



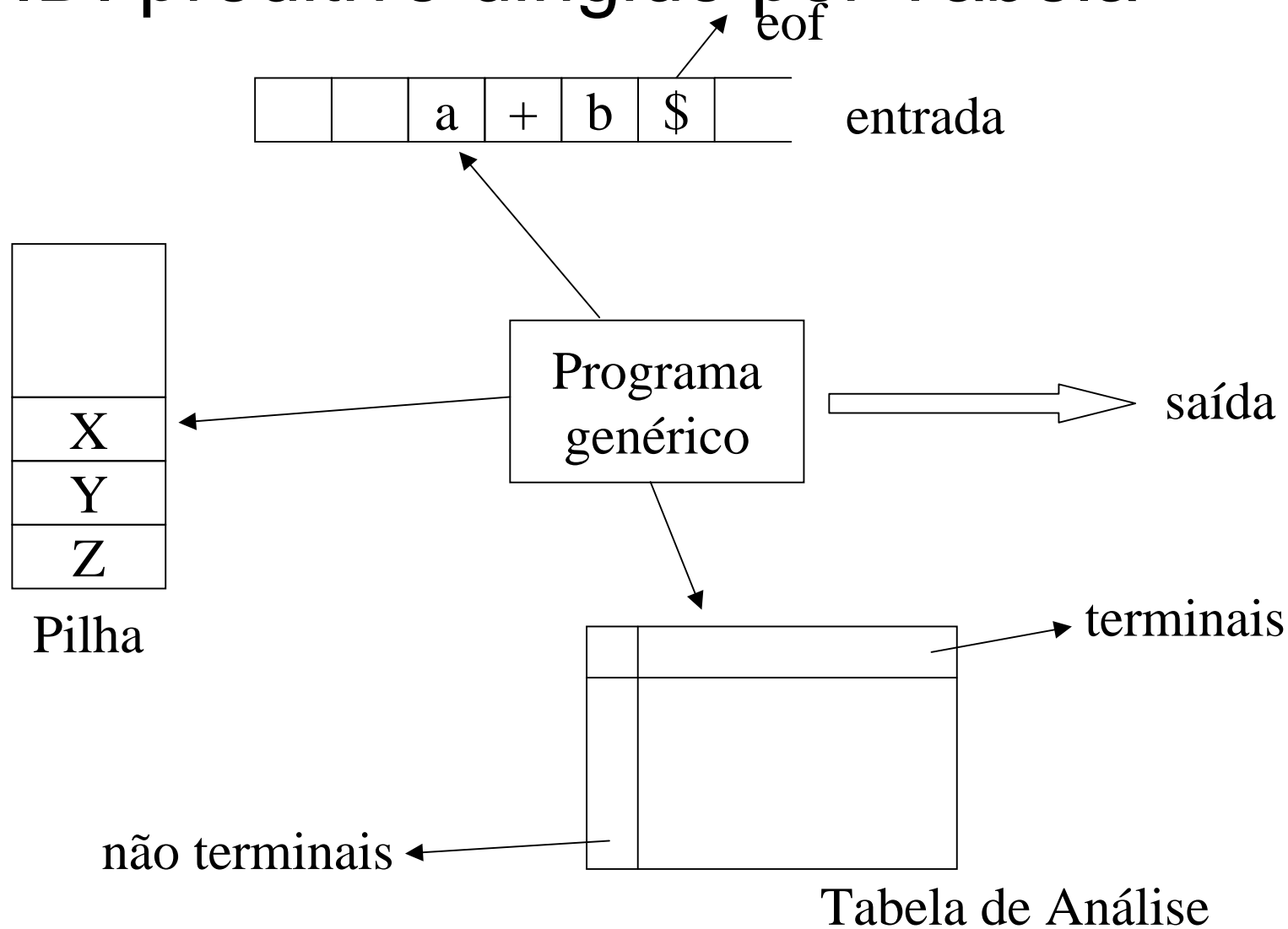
Análise Sintática II: Analisadores Descendentes Preditivos



A.S.D. Preditivos

- Duas formas de implementação:
 - **dirigido por tabela** → uso de uma pilha P para armazenar a parte da forma sentencial esquerda ainda não analisada (fácil de automatizar). São eficientes, pois gerenciam a pilha de ativação explicitamente.
 - **com procedimentos recursivos** (fácil de implementar manualmente; produz código mais legível que facilita a manutenção)

A.S.D. preditivo dirigido por Tabela



Procedimento ASD_dirigido_por_tabela

$P[1] \leftarrow S$; $i \leftarrow 1$; $\text{termino} \leftarrow \text{false}$;
 $\text{simbolo} \leftarrow \text{analex}(S)$;
repita

$X \leftarrow P[i]$;

se X é terminal **então**

se $X = \text{simbolo}$ **então**

$[\text{simbolo} \leftarrow \text{analex}(S)$;

$i \leftarrow i-1$;

se $i=0$ **então** $\text{termino} = \text{true}$]

senão ERRO

senão **se** $\text{tabela}[X, \text{simbolo}] \neq ''$ **então**

$[P[i] \leftarrow X_n$; $P[i+1] \leftarrow X_{n-1}$; ... $P[i+n-1] \leftarrow X_1$;

$i \leftarrow i+n-1$]

senão ERRO

até termino

se $\text{simbolo} \neq \$$ **então** ERRO

senão ACEITAR

desempilha

pilha vazia

troca o topo pela produção,
empilha inversamente e
avança o topo



Construção da Tabela de Análise

Para se fazer a análise de forma determinística e automática, é necessário existir uma tabela que nos dê exatamente a regra que deverá ser aplicada quando se tem um não terminal e um terminal a ser reconhecido.

Entrada: Gramática G

Saída: Tabela de Análise M



Algoritmo para Construção da Tabela de Análise M

Para cada produção $A \rightarrow \alpha$ faça:

1. Para cada terminal $a \in \text{First}(\alpha)$ coloque $A \rightarrow \alpha$ na entrada $M[A, a]$
2. Se $\lambda \in \text{First}(\alpha)$ coloque $A \rightarrow \alpha$ para $M[A, b]$ para cada $b \in \text{Follow}(A)$
3. Se $\lambda \in \text{First}(\alpha)$ e $\$ \in \text{Follow}(A)$ coloque $A \rightarrow \alpha$ para $M[A, \$]$



Calculo das relações First e Follow

FIRST

1. Se x é terminal então
 $\text{First}(X) = \{X\}$
2. Se $X \rightarrow \lambda$ é produção
então coloque λ no
 $\text{First}(x)$
3. Se X é não terminal e $X \rightarrow Y_1 Y_2 \dots Y_n$ é
produção adicione
 $\text{First}(Y_i)$ para $\text{First}(X)$ se
os precedentes Y_j s
contém λ em seus First

FOLLOW

1. $\text{Follow}(S)$ contém $\$$
2. Para $A \rightarrow \alpha B \beta$ tudo em
 $\text{First}(\beta)$ exceto λ vai para
 $\text{Follow}(B)$
3. Para $A \rightarrow \alpha B$ ou $A \rightarrow \alpha B \beta$
onde $\text{First}(\beta)$ contém λ ,
 $\text{Follow}(B)$ contém tudo
que está no $\text{Follow}(A)$



Construção da Tabela de Análise (1)

$S \rightarrow {}^1AS \mid {}^2BA$

$A \rightarrow {}^3aB \mid {}^4C$

$B \rightarrow {}^5bA \mid {}^6d$

$C \rightarrow {}^7c$

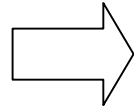
Seja a gramática

$S \rightarrow AS \mid BA$

$A \rightarrow aB \mid C$

$B \rightarrow bA \mid d$

$C \rightarrow c$



First	
S	a, c, b,
A	d
B	a, c
C	b, d
	c

1	A	S
2	B	A
3	a	B
4	C	
5	b	A
6	d	
7	c	

Tabela de Análise

NT ^T	a	b	c	d
S	1	2	1	2
A	3		4	
B		5		6
C			7	

Geração da Tabela de Análise JFLAP

JFLAP: <untitled2>

File Input Convert Help

Editor Convert to PDA (LL) Build LL(1) Parse

Do Selected Do Step Do All Next Parse

S → AS
 S → BA
 A → aB
 A → C
 B → bA
 B → d
 C → c

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
A	{ a, c }	{ d, \$, a, c, b }
B	{ d, b }	{ d, \$, a, c, b }
C	{ c }	{ d, \$, a, c, b }
S	{ d, a, c, b }	{ \$ }

	a	b	c	d	\$
A	aB		C		
B		bA		d	
C			c		
S	AS	BA	AS	BA	

Análise de abcdad

i	P	X	cadeia	Regra escolhida
1	S	S	abcdad	$S \rightarrow AS$
2	SA	A	abcdad	$A \leftarrow aB$
3	SBa	a	abcdad	---
2	SB	B	bcdad	$B \leftarrow bA$
3	SAb	b	bcdad	---
2	SA	A	cdad	$A \leftarrow C$
2	SC	C	cdad	$C \leftarrow c$
2	Sc	c	cdad	---
1	S	S	dad	$S \leftarrow BA$
2	AB	B	dad	$B \leftarrow d$
2	Ad	d	dad	---
1	A	A	ad	$A \leftarrow aB$
2	Ba	a	ad	---
1	B	B	d	$B \leftarrow d$
1	d	d	d	---
∅			\$	

Gramática ambígua

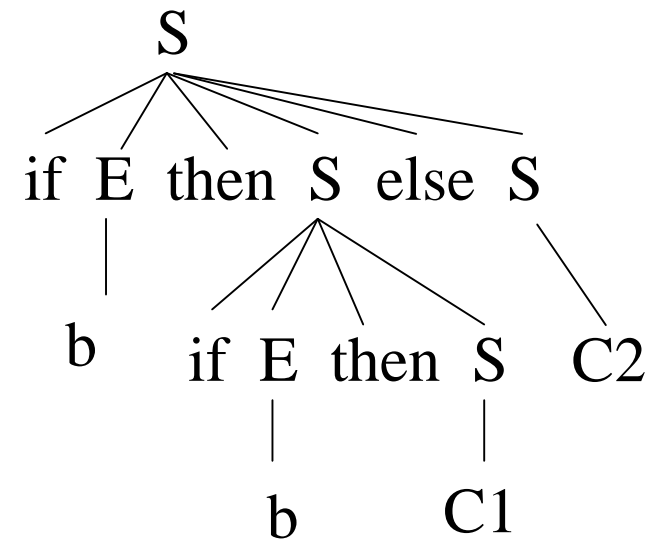
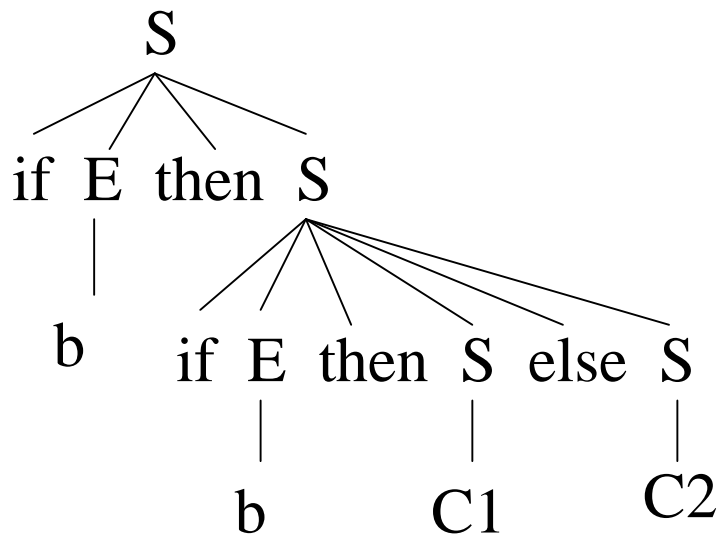
$S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid a$

$E \rightarrow b$

Cadeia

if b then if b then C1 else C2

Duas derivações possíveis:





Exemplo de gramática para a qual é preciso calcular Follows

Reescrevendo para tirar ambigüidade:

$$S \rightarrow \text{if } E \text{ then } S S' \mid a$$
$$S' \rightarrow \text{else } S \mid \lambda$$
$$E \rightarrow b$$

$S \rightarrow^1 \text{if } E \text{ then } S S' \mid^2 a$

$S' \rightarrow^3 \text{else } S \mid^4 \lambda$

$E \rightarrow^5 b$

Entretanto, mesmo
com a reescrita a
gramática tem
entradas
multidefinidas

$\text{First}(S) = \{\text{if}, a\}$

$\text{First}(S') = \{\text{else}, \lambda\}$

$\text{First}(E) = \{b\}$

$\text{Follow}(S) = \{\text{else}, \$\}$

$\text{Follow}(S') = \{\text{else}, \$\}$

$\text{Follow}(E) = \{\text{then}\}$

Tabela de Análise

NT	if	then	else	a	b	\$
S	1			2		
S'			3/4			4
E					5	

topo da
pilha

programa
sendo lido

Problema na tabela: ambigüidade na aplicação das regras para o else

→ solução: exclui-se uma das regras e adota-se a regra informal de associar o else com o if mais próximo

Tabela de Análise

NT	if	then	else	a	b	\$
S	1			2		
S'			3			4
E					5	

excluiu-se a regra 4

Geração da Tabela de Análise JFlap

Parse table complete, but has ambiguity.

	FIRST	FOLLOW
C	{ i, a }	{ \$, e }
E	{ b }	{ t }
R	{ λ , e }	{ \$, e }

	a	b	e	f	h	i	l	n	o	s	t	\$
C	a					ifEthenCR						
E		boolean										
R			λ elseC									λ

Windows taskbar: Iniciar, Perl, Sce185_2005_A_B, 1:xapacura.icmc.usp.br ..., 2:xapacura.icmc.usp.br ..., JFLAP : <untitled1>, 09:07



Exercício

- Construção da Tabela de Análise (1):
 - $E \rightarrow TE'$
 - $E' \rightarrow v TE' \mid \lambda$
 - $T \rightarrow FT'$
 - $T' \rightarrow \& FT' \mid \lambda$
 - $F \rightarrow \text{not } F \mid \text{id}$

Exercícios

LL(1) = Left to right, Left-most derivation, 1 símbolo look-ahead

29. As gramáticas abaixo são LL(1)? Transforme as que não são.

(a) $S \rightarrow ABc$
 $A \rightarrow a|\lambda$
 $B \rightarrow b|\lambda$

(b) $S \rightarrow Ab$
 $A \rightarrow a|B|\lambda$
 $B \rightarrow b|\lambda$

(c) $S \rightarrow ABBA$
 $A \rightarrow a|\lambda$
 $B \rightarrow b|\lambda$

(d) $S \rightarrow aSe|B$
 $B \rightarrow bBe|C$
 $C \rightarrow cBe|d$

LL(1): definição

1. - para todo símbolo $A \in N$ com as regras:
$$A \rightarrow \underline{X_1}\alpha_1 \mid \underline{X_2}\alpha_2 \mid \dots \mid \underline{X_n}\alpha_n$$

onde $X \in V$, temos que:

- $\text{First}(X_i)$ são disjuntos dois a dois:
 - $\text{First}(X_k) \cap \text{First}(X_j) = \{ \}$ para $\forall k, j \in \{1, 2, \dots, n\}$ com $k \neq j$

2. Para toda produção $A \rightarrow \alpha \mid \beta$

- Se $\beta \Rightarrow^* \lambda$, então α não deriva cadeias começando com um terminal no Follow (A), isto é, o First (α) é diferente do Follow (A).
- O mesmo vale para α : se $\alpha \Rightarrow^* \lambda$, então First (β) é diferente de Follow (A).

a) **LL(1)**, pois passa na regra 1 e na regra 2: follow (A) é diferente de First (a) e Follow (B) é diferente de first (b)

S	→	ABc
A	→	a
A	→	λ
B	→	b
B	→	λ

	FIRST	FOLLOW
A	{ λ , a}	{b, c}
B	{ λ , b}	{c}
S	{b, c, a}	{ $\$$ }

b) **Não é LL(1)**, pois as regras para A possuem First iguais

S	→	Ab
A	→	a
A	→	B
A	→	λ
B	→	b
B	→	λ

Parse table complete, but has ambiguity.

	FIRST	FOLLOW
A	{ λ , b, a}	{b}
B	{ λ , b}	{b}
S	{b, a}	{ $\$$ }

c) **Não é LL(1)**, pois não passa na regra 2: Follow (A) tem elemento no conjunto first (a) E Follow (B) tem elemento no first (b)

S	→	ABBA
A	→	a
A	→	λ
B	→	b
B	→	λ

Parse table complete, but has ambiguity.

	FIRST	FOLLOW
A	{ λ , a}	{b, \$, a}
B	{ λ , b}	{b, \$, a}
S	{ λ , b, a}	{\$}

d) **LL(1)**, pois passa na regra 1. A regra 2 não se aplica

S	→	aSe
S	→	B
B	→	bBe
B	→	C
C	→	cBe
C	→	d

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
B	{ d, b, c }	{ e, \$ }
C	{ d, c }	{ e, \$ }
S	{ d, b, c, a }	{ e, \$ }

Reescrita de b): remover λ com Jflap

JFLAP : <untitled1 >

File Input Convert Help

Editor Lambda Removal

S	→	Ab
A	→	a
A	→	B
A	→	λ
B	→	b
B	→	λ

b) Não é LL(1), nem depois da reescrita, mas é LL(2)

Do Step Do All Proceed Export

Lambda removal complete.
"Proceed" or "Export" available.
Set that derives lambda: [A, B]

Delete Complete Selected

S	→	Ab
A	→	a
A	→	B
B	→	b
S	→	b

Editor Build LL(1) Parse

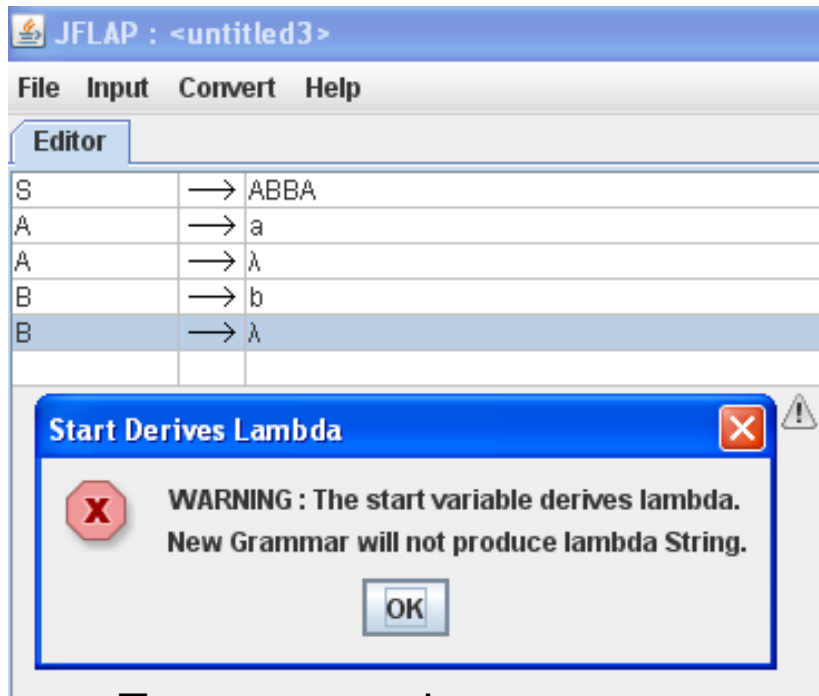
Do Selected Do Step Do All Next

S	→	Ab
A	→	a
A	→	B
B	→	b
S	→	b

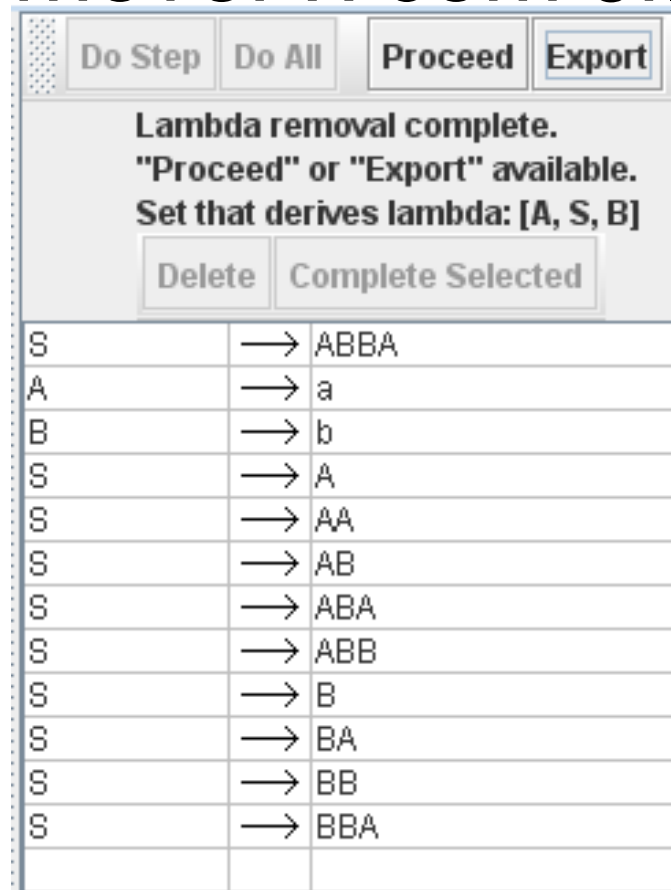
Parse table complete, but has ambiguity.

	FIRST	FOLLOW
A	{ b, a }	{ b }
B	{ b }	{ b }
S	{ b, a }	{ \$ }

Reescrita de c): remover λ com Jflap



Temos que colocar manualmente



c) Não é LL(1), nem depois da reescrita

Parse table complete, but has ambiguity.

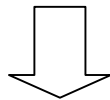
	FIRST	FOLLOW
A	{ a }	{ b, \$, a }
B	{ b }	{ b, \$, a }
S	{ λ , b, a }	{ \$ }



A.S.D. preditiva com procedimentos recursivos

- Outra maneira de implementar é usar procedimentos recursivos
 - cria-se um procedimento para cada não terminal da gramática
 - cada procedimento deve ser responsável por reconhecer a parte da sentença de entrada derivada dele

Exemplo de gramática que foi reescrita na notação EBNF

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

$$E \rightarrow (+ T \mid - T \mid T) \{+ T \mid - T\} \rightarrow [+ \mid -] T \{ (+ \mid -) T \}$$
$$T \rightarrow F \{ * F \mid / F \} \rightarrow F \{ (* \mid /) F \}$$
$$F \rightarrow a \mid b \mid (E)$$



```
program
```

```
begin
```

```
    simbolo ← analex(S);
```

```
    E;
```

```
    se (terminou_cadeia)
```

```
        então SUCESSO
```

```
        senão ERRO
```

```
end;
```

```
procedure E;
```

```
begin
```

```
    if simbolo in [+,-] then simbolo ← analex(S);
```

```
    T;
```

```
    while simbolo in [+,-] do
```

```
        begin
```


```
            simbolo ← analex(S);
```

```
            T;
```

```
        end;
```

```
end;
```

$E \rightarrow (+ T \mid - T \mid T) \{+ T \mid - T\} \rightarrow [+ \mid -] T \{ (+ \mid -) T \}$ $T \rightarrow F \{ * F \mid / F \} \rightarrow F \{ (* \mid /) F \}$ $F \rightarrow a \mid b \mid (E)$
--



```
procedure T;  
begin
```

```
    F;  
    while simbolo in [*,/] do  
        begin
```

```
            simbolo ← analex(S);  
            F;
```

```
        end;
```

```
end;
```

```
procedure F;  
begin
```

```
    case simbolo of
```

```
        a, b: simbolo ← analex(S);  
        (: begin
```

```
            simbolo ← analex(S);  
            E;
```

```
            if simbolo = ) then simbolo ← analex(S)  
            else ERRO(" esperado");
```

```
        end
```

```
    else ERRO("a, b ou ( esperado");
```

```
end;
```

```
E → (+ T | - T | T) {+ T | - T} → [+ | - ] T { (+ | -) T }  
T → F { * F | / F } → F { (* | /) F }  
F → a | b | (E)
```



ASD preditiva com procedimentos recursivos

- Método formal para gerar os procedimentos
 - **Regras de transformação**: mapeamento das regras de um não terminal em grafos sintáticos
 - **Regras de tradução**: mapeamento dos grafos em procedimentos

- Exemplo

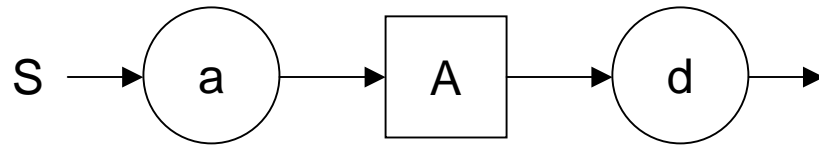
$\langle S \rangle ::= a \langle A \rangle d$

$\langle A \rangle ::= c \langle A \rangle \mid e \langle B \rangle$

$\langle B \rangle ::= f \mid g$

ASD preditiva recursiva

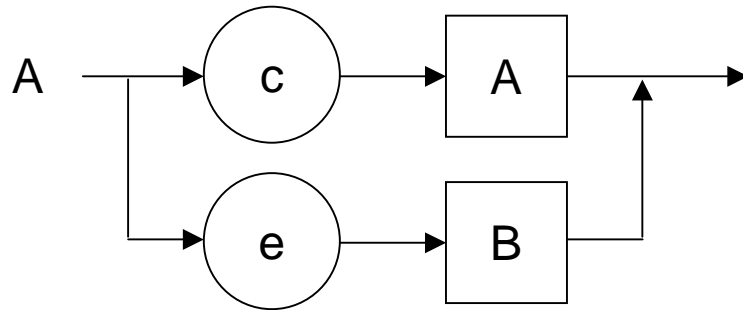
■ $\langle S \rangle ::= a\langle A \rangle d$



```
procedimento S
begin
  se (simbolo='a') então
    obter_simbolo;
    A;
  se (simbolo='d')
    então obter_simbolo
    senão ERRO ("d esperado");
  senão ERRO ("a esperado");
end
```

ASD preditiva recursiva

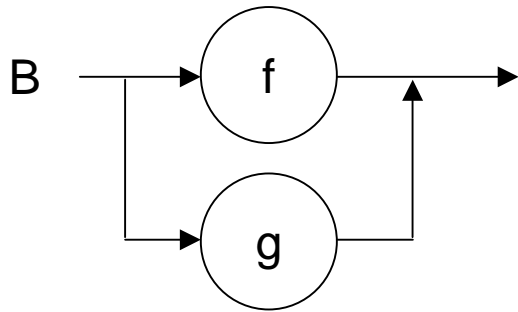
■ $\langle A \rangle ::= c\langle A \rangle \mid e\langle B \rangle$



```
procedimento A
begin
  se (simbolo='c') então
    obter_simbolo;
    A;
  senão se (simbolo='e') então
    obter_simbolo;
    B;
  senão ERRO("c ou e esperados");
end
```


ASD preditiva recursiva

■ $\langle B \rangle ::= f \mid g$



```
procedimento B
begin
  se (simbolo='f') ou (simbolo='g')
    então obter_simbolo
  senão ERRO("f ou g esperados");
end
```



ASD preditiva recursiva

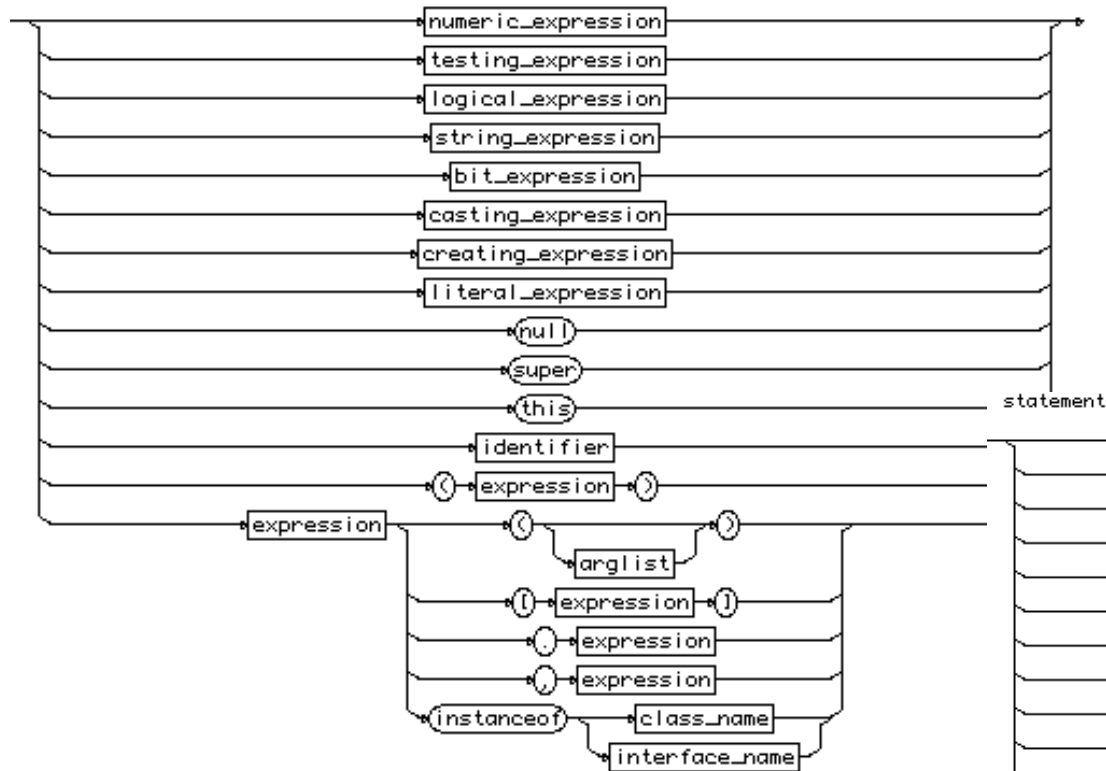
■ Programa principal

```
procedimento ASD
begin
  obter_simbolo;
  S;
  se (terminou_cadeia)
    então SUCESSO
    senão ERRO
end
```

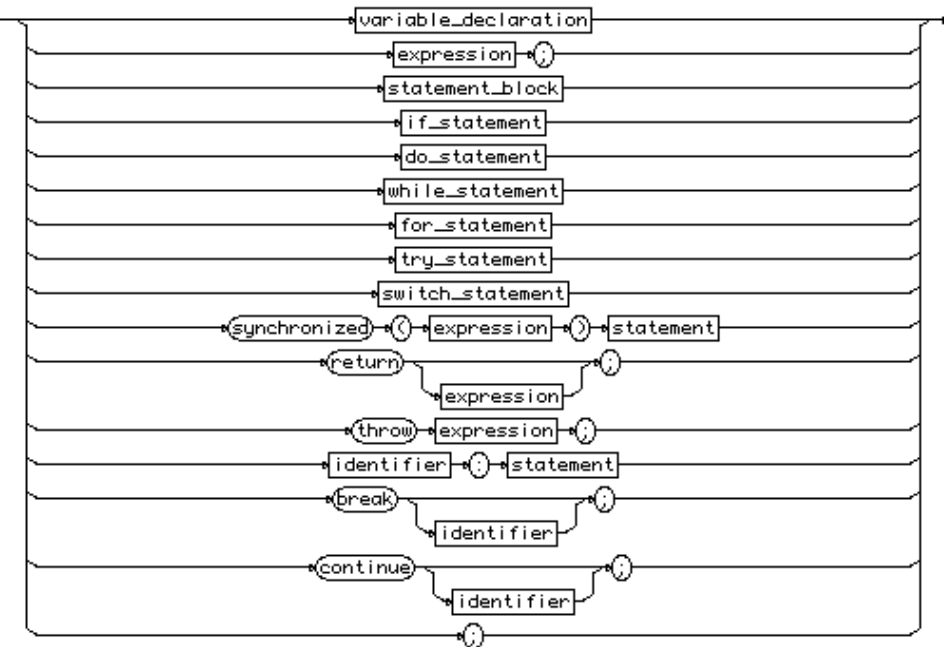
Grafos sintáticos para Java

<http://www.cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>

expression

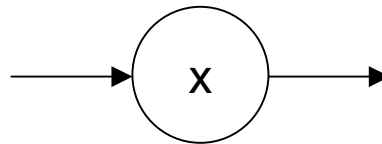


statement



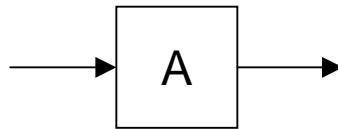
ASD preditiva recursiva

- Regras de transformação
 - Regras gramaticais → grafos sintáticos
- 1. Toda regra é mapeada em um grafo
- 2. Toda ocorrência de um terminal x em uma forma corresponde ao seu reconhecimento na cadeia de entrada e à leitura do próximo símbolo dessa cadeia

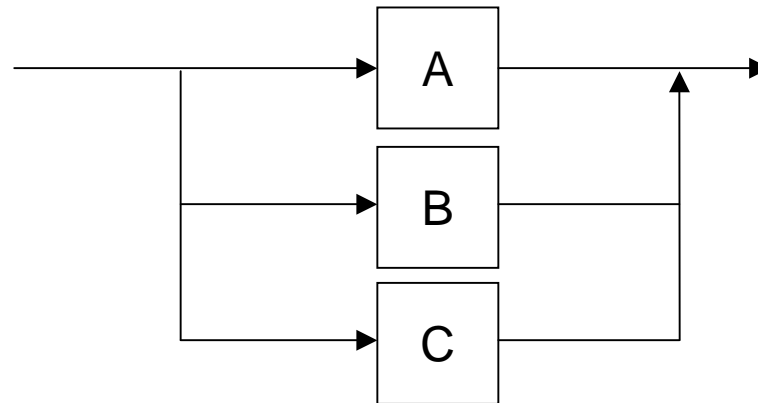


ASD preditiva recursiva

3. Toda ocorrência de um não-terminal A corresponde a análise imediata de A

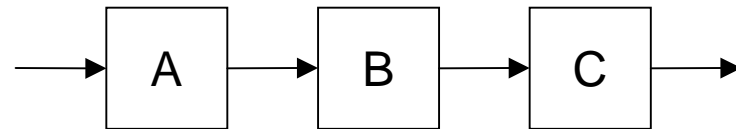


4. Alternativas são representadas como

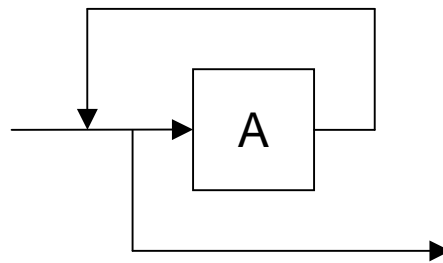


ASD preditiva recursiva

5. Uma seqüência A B C é mapeada em



6. A forma $\{A\}^*$ ou A^* é representada por





ASD preditiva recursiva

■ Exercício

$$\langle A \rangle ::= x \mid (\langle B \rangle)$$
$$\langle B \rangle ::= \langle A \rangle \langle C \rangle$$
$$\langle C \rangle ::= +\langle A \rangle \langle C \rangle \mid \lambda$$

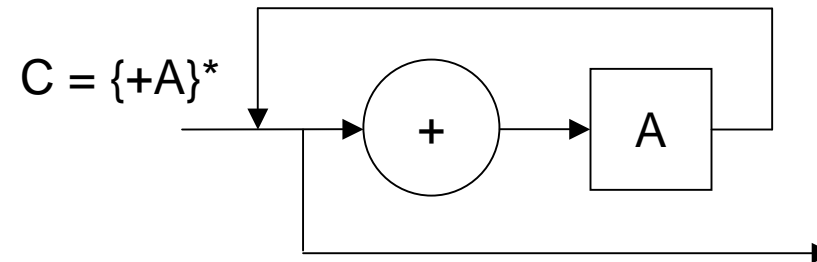
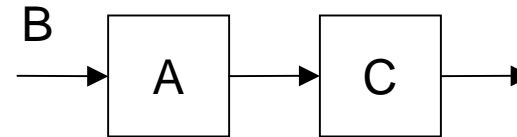
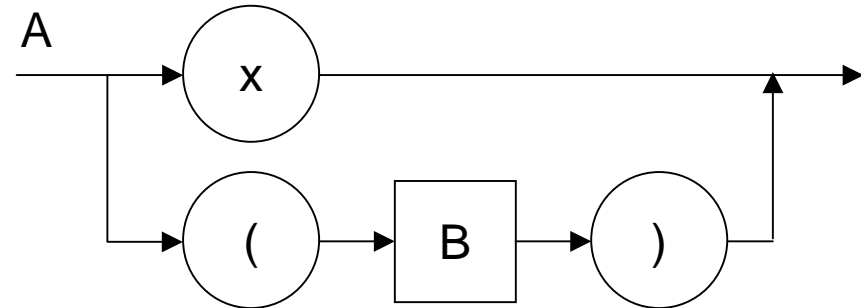
ASD preditiva recursiva

■ Exercício

$\langle A \rangle ::= x \mid (\langle B \rangle)$

$\langle B \rangle ::= \langle A \rangle \langle C \rangle$

$\langle C \rangle ::= +\langle A \rangle \langle C \rangle \mid \lambda$





ASD preditiva recursiva

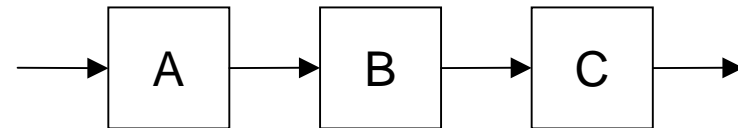
- Regras de tradução

- Grafos sintáticos → procedimentos

1. Reduzir o número de grafos: união de grafos para maior simplicidade e eficiência
 - Bom senso!
2. Escrever um procedimento para cada grafo

ASD preditiva recursiva

3. A seqüência



origina o procedimento

begin

A;

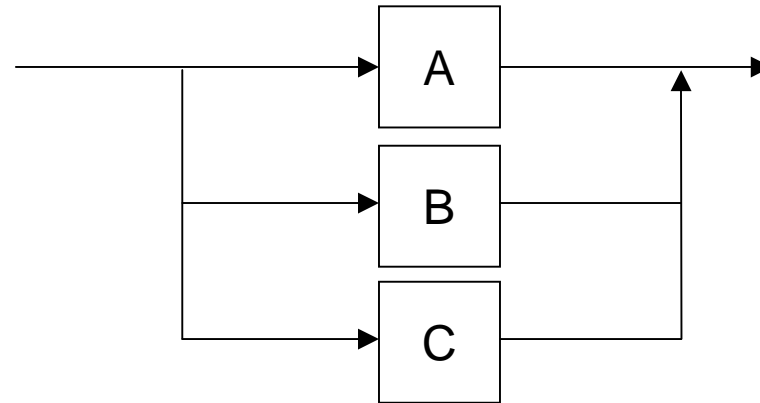
B;

C;

end

ASD preditiva recursiva

4. A alternativa



origina o procedimento

begin

se (símbolo está em $\text{first}(A)$) então A

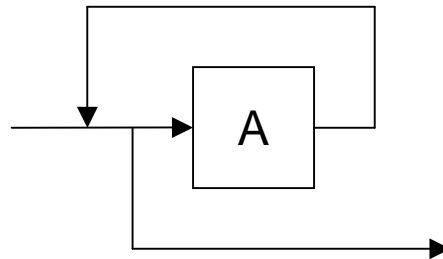
senão se (símbolo está em $\text{first}(B)$) então B

senão se (símbolo está em $\text{first}(C)$) então C

end

ASD preditiva recursiva

5. Uma repetição



origina o procedimento

begin

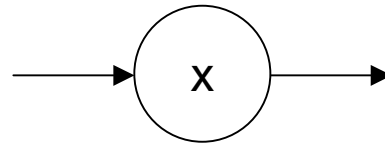
 enquanto (símbolo está em $\text{first}(A)$) faça

 A;

end

ASD preditiva recursiva

6. O terminal



origina

begin

se (símbolo=x)

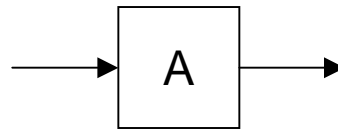
então obter_simbolo

senão ERRO("x esperado");

end

ASD preditiva recursiva

7. O não terminal



origina

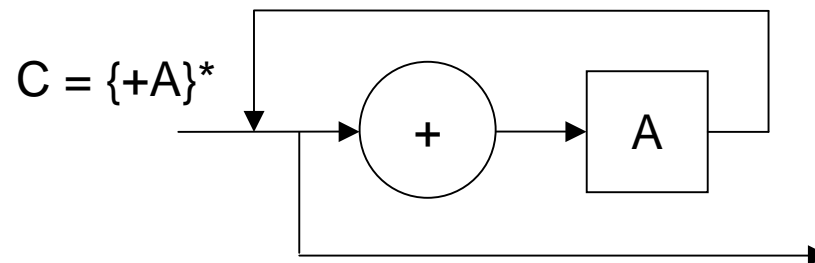
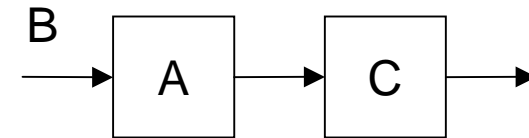
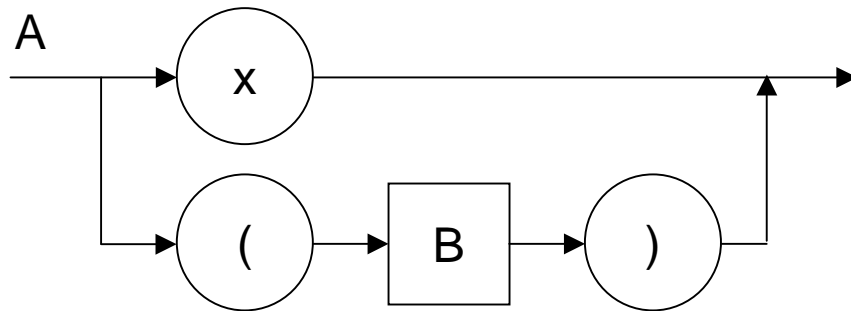
begin

A;

end

ASD preditiva recursiva

- Exercício: fazer o(s) procedimento(s) para os grafos sintáticos



ASD preditiva recursiva

procedimento A

begin

se (simbolo='x') então obter_simbolo

senão se (simbolo='(') então

repita

obter_simbolo;

A;

até que (simbolo<>'+');

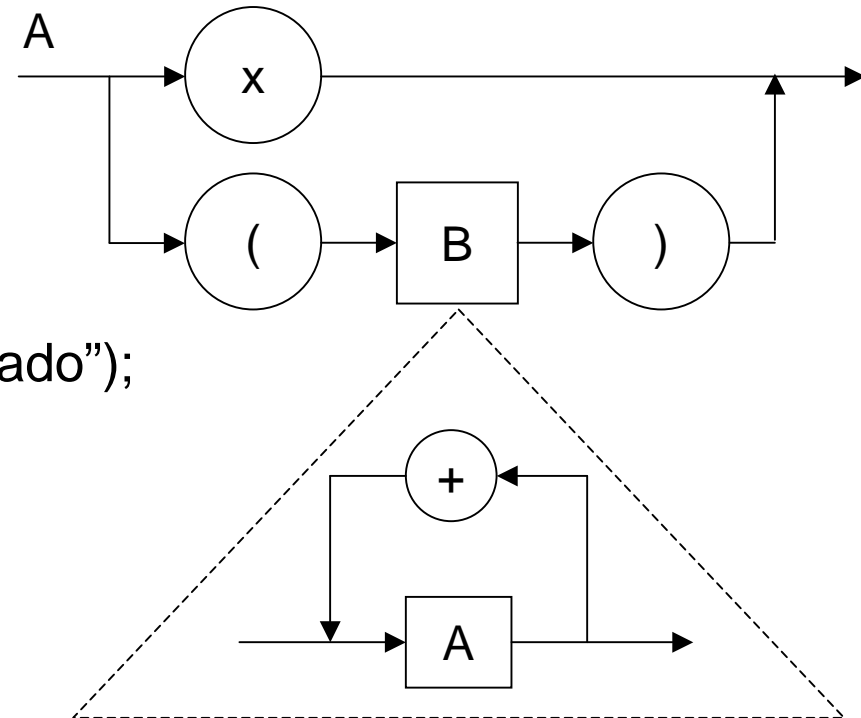
se (simbolo=')')

então obter_simbolo

senão ERRO("(" esperado");

senão ERRO("x ou (esperados") ;

end





ASD preditiva

■ Exercício

- A gramática seguinte é LL(1)? Se não é, transforme-a

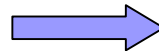
$\langle S \rangle ::= i \langle A \rangle$
 $\langle A \rangle ::= := \langle E \rangle$
 $\langle E \rangle ::= \langle T \rangle + \langle E \rangle \mid \langle T \rangle$
 $\langle T \rangle ::= \langle F \rangle * \langle T \rangle \mid \langle F \rangle$
 $\langle F \rangle ::= \langle P \rangle - \langle F \rangle \mid \langle P \rangle$
 $\langle P \rangle ::= i \mid (\langle E \rangle)$

ASD preditiva

■ Exercício

- A gramática seguinte é LL(1)? Se não é, transforme-a. Por re-escrita:

$\langle S \rangle ::= i \langle A \rangle$
 $\langle A \rangle ::= := \langle E \rangle$
 $\langle E \rangle ::= \langle T \rangle + \langle E \rangle \mid \langle T \rangle$
 $\langle T \rangle ::= \langle F \rangle * \langle T \rangle \mid \langle F \rangle$
 $\langle F \rangle ::= \langle P \rangle - \langle F \rangle \mid \langle P \rangle$
 $\langle P \rangle ::= i \mid (\langle E \rangle)$



$\langle S \rangle ::= i \langle A \rangle$
 $\langle A \rangle ::= := \langle E \rangle$
 $\langle E \rangle ::= \langle T \rangle \langle E' \rangle$
 $\langle E' \rangle ::= + \langle E \rangle \mid \lambda$
 $\langle T \rangle ::= \langle F \rangle \langle T' \rangle$
 $\langle T' \rangle ::= * \langle T \rangle \mid \lambda$
 $\langle F \rangle ::= \langle P \rangle \langle X \rangle$
 $\langle X \rangle ::= - \langle F \rangle \mid \lambda$
 $\langle P \rangle ::= i \mid (\langle E \rangle)$

- EBNF?

- $\langle E' \rangle ::= + \langle E \rangle \mid \lambda \rightarrow \langle E' \rangle ::= [+E]$