \Rightarrow^{+} Y\gamma$
- X $\gtrdot$ Y se Y é um símbolo terminal e existe Z e W tais que

  $Z = W,\ Z \Rightarrow^{+} \alpha X$ e $W \Rightarrow * Y\beta$

As relações $\lessdot$, = e $\gtrdot$ não simétricas nem transitivas.

Precedência Simples II

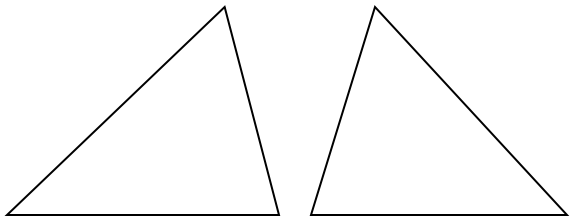
X = Z



Y. . .

⇨ X < Y

Z = W



. . . X Y . . .

⇨ X > Y e X > W

Precedência Simples III

Essas tres relações são usadas pelo analisador para conduzir o processo de análise sintática:

Supondo que X é o símbolo no topo da pilha e Y é o símbolo lido,

- X = Y significa que Y pode aparecer ao lado de X num redutendo. A ação, nesse caso é “avançar”
- X < Y significa que X e Y não fazem parte do mesmo redutendo, mas Y inicia um novo redutendo. Ação: “avançar”.
- X > Y significa que X é o último símbolo do redutendo na pilha. Ação: “reduzir”.

Precedência Simples IV

G é uma gramática de precedência simples se

1. As relações <, = e >, para os símbolos de G são disjuntas.
2. Não existem duas regras com o mesmo lado direito.

Algoritmo I

```
analise_por_precedencia_simples() {
     empilhe(#); leia proximo simbolo;
     repita {
        caso (
                precedência entre símbolo no topo da
                pilha e o símbolo lido
              )seja
        {
           <, = :    {
                        empilhe símbolo lido;
                        leia proximo simbolo
                     }
            > :       reduza;
            outro :   erro
        }
   } até que a pilha contenha #S#;
}
```

Algoritmo II

Para a operação de redução, é necessário identificar o redutendo, que deve corresponder a uma sequencia de símbolos no topo da pilha.

O redutendo é delimitado pelas relações entre os símbolos na pilha (... $< X_1 = X_2 .. = X_k >$ simbolo lido), o que facilita a operação.

Eventualmente o *redutendo* identificado pode não corresponder a um lado direito de produção. Isso significa que a sentença de entrada está incorreta.

Construção das Relações I

As relações de precedência simples são construídas a partir de “relações primitivas” entre os símbolos extraídas da gramática.

Essa construção pode ser mecânica, através de operações com relações (como aquelas usadas em *álgebra relacional*) ou manualmente, a partir da definição das mesma.

Construção das Relações II

A relação = é facilmente obtida procurando na gramática pares de símbolos que aparecem juntos num lado direito.

Por exemplo, para a nossa gramática de expressões, $E \rightarrow (E O E) \mid a \mid b \mid c \quad O \rightarrow * \mid +$ a relação = teria os seguintes pares:

(= E E = O O = E e E =)

As demais relações podem ser construídas a partir dela.

Construção das Relações III

Para cada par (X,Y), construímos uma tabela com duas colunas.

Se Y for um símbolo não terminal, colocamos na coluna correspondente os símbolos que podem iniciar uma forma sentencial derivada a partir de X.

(	E
	(
	a
	b
	c

X < Z para todo Z na coluna da direita.

(< (, (< a, (< b, (< c

Construção das Relações IV

Se X for um símbolo não terminal, colocamos na coluna correspondente os símbolos que podem terminar uma forma sentencial derivada a partir de X.

E	O
a	+
b	*
c	
)	

X > Y para todo X na esquerda e y na direita

E > +, E > *, a>+, a>*, b>+, b>*, c>+, c> *,)>+,)>*

Construção das Relações V

As relações de precedência para a nossa gramática podem ser representadas numa tabela:

	E	O	(	)	+	*	a	b	c
E		=		=	<	<			
O	=		<				<	<	<
(	=								
)				>	>	>			
+			<				<	<	<
*****			<				<	<	<
a				>	>	>			
b				>	>	>			
c				>	>	>			

Precedência Simples

A análise por precedência simples tem algumas desvantagens:

- As tabelas crescem com o quadrado no número de símbolos da gramática
- A construção manual da tabela é trabalhosa e sujeita a erros
- O método impõe sérias restrições à gramática

Este método, no entanto, tem a grande virtude de servir de base para a família de métodos LR, mais abrangentes e mais eficientes.

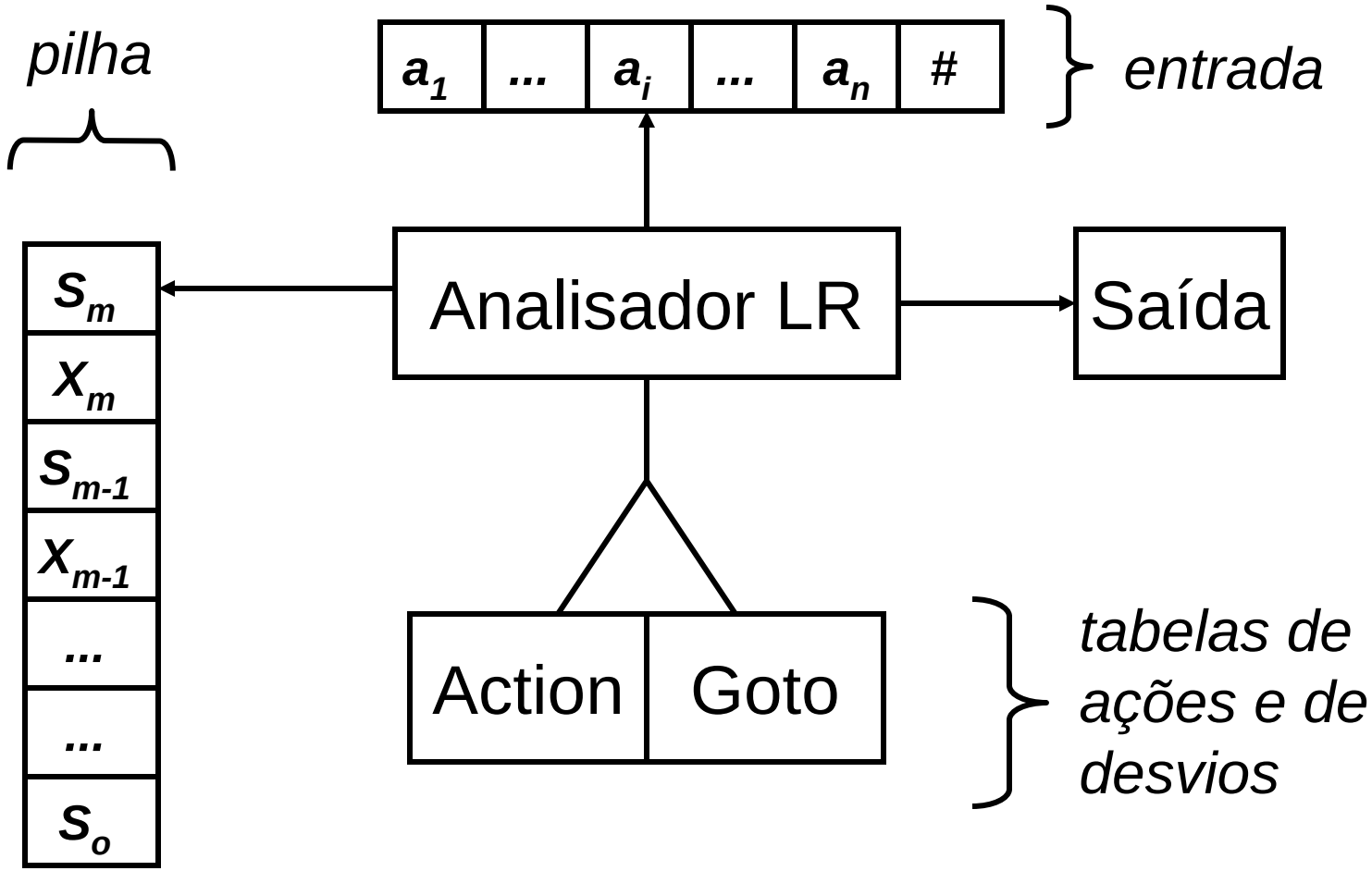
Análise LR - I

Os métodos LR levam em conta que durante a análise ascendente, as “sentenças” na pilha constituem uma linguagem regular.

Os métodos LR usam a informação de “estado do topo da pilha” para simplificar e acelerar o processo de análise sintática.

Em virtude de tomar as decisões com base no “estado”, algumas restrições à gramática podem ser relaxadas, tornando o método potencialmente mais abrangente.

Análise LR - II



Algoritmo Genérico

```
analise_LR() {
  leia próximo símbolo;  empilhe estado inicial;
  repita {
    S ← estado no topo da pilha; A ← action[S,simbolo lido];
    caso A seja {
      aceita: FIM;
      empilha S': {   empilha simbolo lido; empilha S';
                        leia próximo símbolo
                      }
      reduz R:  {   /* R é uma regra da forma  X → β */
                      desempilhe 2* |β| símbolos;
                      S'← estado no topo da pilha;
                      empilhe(X); empilhe(goto[S',X]);
                    }
      outro:       ERRO;
    }
 } até FIM;
```

Um Exemplo

Usando a gramática para expressões:

(1) $E \rightarrow E + T$
(2) $E \rightarrow T$
(3) $T \rightarrow T * F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow id$

Um Exemplo (cont.)

	Id	+	*	(	)	$		E	T	F
0	S4			S4				1	2	3
1		S6				aceita				
2		R2	S7		R2	R2				
3		R4	R4		R4	R4				
4	S5			S4				8	2	2
5		R6	R6		R6	R6				
6	S5			S4					9	3
7	S5			S4						10
8		S6			S11					
9		R1	S7		R1					
10		R3	R3		R3	R3				
11		R5	R5		R5	R5				

Ferramentas I

O analisador sintático ascendente é formado basicamente pelo algoritmo de análise sintática e por um conjunto de tabelas.

O algoritmo depende unicamente da variante do método de análise sintática utilizado.

As tabelas dependem não só do método como também da gramática.

Ferramentas II

Sendo assim, uma vez escolhido o método de análise sintática, a implementação do algoritmo pode ser a mesma para todas as gramáticas (que atendam às restrições impostas pelo método).

A construção manual dessas tabelas é trabalhosa e sujeita a erros.

A construção dessas tabelas através de ferramentas dedicadas não só é possível como também torna possível a “construção automática” do analisador sintático.

Ferramentas III

Essas ferramentas recebem como entrada a definição da gramática e produzem como saída as tabelas e uma implementação do algoritmo de análise sintática adaptada à gramática (tamanhos, nomes, etc...).

Elas produzem também a rotina básica para o *processo de tradução* desencadeado pela análise sintática.

Ferramentas IV

As ferramentas mais comuns são aquelas baseadas em variantes do método LR, pelo fato de serem mais eficientes e por impor poucas restrições à gramática.

Embora não sejam tão comuns, existem também ferramentas baseadas em métodos descendentes.

O Processo de Tradução I

O analisador sintático determina a estrutura da sentença de entrada (ou do programa).

A partir dessa estrutura é possível traduzir a sentença (ou o programa) para uma outra notação. Essa tradução constitui uma parte importante no processo de compilação.