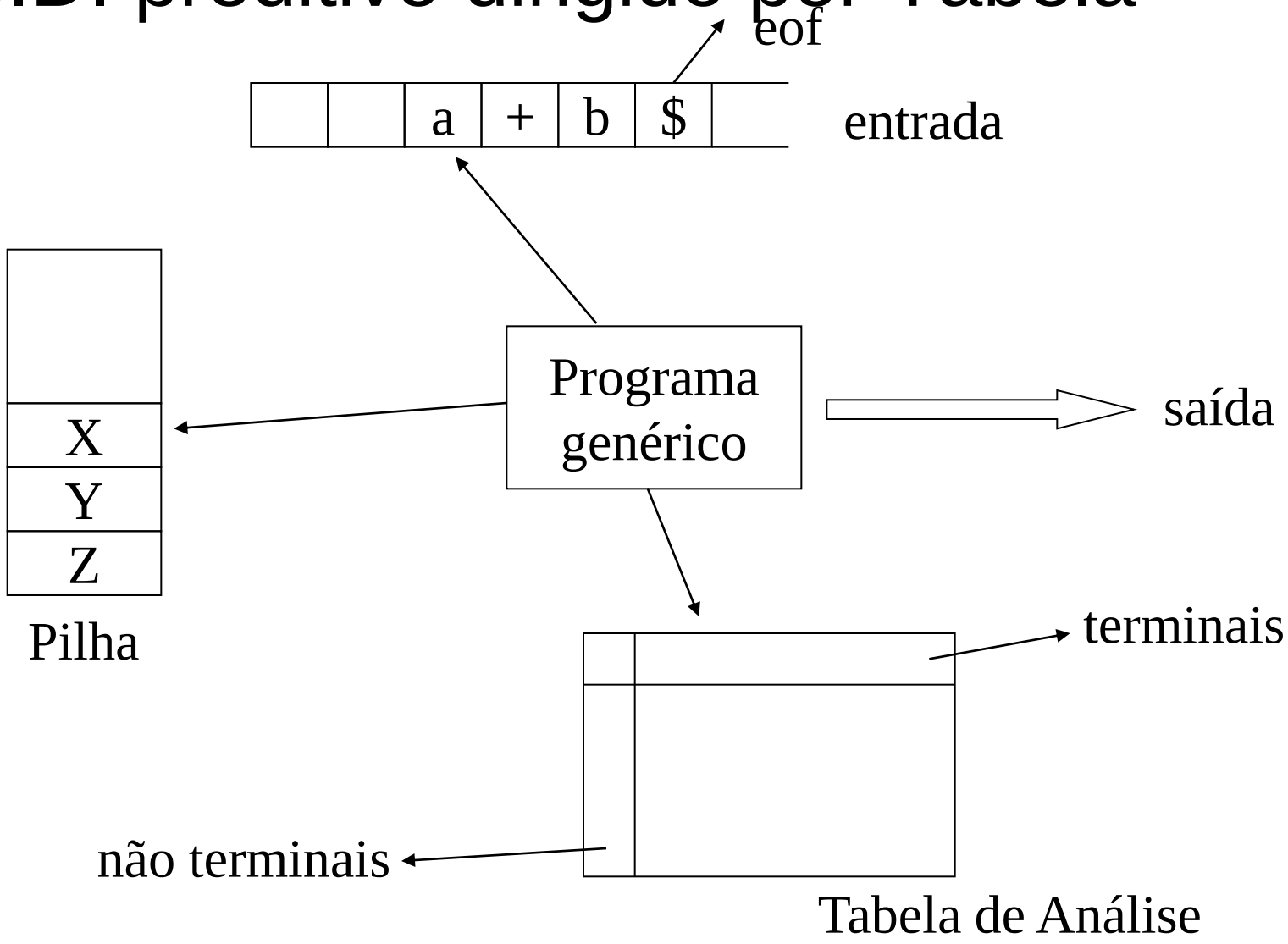
Análise Sintática II: Analisadores Descendentes Preditivos

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]←S; i←1; termino←false;
simbolo←analex(S);
repita
        X←P[i];
        se X é terminal então
                se X=simbolo então
                        [simbolo←analex(S);
                        i←i-1;
                        se i=0 então termino=true]
                senão ERRO
        senão se tabela[X,simbolo] ≠ ' ' então
                        [P[i]←Xn; P[i+1]←Xn-1; ... P[i+n-1]←X1;
                        i←i+n-1]
                senão ERRO
até termino
se simbolo ≠ $ 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 -> α faça:

1. Para cada terminal a Є First (α) coloque A -> α na entrada M[A, a]
2. Se λ Є First (α) coloque A -> α para M[A,b] para cada b Є Follow (A)
3. Se λ Є First (α) e $ Є Follow (A) coloque A -> α para M[A,$]

Calculo das relações First e Follow

FIRST

1. Se x é terminal então First(X) = {X}
2. Se X -> λ é produção então coloque λ no First(x)
3. Se X é não terminal e X -> Y1Y2 ...Yn é produção adicione First(Yi) para First(X) se os precedentes Yjs contém λ em seus First

FOLLOW

1. Follow(S) contém $
2. Para A -> αBβ tudo em First(β) exceto λ vai para Follow(B)
3. Para A -> αB ou A -> αBβ onde First(β) contém λ, Follow(B) contém tudo que está no Follow (A)

Exemplos de construção de Tabelas de Análise

- Gramáticas simples: sem λ
- Gramáticas ambíguas e com λ

1. Construção da Tabela de Análise (com regra 1) para uma gramática sem λ

S → ^{1}AS | ^{2} BA

A → ^{3} aB | ^{4} C

B → ^{5} bA | ^{6} d

C → ^{7} c

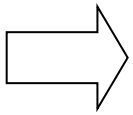
Seja a gramática

S → AS | BA

A → aB | C

B → bA | d

C → c



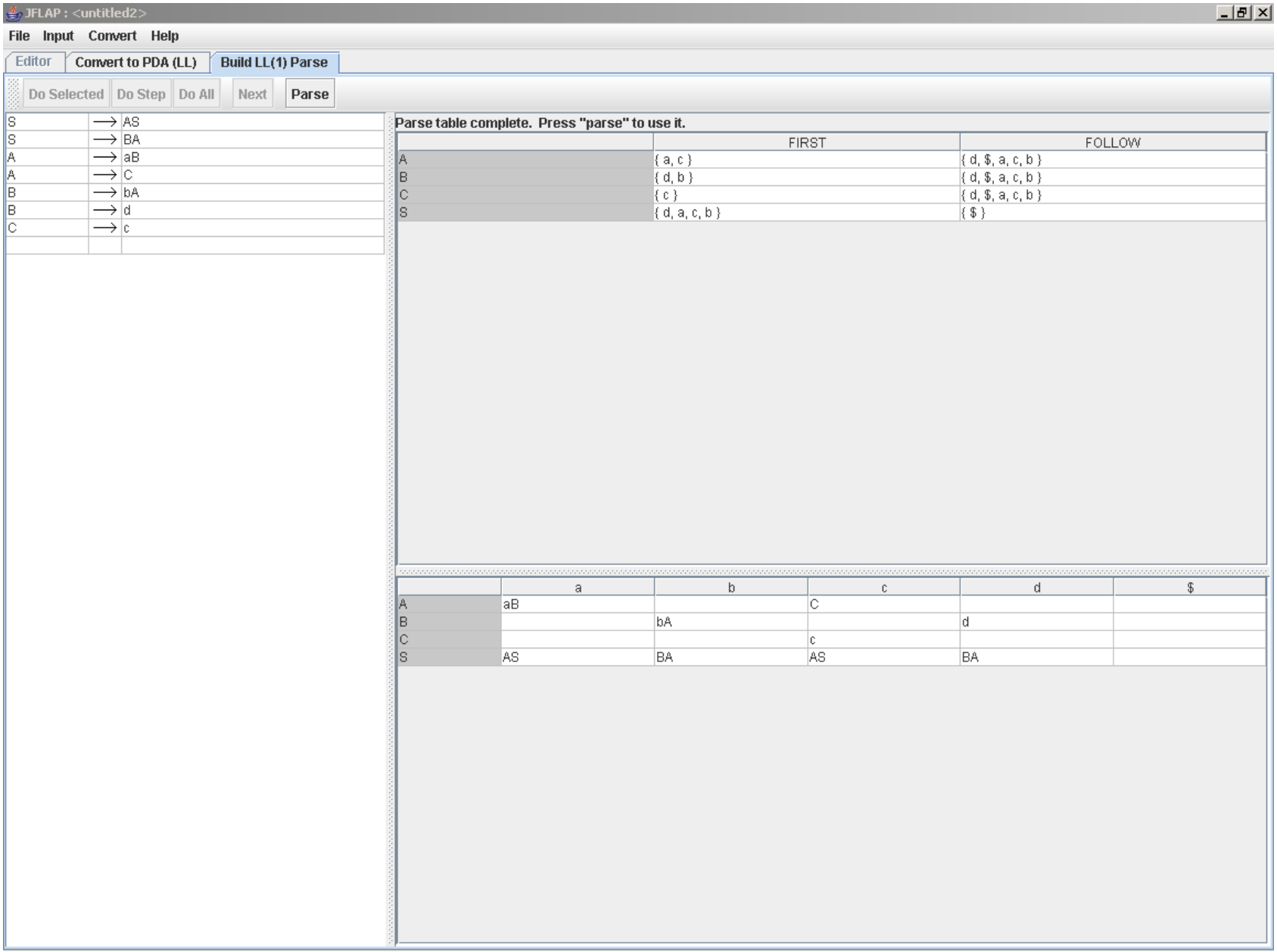
First	
S	a, c, b, d
A	a, c
B	b, d
C	c

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



Análise de abcdad

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

2. Exemplo de gramática para a qual é preciso calcular Follows

Gramática ambígua:

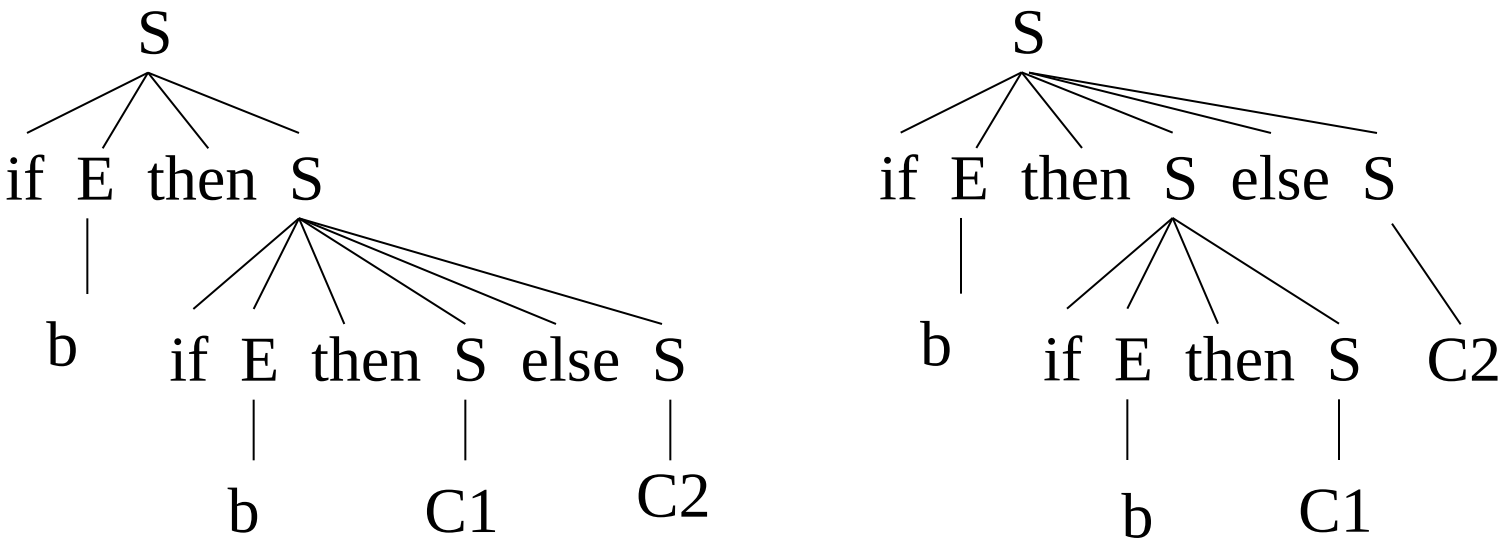
S → if E then S | if E then S else S | a

E → b

Cadeia

if b then if b then C1 else C2

Duas derivações possíveis:



Reescrevendo para tentar tirar ambigüidade

S → if E then S S’ | a

S’ → else S | λ

E → b

S → [1] if E then S S' | a [2]
S' → [3] else S | λ [4]
E → [5] b

Entretanto, mesmo com a reescrita a gramática tem entradas multidefinidas

First(S)={if, a}
First(S')={else, λ}
First(E)={b}

Follow(S)={else, $}
Follow(S')={else, $}
Follow(E)={then}

Tabela de Análise

NT / T	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

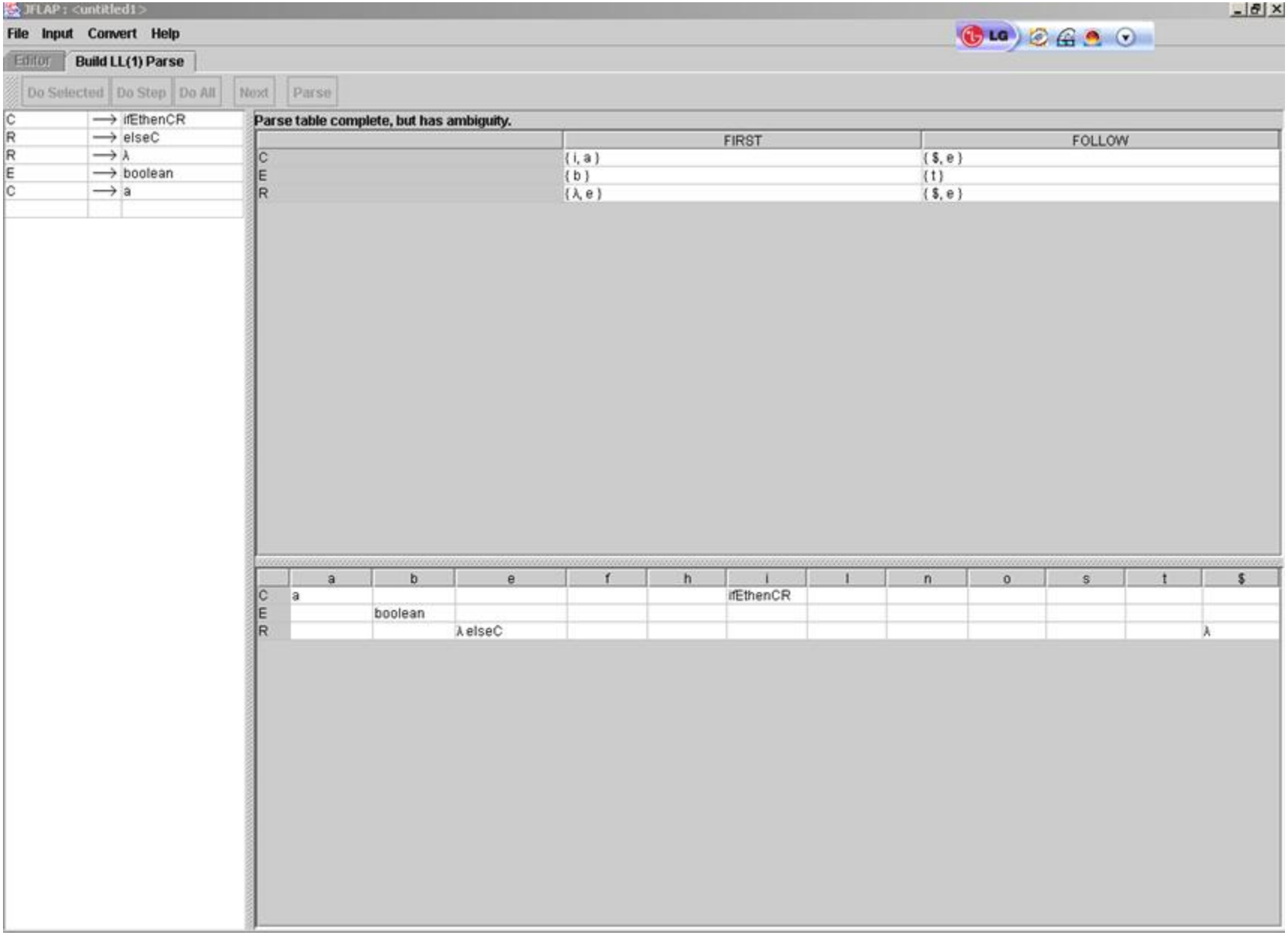
e adota-se a regra informal de associar o else com o if mais próximo

Tabela de Análise

NT / T	if	then	else	a	b	$
S	1			2		
S’			3			4
E					5	

excluiu-se a regra 4

Geração da Tabela de Análise JFlap



Análise da sentença

- If b then if b then a else a
- Mostrem a Pilha, Entrada, Regra usada

Exercício

- Construção da Tabela de Análise:

  E -> TE´

  E´-> v TE´ | λ

  T -> FT´

  T´-> & FT´ | λ

  F -> not F | id

- Análise da sentença: **id V id & id**

Tabela de Análise

- First (F) = First(T) = First(E) = {not, id}
- First(T´) = {&, λ}
- First(E´) = {v, λ}

- Follow(E) = Follow(E´) = {$}
- Follow(T) = First(E´) + Follow(E´) {regras 2 e 3} = {v, $}
- Follow(T´) = Follow(T) = {v, $}
- Follow(F) = First(T´) + Follow(T´) = {v,&,$}

Tabela de Análise

	id	v	&	not	$
E	E -> TE´			E -> TE´	
E´		E´-> v TE´			E´-> λ
T	T -> FT´			T -> FT´	
T´		T´-> λ	T´-> & FT´		T´-> λ
F	F-> id			F-> not F	

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 ϵ N com as regras:

$$A \rightarrow X_1\alpha_1 \mid X_2\alpha_2 \mid \ldots \mid X_n\alpha_n$$

onde X ϵ V, temos que:

- First(X_i) são disjuntos dois a dois:
  - First(X_k) ∩ First(X_j) = { } para ∀ k,j ϵ {1, 2...n} com k≠j

2. Para toda produção A -> α | β
   - Se β =>* λ, 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 α =>* λ, 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 }	{ $ }

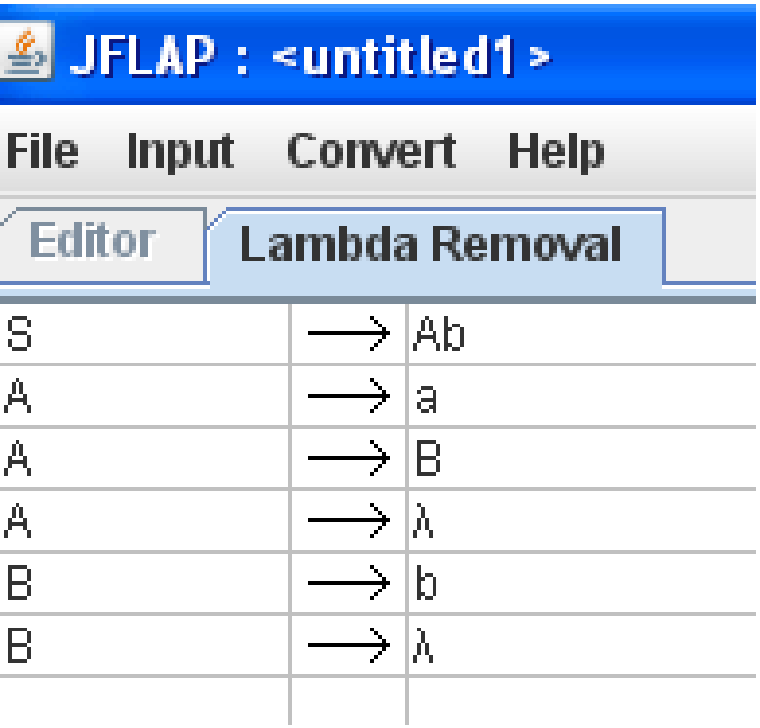
d) LL(1), pois passa na regra 1. A regra 2 não se aplica

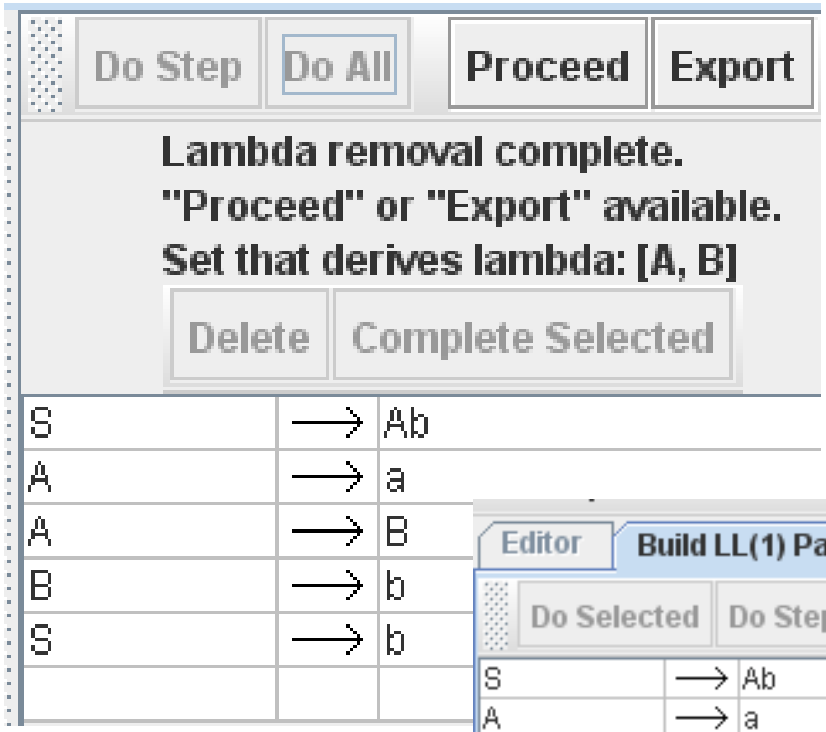
S	→	aSe
S	→	B
B	→	bBe
B	→	C
C	→	cBe
C	→	d

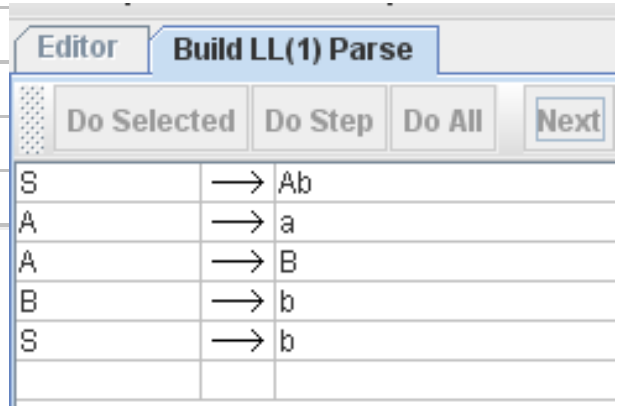
Parse table complete. Press "parse" to use it.

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

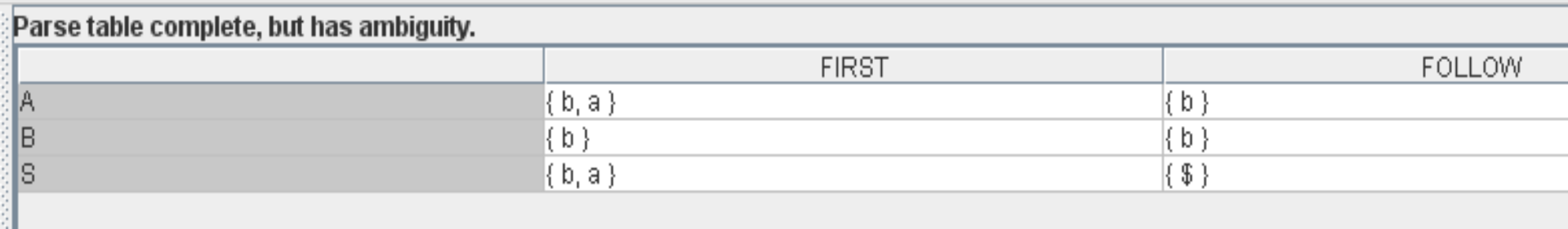
Reescrita de b): remover λ com Jflap



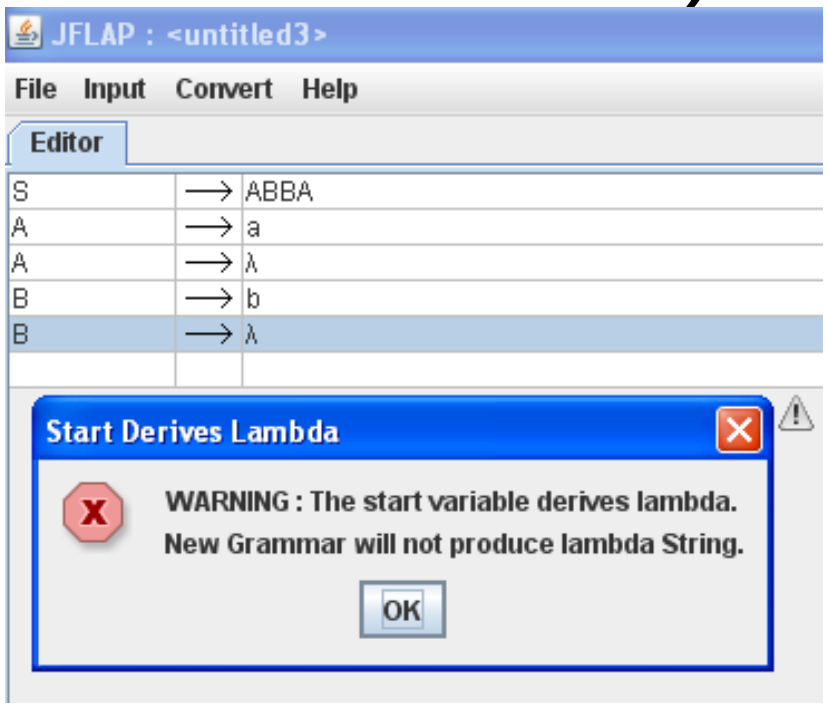




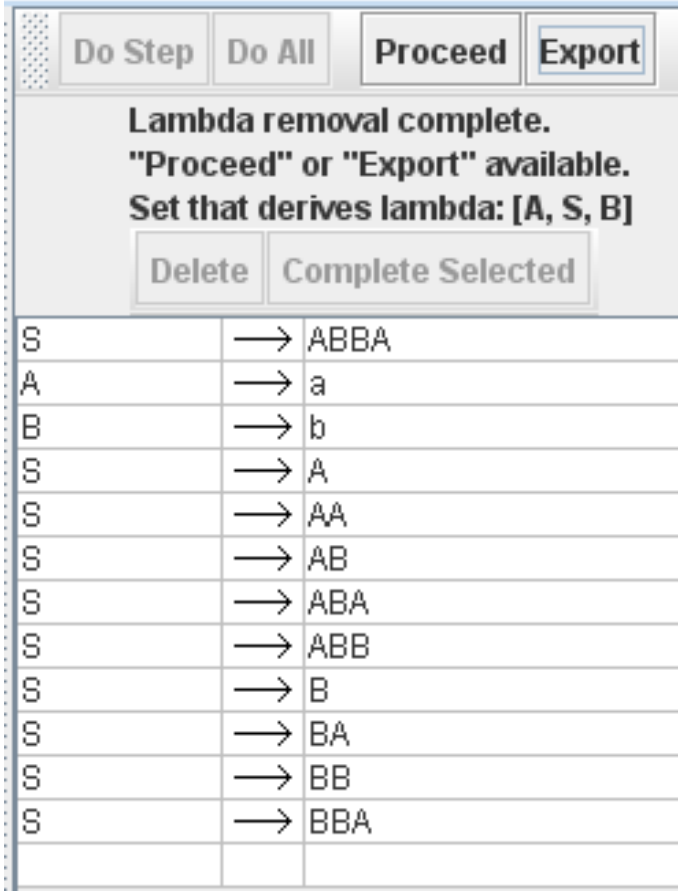
b) Não é LL(1), nem depois da reescrita, mas é LL(2)



Reescrita de c): remover λ com Jflap



Temos que colocar λ 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

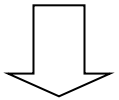
- Outra maneira de implementar é usar procedimentos recursivos
  - cria-se um procedimento para cada não terminal da gramática
  - cada procedimento deve ser responsável por reconhecer a parte da sentença de entrada derivada dele

Exemplo de gramática que foi reescrita na notação EBNF

E → E + T | E - T | + T | - T | T

T → T * F | T / F | F

F → a | b | (E)



E → (+ T | - T | T) {+ T | - T} → [+ | -] T { (+ | -) T}

T → F { * F | / F} → F { (* | /) F }

F → a | b | (E)

Veremos a criação do parser via exemplos e depois mais formalmente.

```
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;
```

{α} = while
[α] = if

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

{α} = while
[α] = if

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

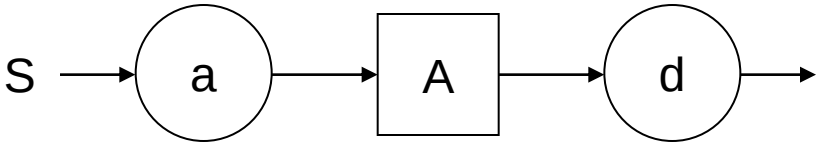
<S> ::= a<A>d

<A> ::= c<A> | e<B>

<B> ::= f | g

ASD preditiva recursiva

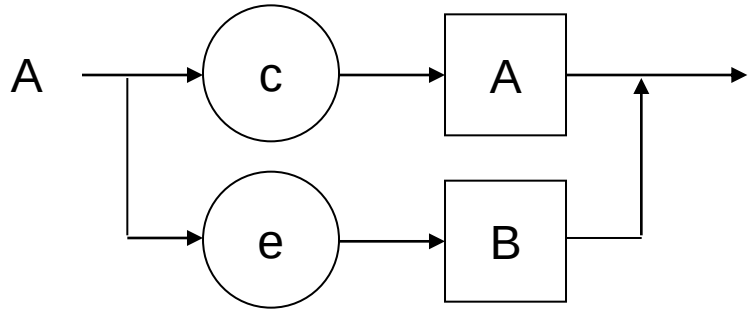
- <S> ::= a<A>d




```
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

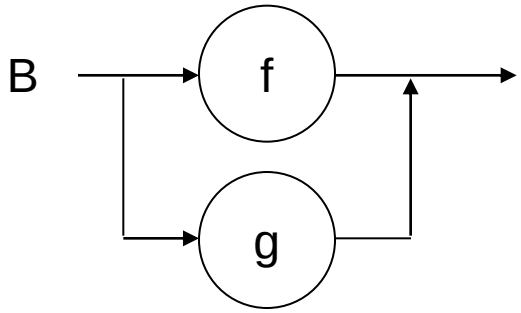
- <A> ::= c<A> | e<B>




```
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

- <B> ::= f | g




```
procedimento B
begin
   se (simbolo=‘f’) ou (simbolo=‘g’)
      então obter_simbolo
      senão ERRO(“f ou g esperados”);
end
```

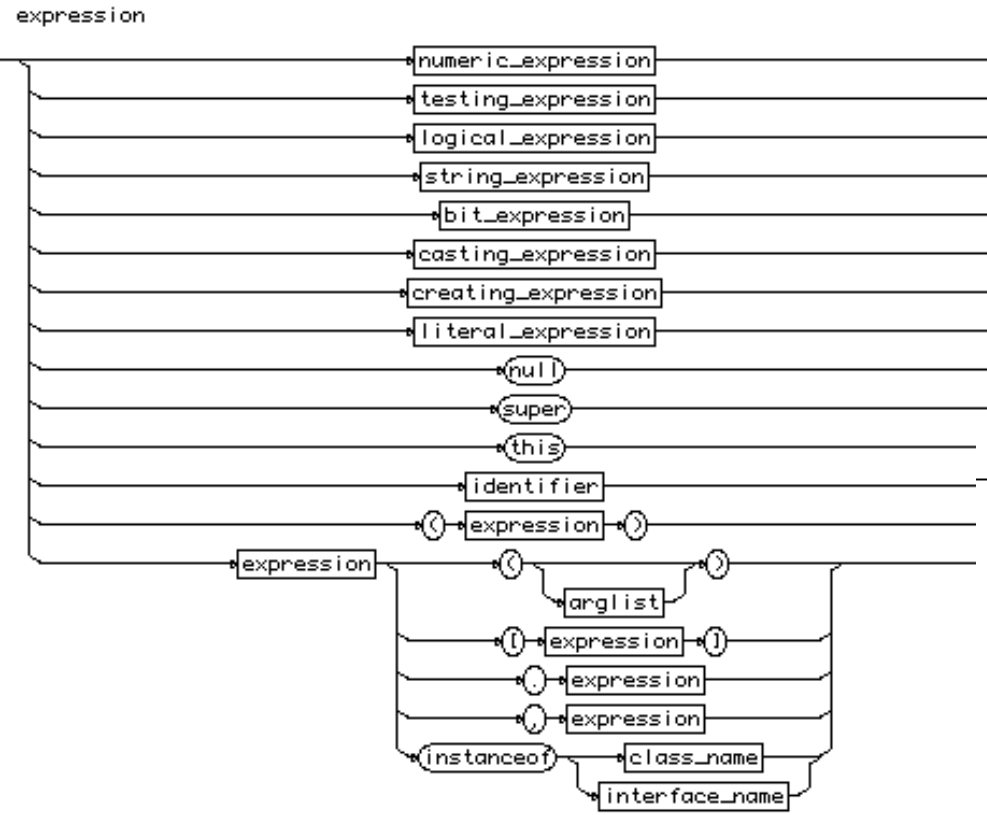
ASD preditiva recursiva

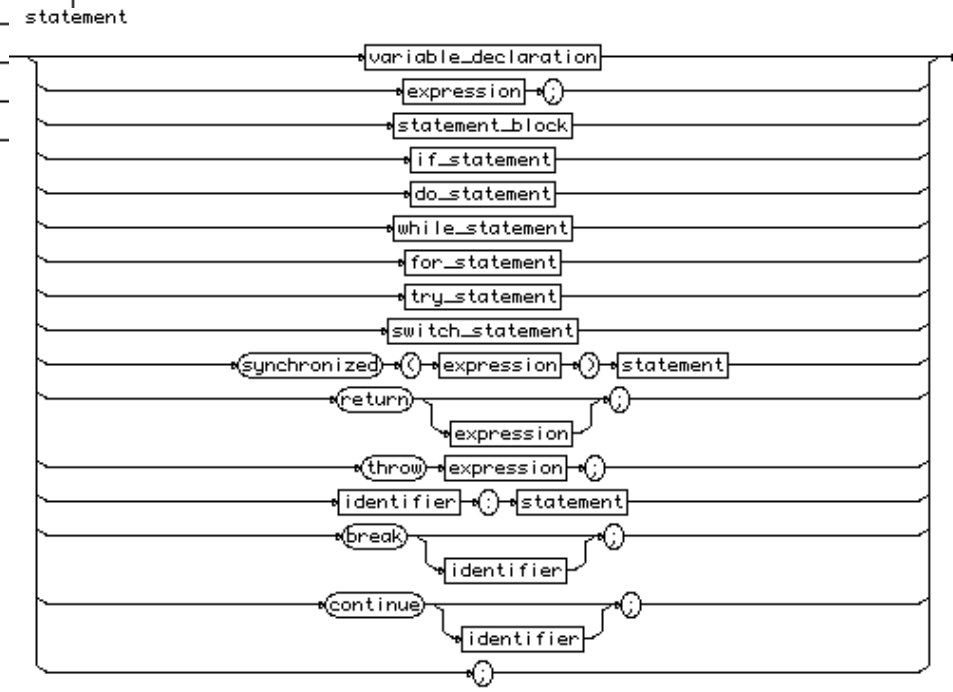
- Programa principal

```
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



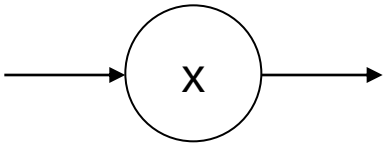


ASD preditiva recursiva

- Regras de transformação
  - Regras gramaticais → grafos sintáticos

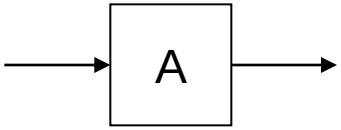
1. Toda regra é mapeada em um grafo

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

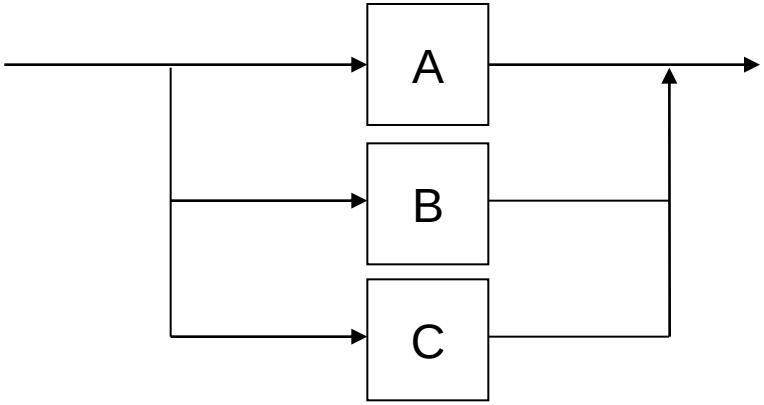


ASD preditiva recursiva

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

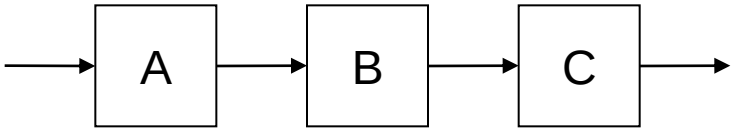


4. Alternativas são representadas como

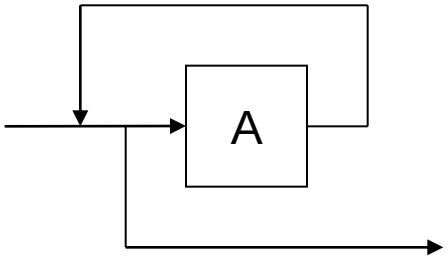


ASD preditiva recursiva

5. Uma seqüência A B C é mapeada em



6. A forma {A}* ou A* é representada por



ASD preditiva recursiva

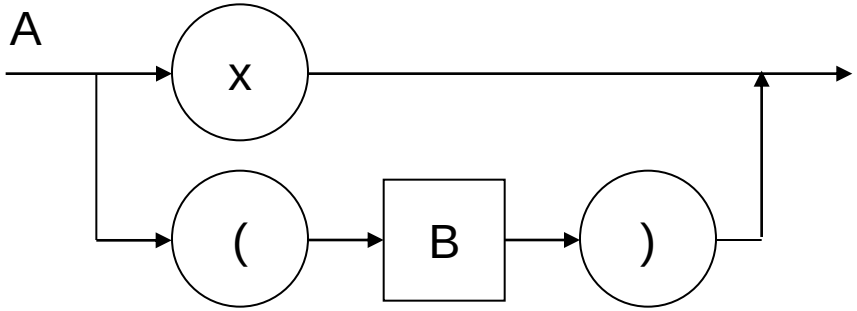
- Exercício

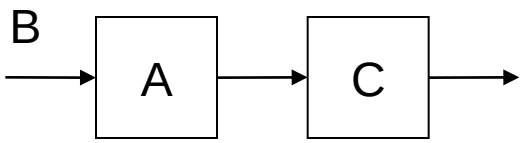
<A> ::= x | (<B>)
<B> ::= <A><C>
<C> ::= +<A><C> | λ

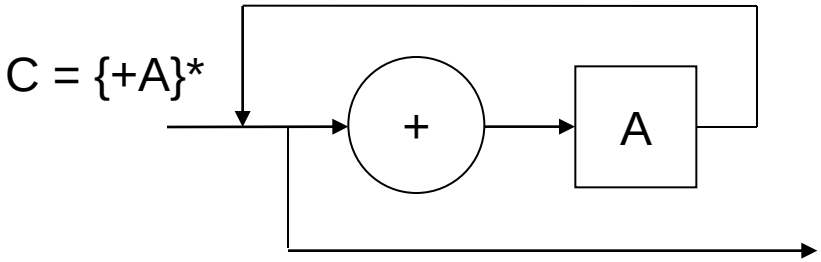
ASD preditiva recursiva

- Exercício

<A> ::= x | (<B>)
<B> ::= <A><C>
<C> ::= +<A><C> | λ







ASD preditiva recursiva

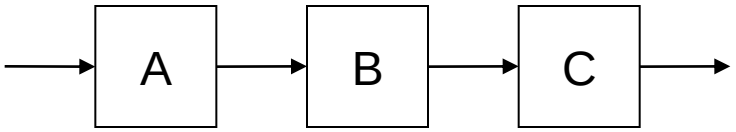
- Regras de tradução
  - Grafos sintáticos → procedimentos

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

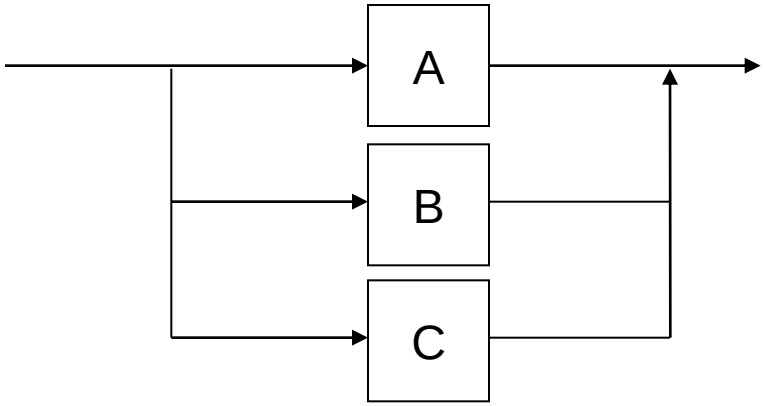


origina o procedimento

```
begin
        A;
        B;
        C;
end
```

ASD preditiva recursiva

4. A alternativa

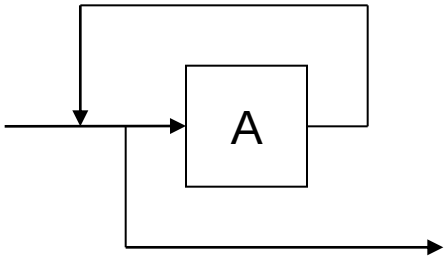


origina o procedimento

```
begin
        se (símbolo está em first(A)) então A
        senão se (símbolo está em first(B)) então B
        senão se (símbolo está em first(C)) então C
end
```

ASD preditiva recursiva

5. Uma repetição

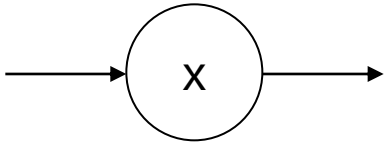


origina o procedimento

```
begin
        enquanto (símbolo está em first(A)) faça
                A;
end
```

ASD preditiva recursiva

6. O terminal

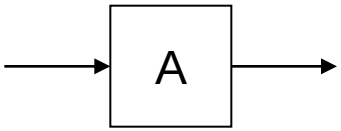


origina

```
begin
        se (símbolo=x)
                então obter_simbolo
                senão ERRO(“x esperado”);
end
```

ASD preditiva recursiva

7. O não terminal

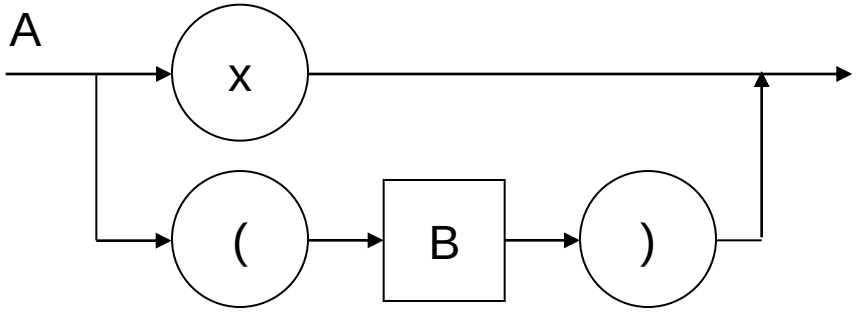


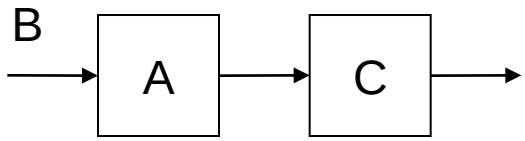
origina

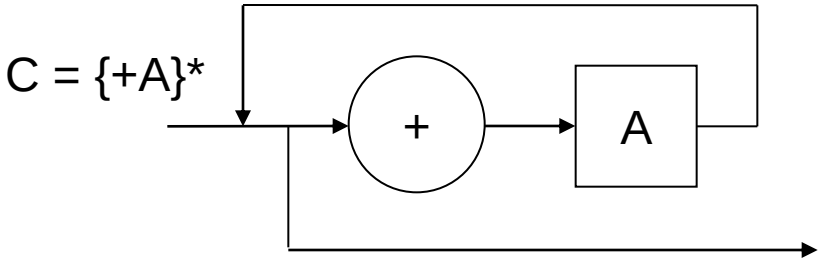
```
begin
        A;
end
```

ASD preditiva recursiva

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

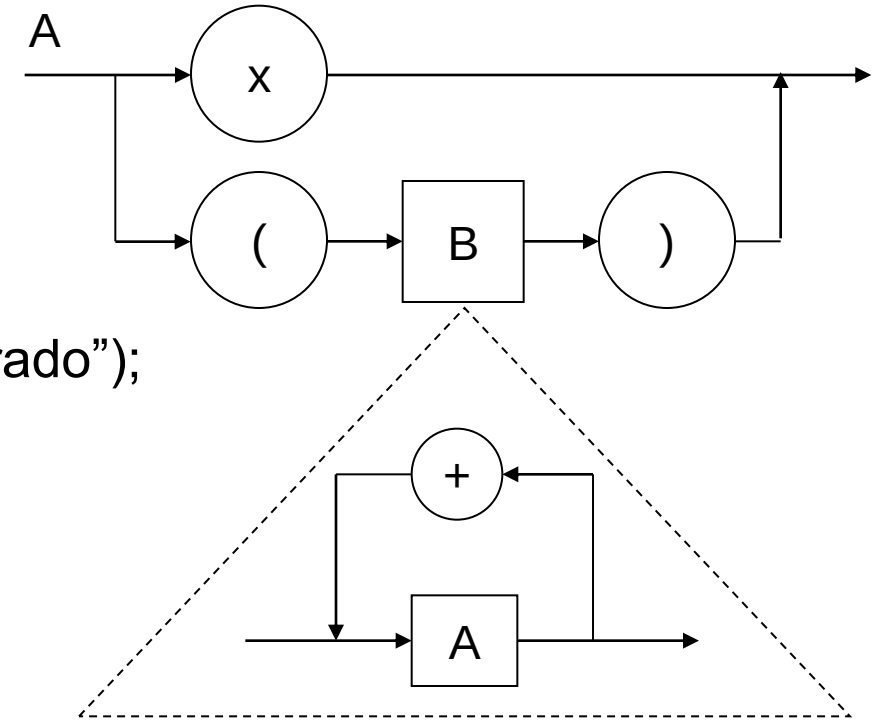






ASD preditiva recursiva

```
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

<S> ::= i<A>
<A> ::= :=<E>
<E> ::= <T> + <E> | <T>
<T> ::= <F> * <T> | <F>
<F> ::= <P> - <F> | <P>
<P> ::= i | (<E>)

ASD preditiva

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

<S> ::= i<A>
<A> ::= :=<E>
<E> ::= <T> + <E> | <T>
<T> ::= <F> * <T> | <F>
<F> ::= <P> - <F> | <P>
<P> ::= i | (<E>)

<S> ::= i<A>
<A> ::= :=<E>
<E> ::= <T><E'>
<E'> ::= +<E> | λ
<T> ::= <F><T'>
<T'> ::= *<T> | λ
<F> ::= <P><X>
<X> ::= -<F> | λ
<P> ::= i | (<E>)

  - EBNF?
  - <E'> ::= +<E> | λ ➔ <E´> ::= [+E]