Análise Sintática II:
Analisadores Descendentes Preditivos

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 }	{ $ }

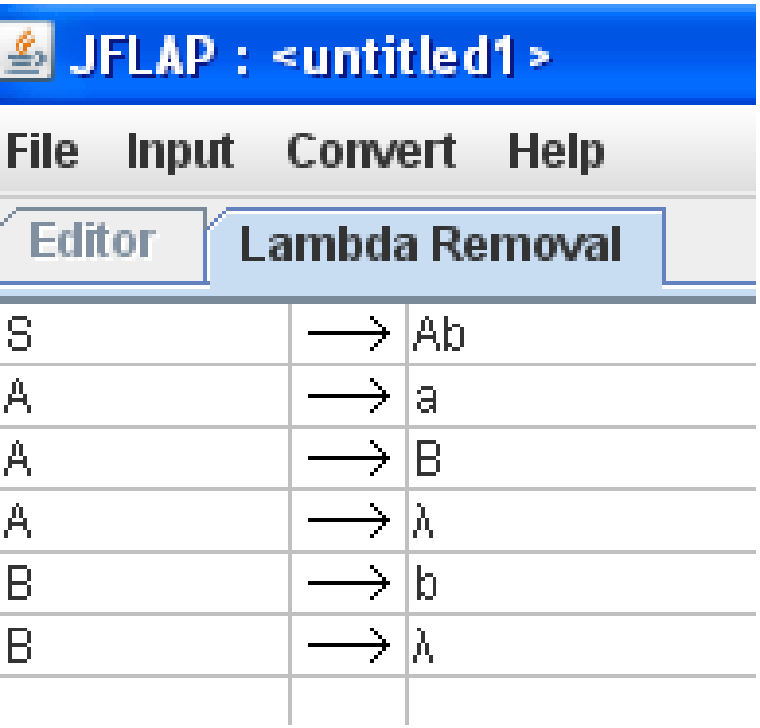
d) LL(1), pois passa na regra 1. A regra 2 não se aplica

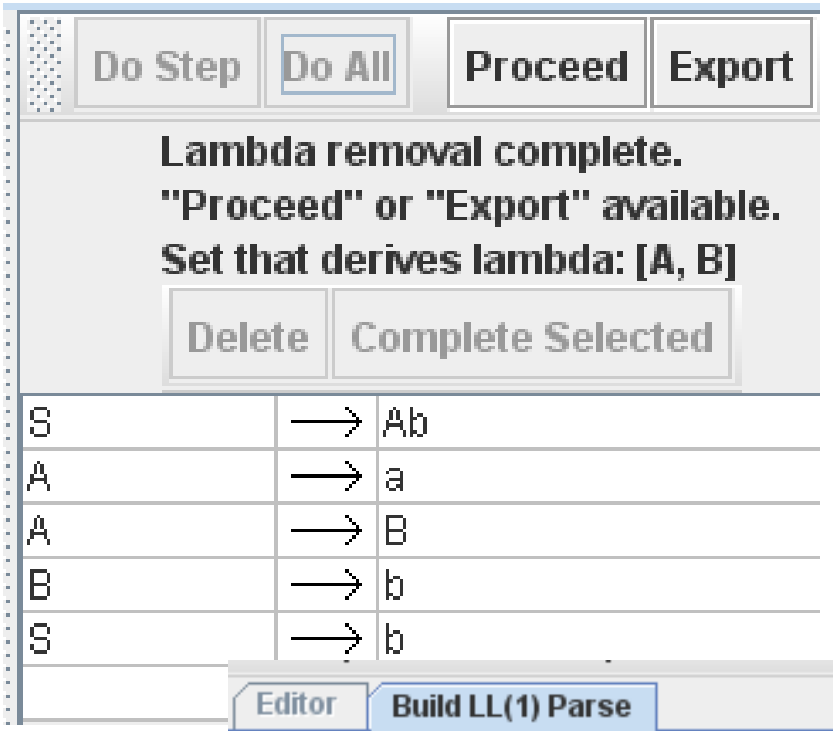
S	→	aSe
S	→	B
B	→	bBe
B	→	C
C	→	cBe
C	→	d

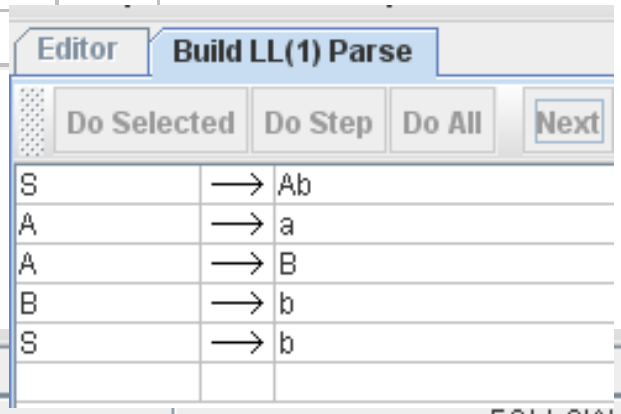
Parse table complete. Press "parse" to use it.

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

Reescrita de b): remover λ com Jflap

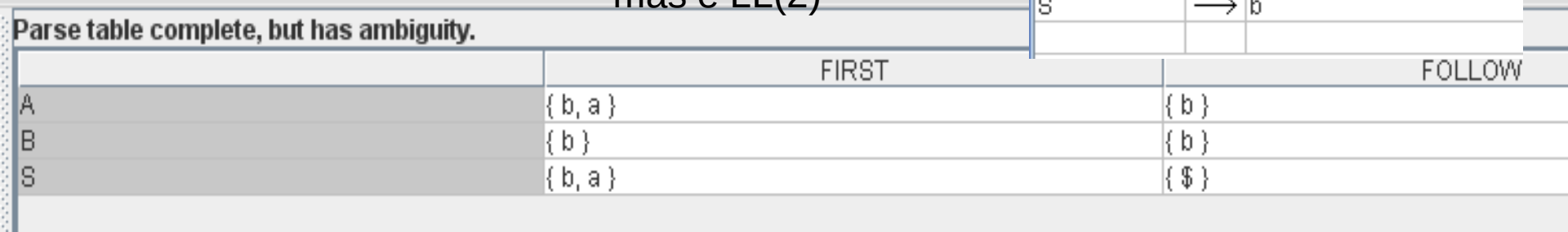




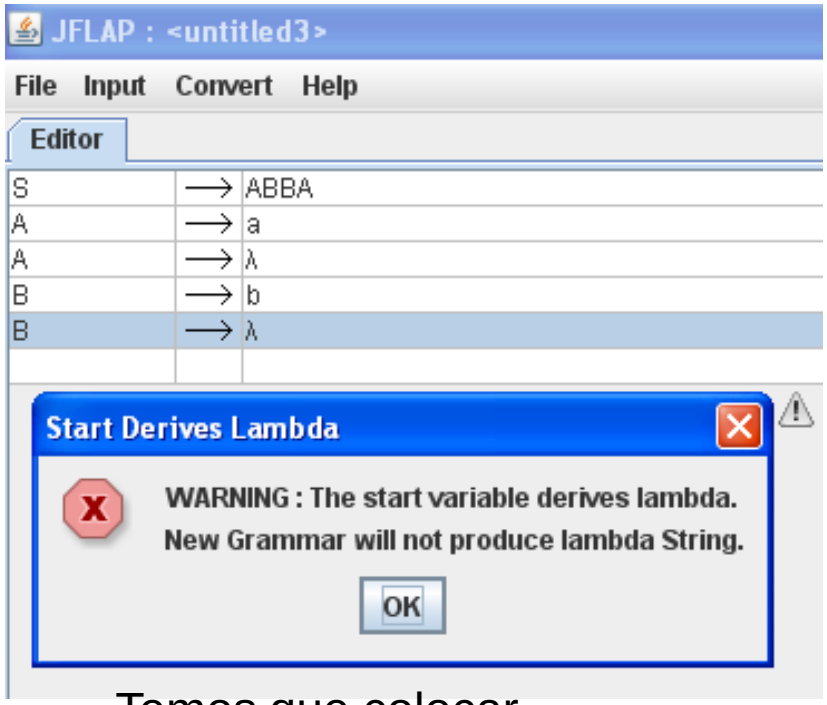


S ::= ab | b | bb não é LL(1)
EBNF: S ::= ab | b[b] LL(1)

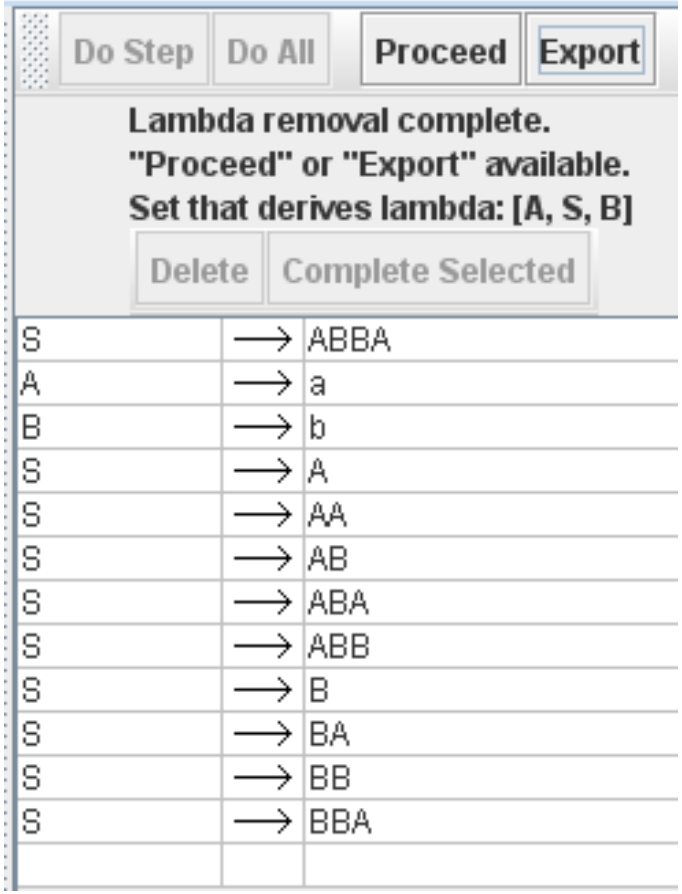
b) Não é LL(1), nem depois da reescrita, mas é LL(2)



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 }	{ $ }

Exercícios – Lista 3

20. Considere a gramática a seguir:

G=({S, E, C}, {do, while, comando, true}, P, S), em que P é

<S> ::= do <E> while <C>
<E> ::= comando | λ
<C> ::= true

Faça:

(a) Considere a análise sintática descendente não recursiva. Construa a tabela sintática para a gramática anterior.
(b) Utilizando a tabela anterior, reconheça a cadeia **do while true**

S	→	dEwC	1
E	→	c	2
E	→	λ	3
C	→	t	4

	FIRST	FOLLOW
C	{ t }	{ $ }
E	{ λ, c }	{ w }
S	{ d }	{ $ }

	c	d	t	w	$
C			t 4		
E	c 2			λ 3	
S		dEwC 1			

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

Escrever em EBNF

E → E + T | E - T | + T | - T | T

T → T * F | T / F | F

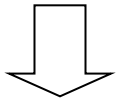
F → a | b | (E)

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)

```
program
begin
        simbolo←analex(S);
        E;
        se (terminou_cadeia => simbolo = $)
                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 → (+ 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;
```

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

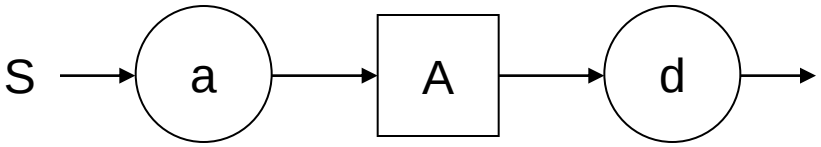
<A> ::= c<A> | e<B>

<B> ::= f | g

ASD preditiva recursiva

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

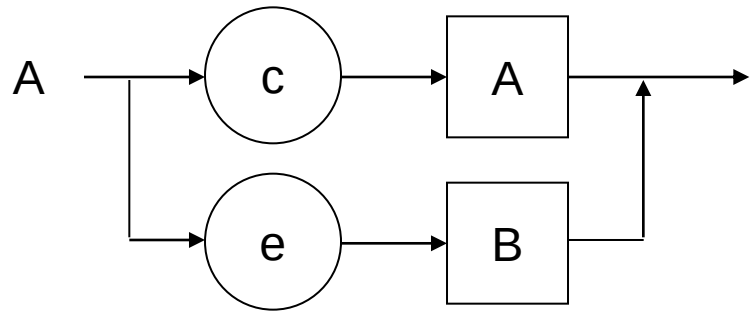
<S> ::= a<A>d
<A> ::= c<A> | e<B>
<B> ::= f | g




```
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>



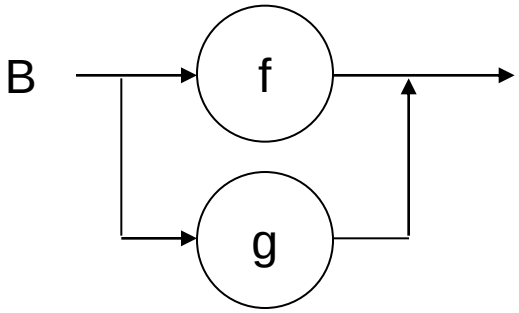

```
procedimento A
begin
   se (simbolo='c') então
      obter_simbolo;
      A;
   senão se (simbolo='e') então
         obter_simbolo
         B;
   senão ERRO("c ou e esperados");
end
```

<S> ::= a<A>d
<A> ::= c<A> | e<B>
<B> ::= f | g

ASD preditiva recursiva

- <B> ::= f | g

<S> ::= a<A>d
<A> ::= c<A> | e<B>
<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

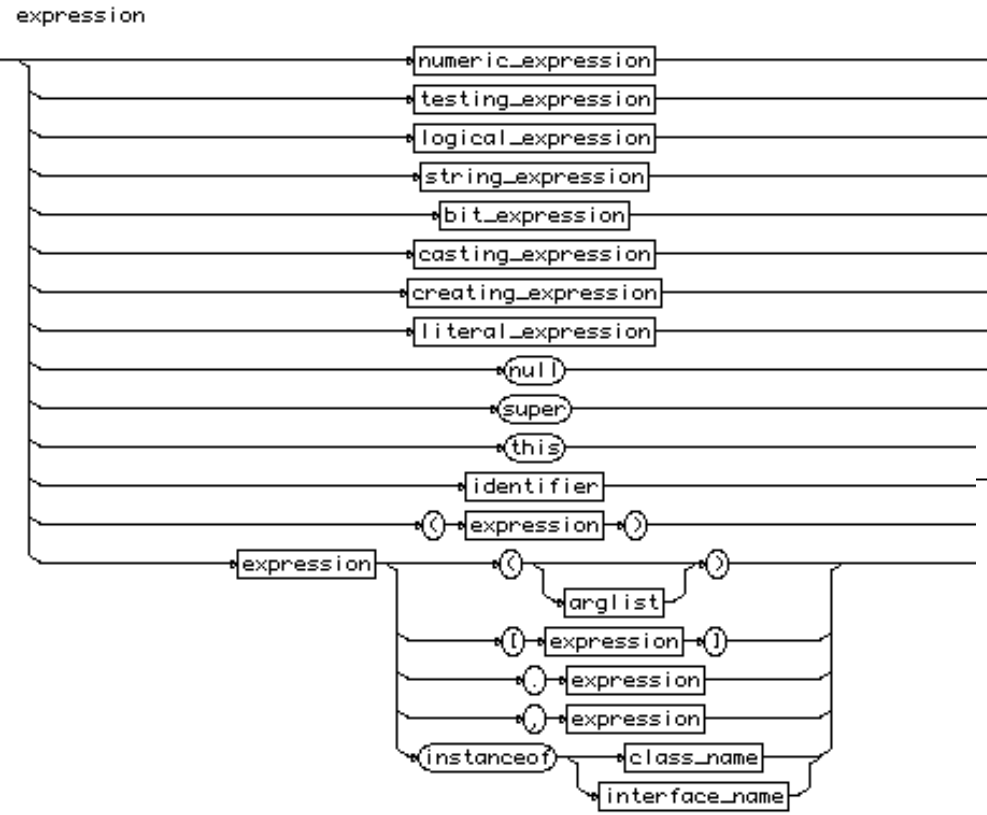
- Programa principal

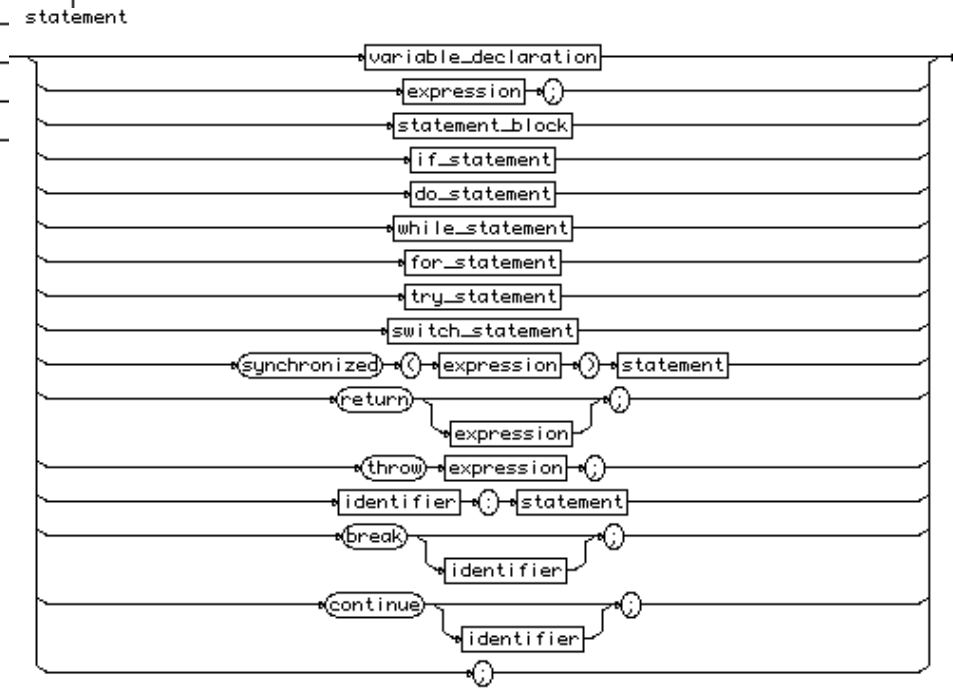
<S> ::= a<A>d
<A> ::= c<A> | e<B>
<B> ::= f | g

```
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



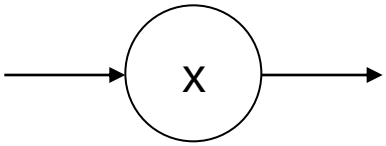


ASD preditiva recursiva

- Regras de transformação
  - Regras gramaticais → grafos sintáticos

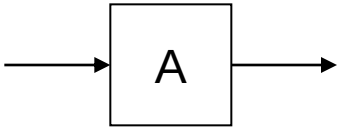
1. Toda regra é mapeada em um grafo

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

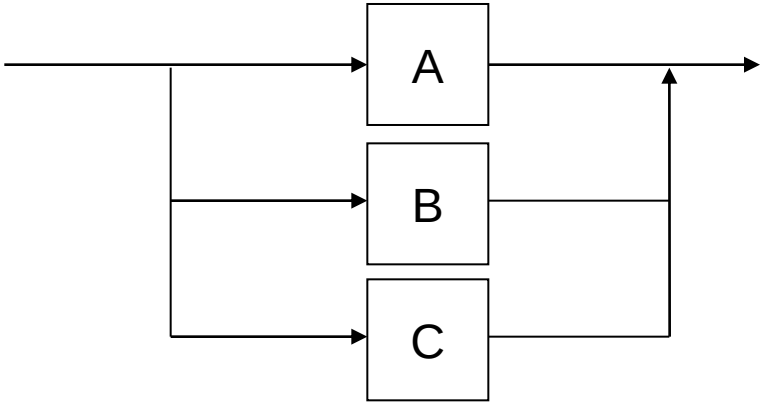


ASD preditiva recursiva

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

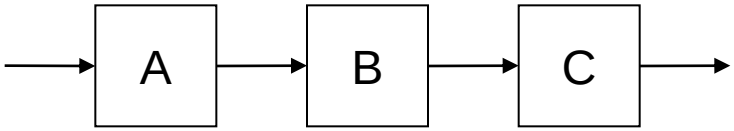


4. Alternativas são representadas como

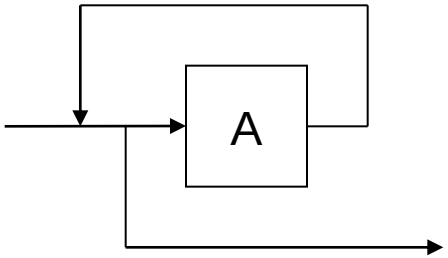


ASD preditiva recursiva

5. Uma seqüência A B C é mapeada em



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



ASD preditiva recursiva

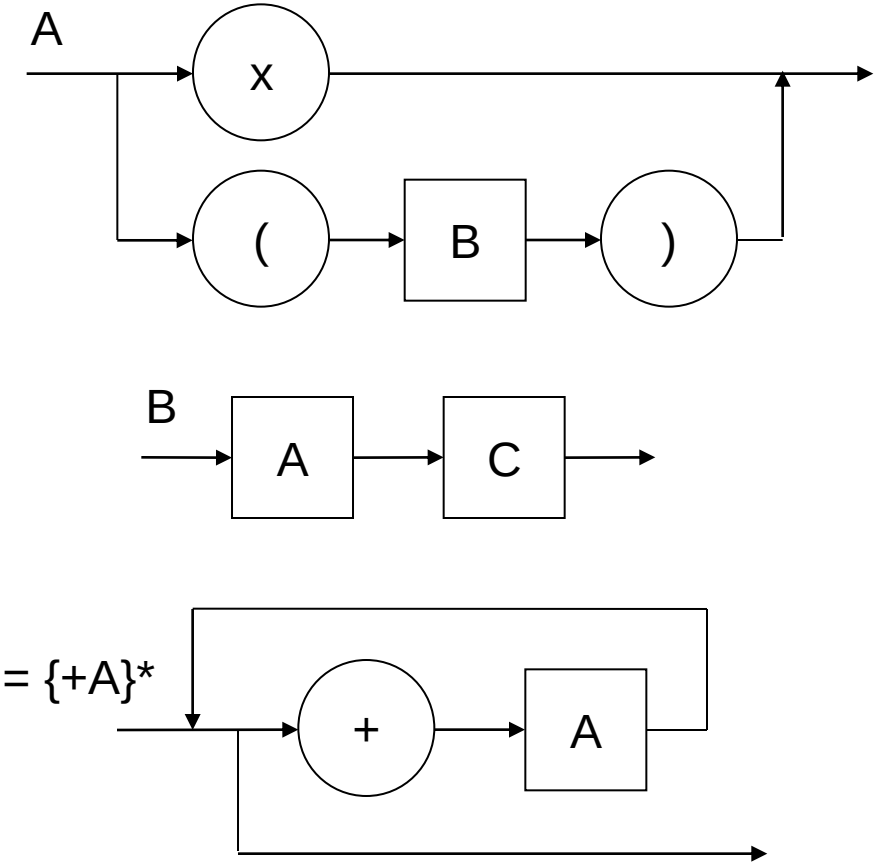
- Exercício

<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

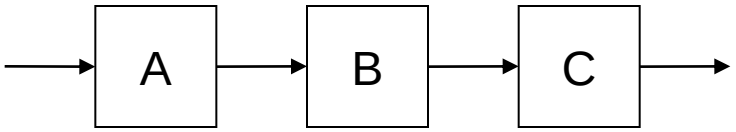
- Regras de tradução
  - Grafos sintáticos → procedimentos

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

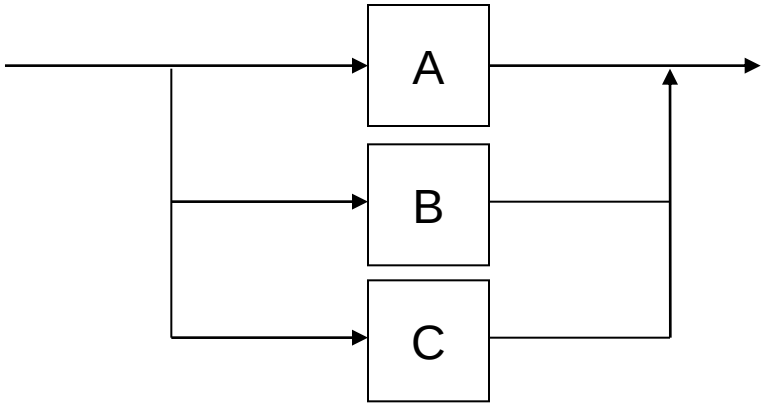


origina o procedimento

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

ASD preditiva recursiva

4. A alternativa

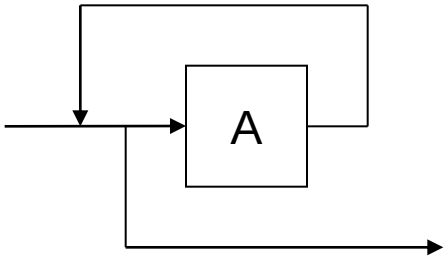


origina o procedimento

```
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

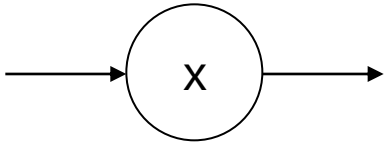


origina o procedimento

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

ASD preditiva recursiva

6. O terminal

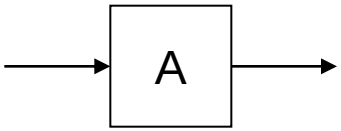


origina

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

ASD preditiva recursiva

7. O não terminal

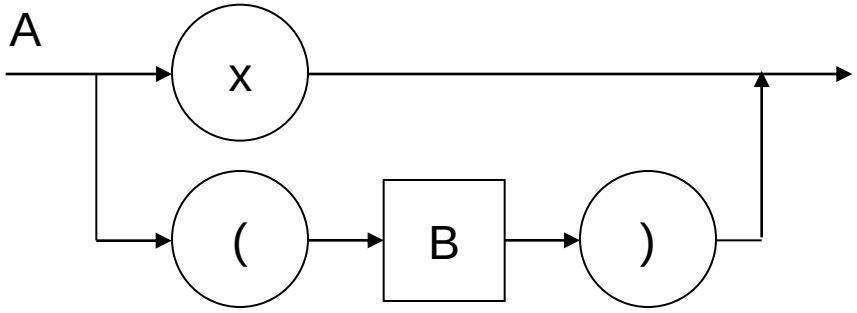


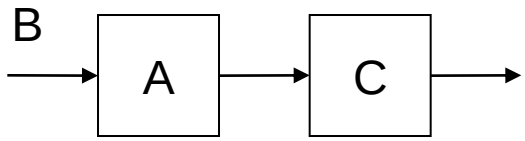
origina

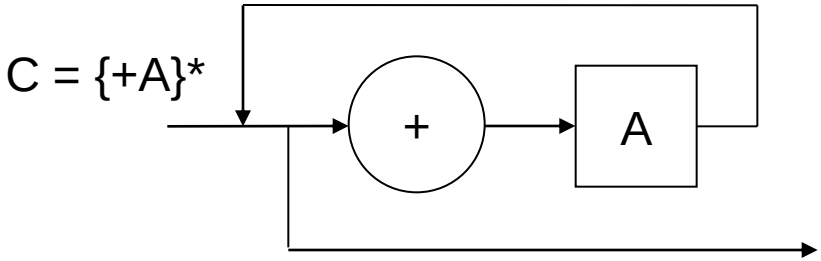
```
begin
        A;
end
```

ASD preditiva recursiva

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

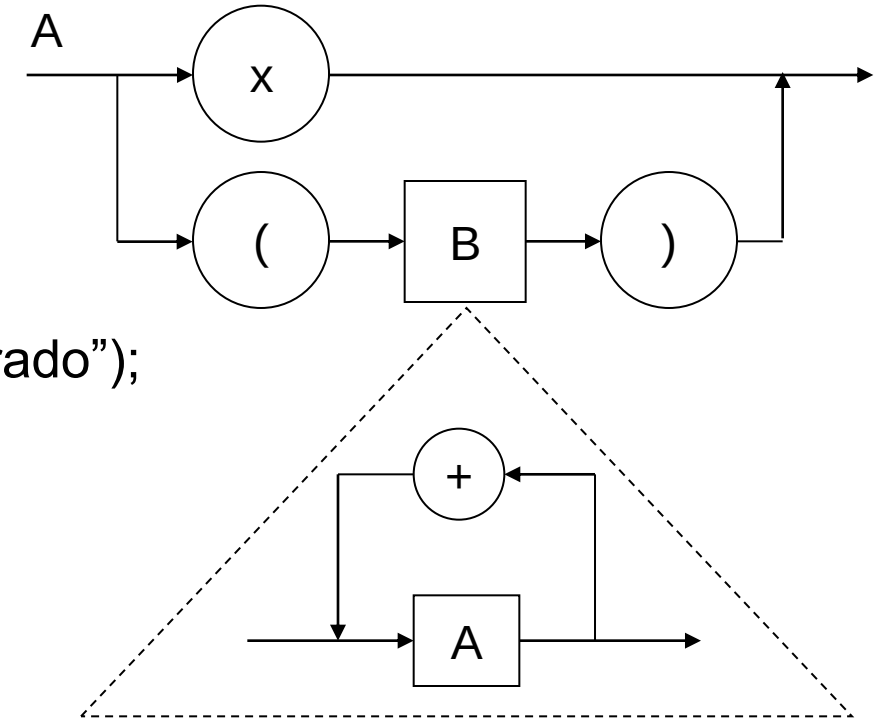






ASD preditiva recursiva

```
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>]
  - <T'> ::= *<T> | λ ➔ <T´> ::= [*<T>]
  - <X> ::= -<F> | λ ➔ <X> ::= [-<F>]