



Compiladores

Aula 5

Celso Olivete Júnior

olivete@fct.unesp.br



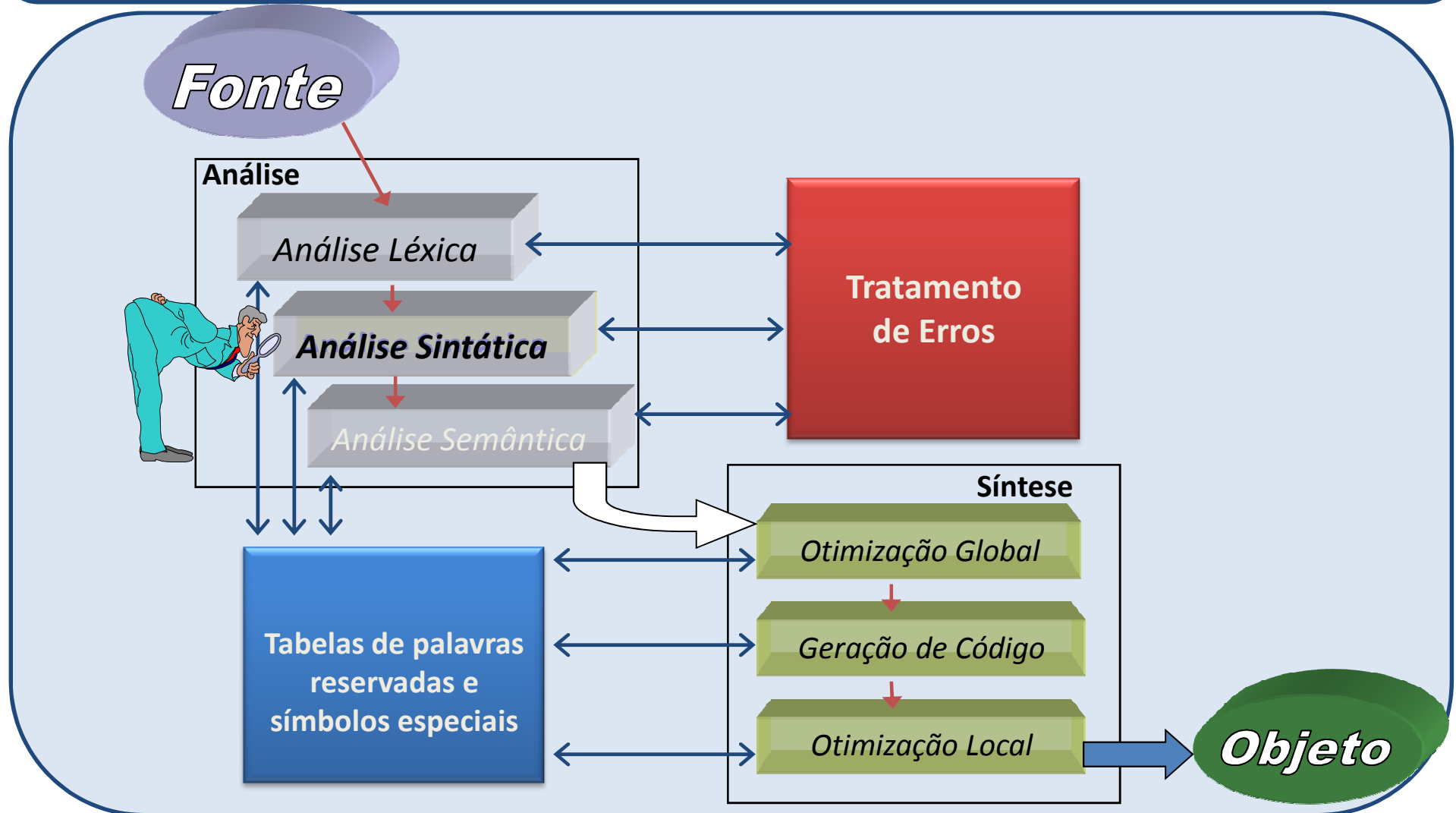
Na aula passada

- ❑ Conjuntos **Primeiro**(*First*) e **Seguidor**(*Follow*)

□ Análise Sintática

- Análise Sintática Descendente
 - Recursividade
 - Fatoração

Arquitetura básica de um compilador





Análise sintática

definições e características

- ❑ É a segunda fase de um compilador.
- ❑ É responsável pela leitura do fluxo de **tokens** produzido pelo analisador léxico **checando** se tal **fluxo pode ser gerado** pela **gramática** da linguagem-fonte.
- ❑ Também é **chamada** de **análise gramatical** ou ***parsing***.
- ❑ Pode produzir uma árvore gramatical que poderá ser usada na geração de código.

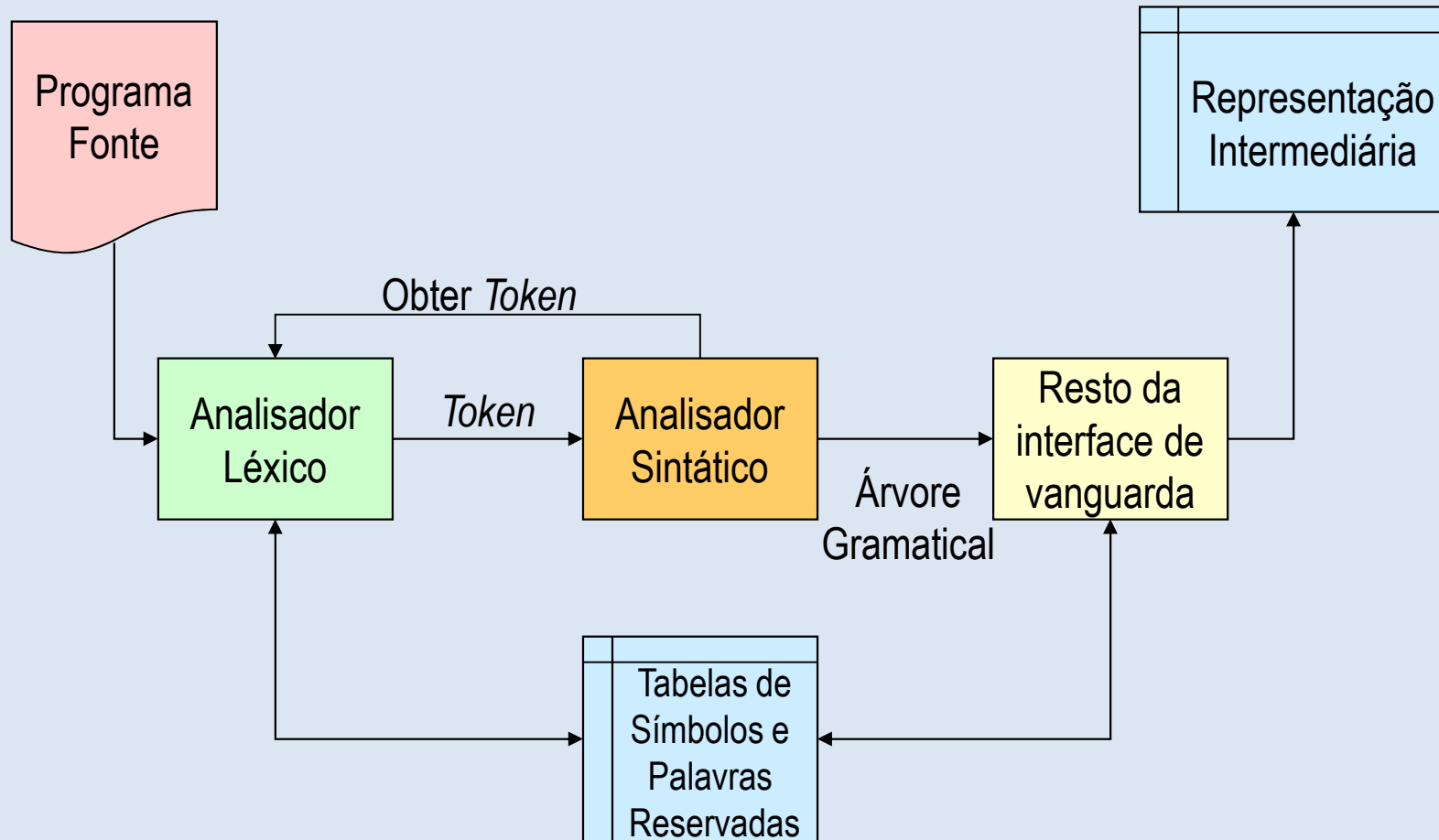


Análise sintática

definições e características

- ❑ Permite determinar se a tradução de um programa-fonte em códigos-objeto é possível localizando erros gramaticais.
- ❑ É usualmente tratada como uma fase separada por questões de simplicidade e eficiência.

Posicionamento da análise sintática



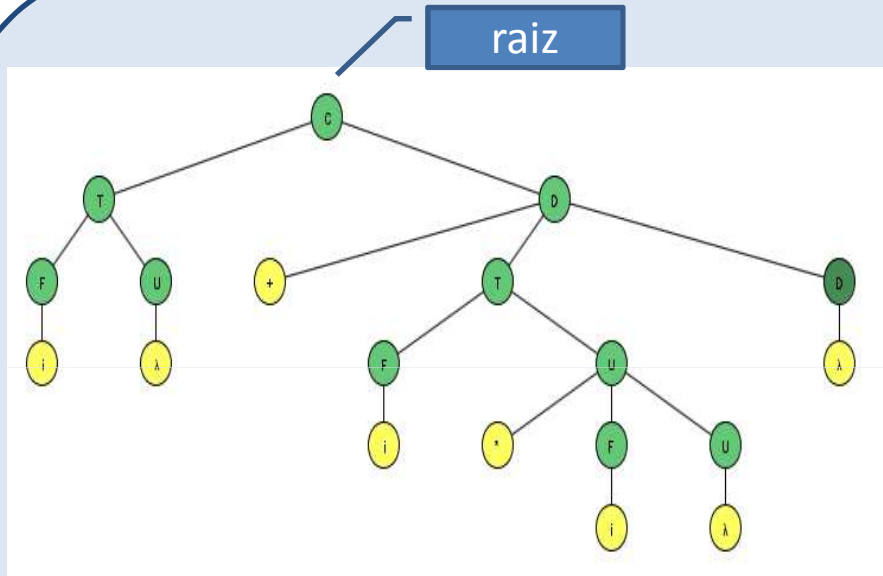


Tipos de analisadores sintáticos

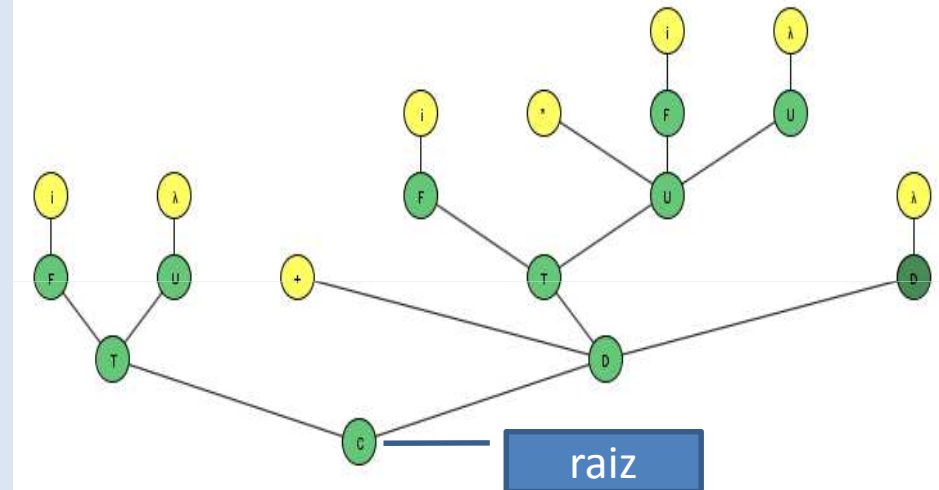
❑ Existem dois tipos principais:

- **Analisadores Sintáticos Descendentes (*Top-Down* - *ASD*):** constroem a árvore gramatical do topo (raiz) para o fundo (folhas).
- **Analisadores Sintáticos Ascendentes (*Bottom-Up* - *ASA*):** constroem a árvore gramatical do fundo (folhas) para o topo (raiz).

Tipos de analisadores sintáticos



Estratégia de Construção
ASD (Top-Down)



Estratégia de Construção
ASA (Bottom-Up)

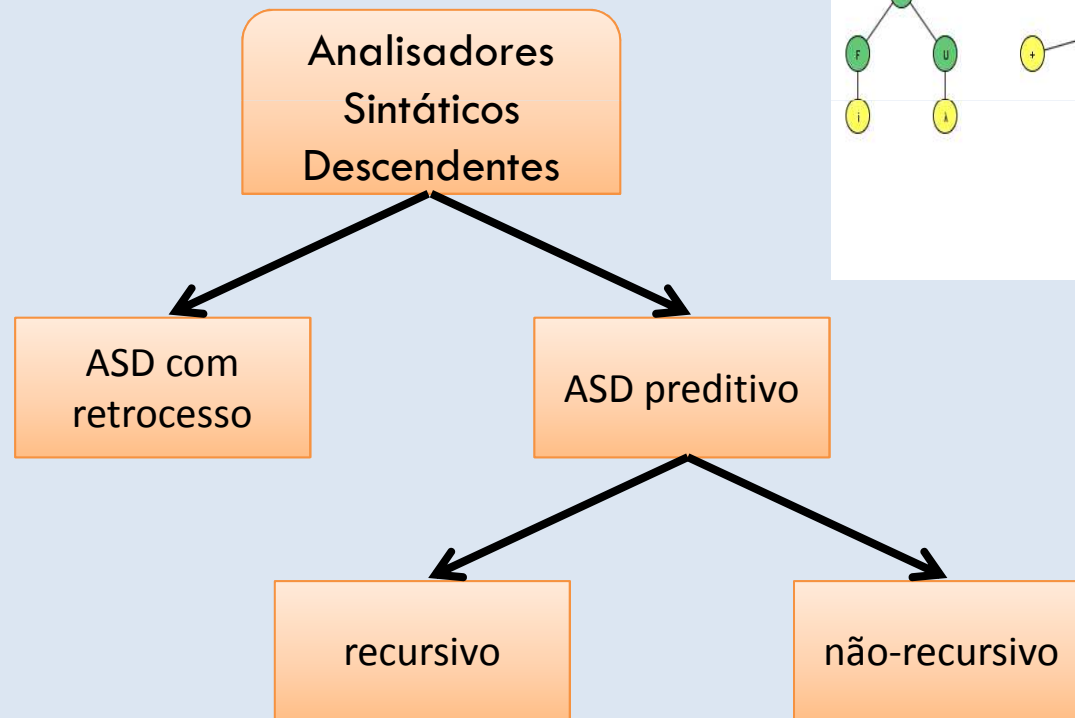
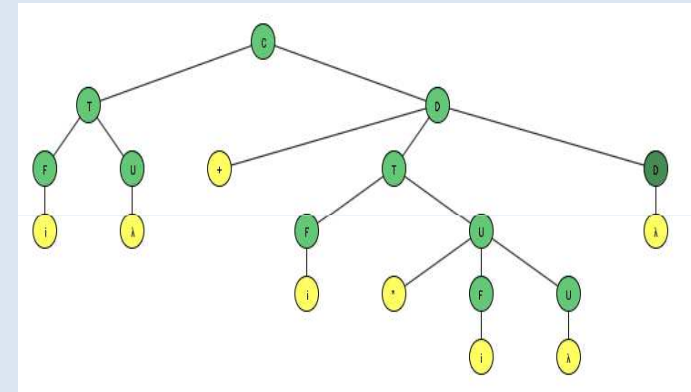


Tipos de analisadores sintáticos

- ❑ **ASD** e **ASA** são aplicáveis em quase todos os casos, mas com eficiência apenas a certas subclasses gramaticais
 - **LL** (*Left to Right - Leftmost derivation*)
 - **LR** (*Left to Right - Rightmost derivation*)
- ❑ A maioria das linguagens de programação consegue ser descritas por gramáticas LL (convenientes para implementação manual) e LR (adequadas para construção automática).

Analísadores Sintáticos Descendentes (ASD) e seus tipos

- Parte-se do símbolo inicial da gramática objetivando atingir as folhas.





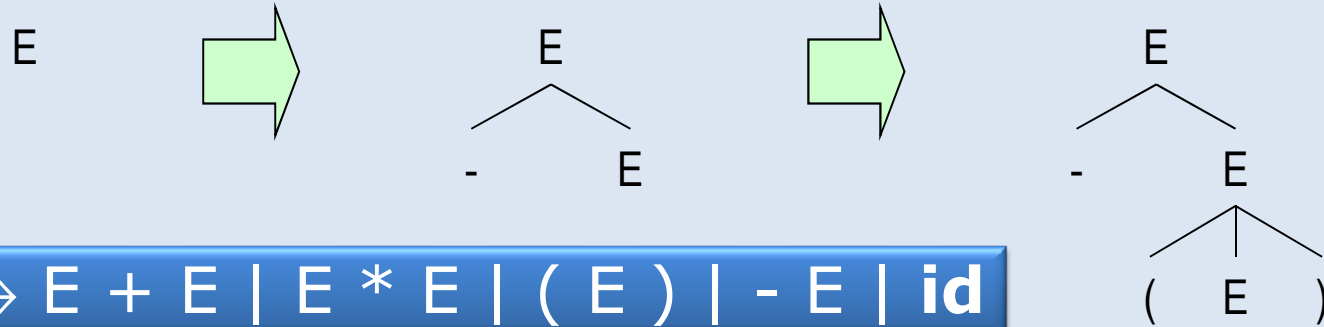
Analísadores Sintáticos Descendentes (ASD)

- ❑ ASD funcionam tal como **derivações mais à esquerda**
- ❑ As árvores sintáticas são representações gráficas das derivações possíveis numa certa ordem
- ❑ A substituição de *não-terminais* a cada passo corresponde a expansão de um nó da árvore sintática em construção, por exemplo, cada nó interior é rotulado por um *não-terminal* e cada folha é rotulada por um *terminal* ou ϵ .

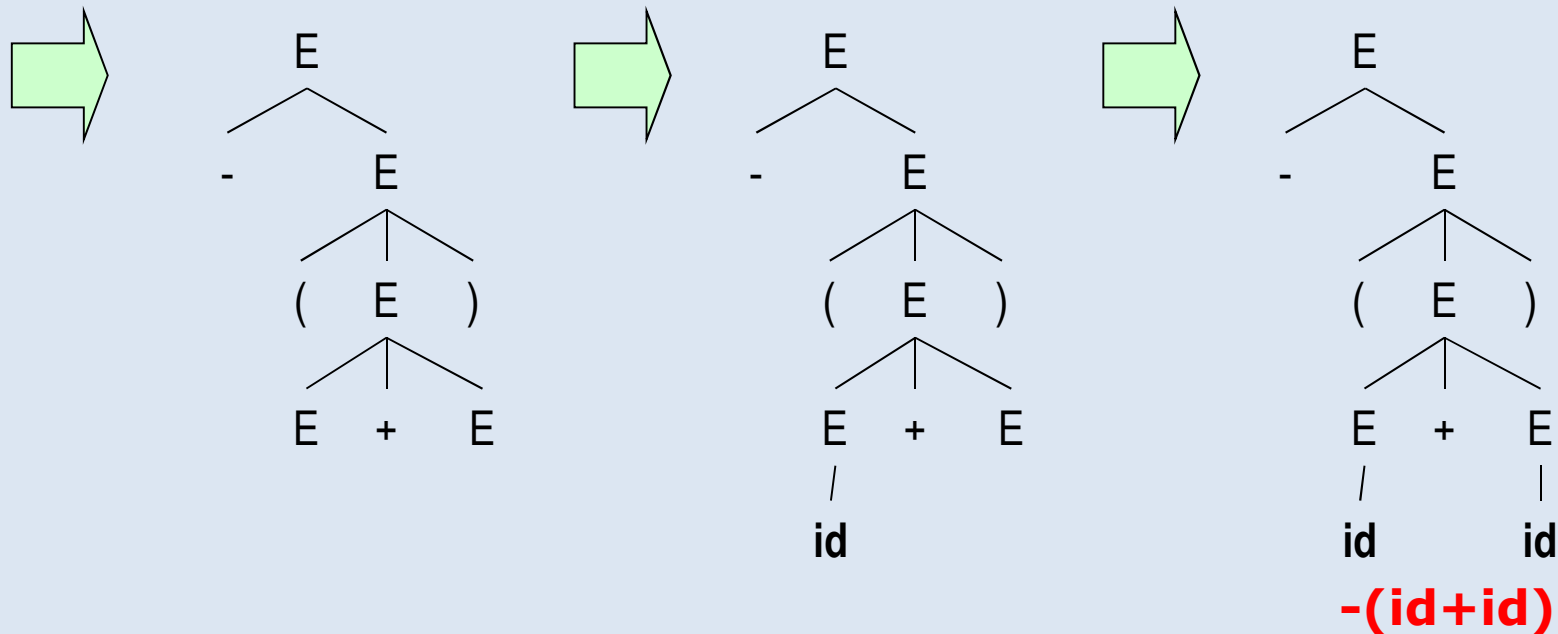


unesp

ASD



$G: E \rightarrow E + E \mid E * E \mid (E) \mid - E \mid id$



ASD

JFLAP : <untitled1>

File Input Test Convert Help

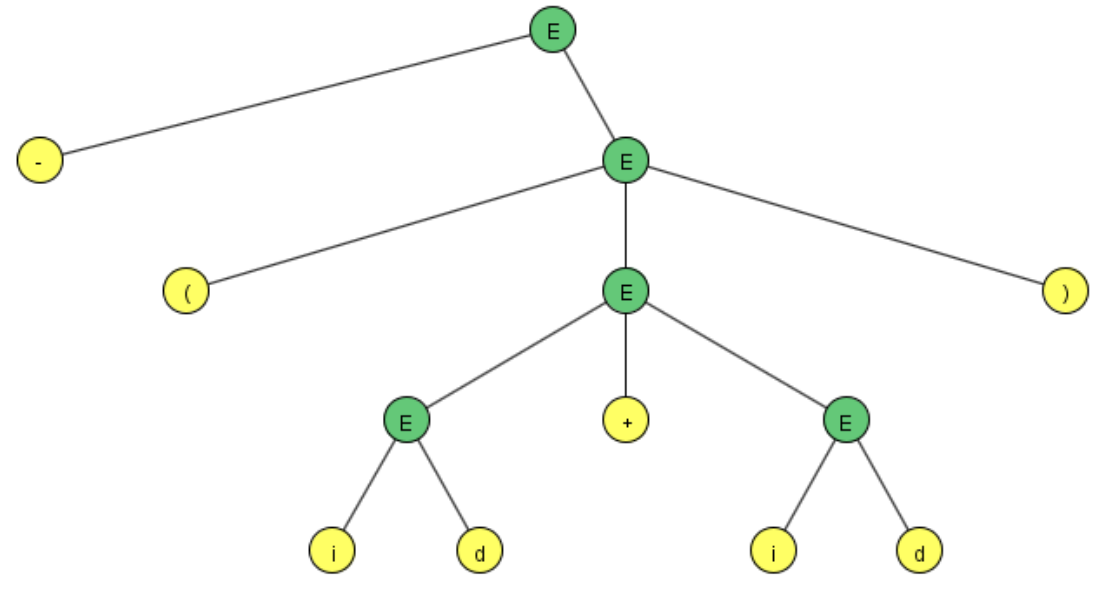
Editor Brute Parser

Table Text Size

Start Pause Step Noninverted Tree

Input: (id+id)
String accepted! 9 nodes generated.

LHS		RHS
E	→	E+E
E	→	E*E
E	→	(E)
E	→	-E
E	→	id



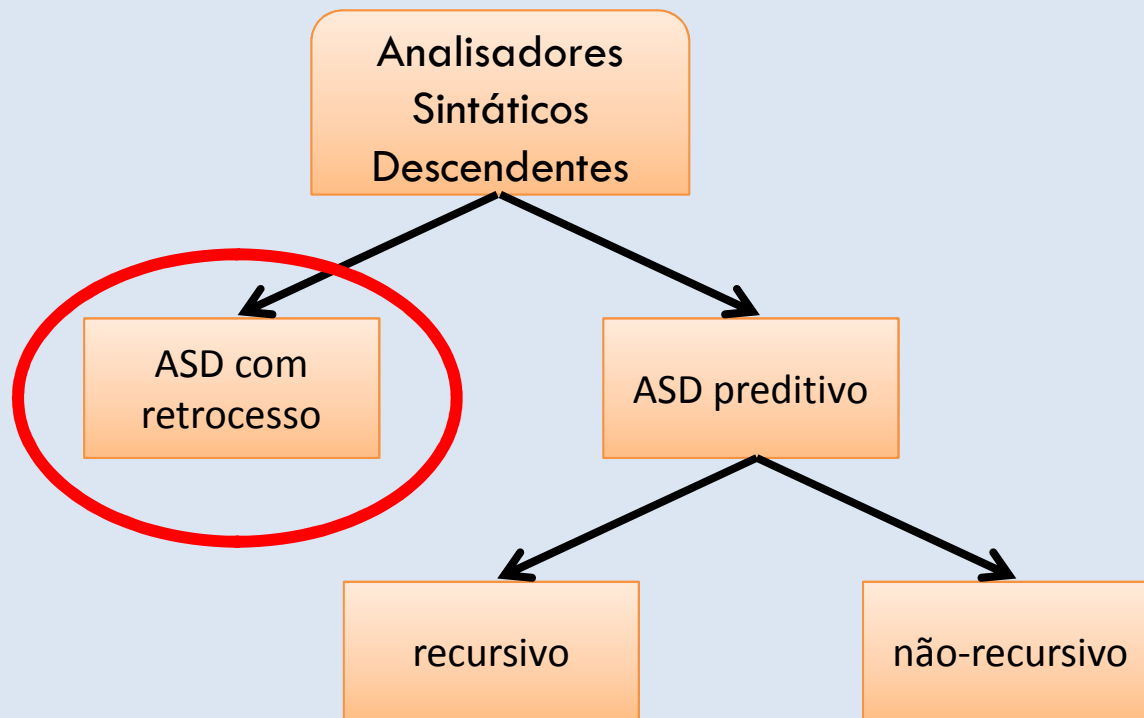
```

graph TD
    E1((E)) --- E2((E))
    E1 --- Minus((-))
    E2 --- LParen("(")
    E2 --- E3((E))
    E2 --- RParen(")")
    E3 --- E4((E))
    E3 --- Plus("+")
    E3 --- E5((E))
    E4 --- I1("i")
    E4 --- D1("d")
    E5 --- I2("i")
    E5 --- D2("d")
  
```

Derived id from E. Derivations complete.



ASD

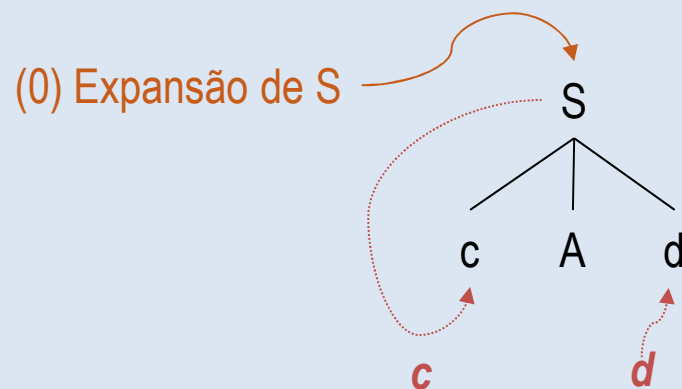


ASD com retrocesso

- ❑ Método que objetiva encontrar uma derivação mais à esquerda para uma cadeia de entrada procurando construir uma árvore gramatical a partir da raiz criando nós em pré-ordem;
- ❑ Características
 - ❑ Exploratório: tenta todas as possibilidades
 - ❑ Ineficiente
- ❑ Funcionamento
 - ❑ A cada passo, escolhe uma regra e aplica
 - ❑ Se falhar em algum ponto, retrocede e escolhe uma outra regra
 - ❑ O processo termina quando a cadeia é reconhecida ou quando as regras se esgotaram e a cadeia não foi reconhecida

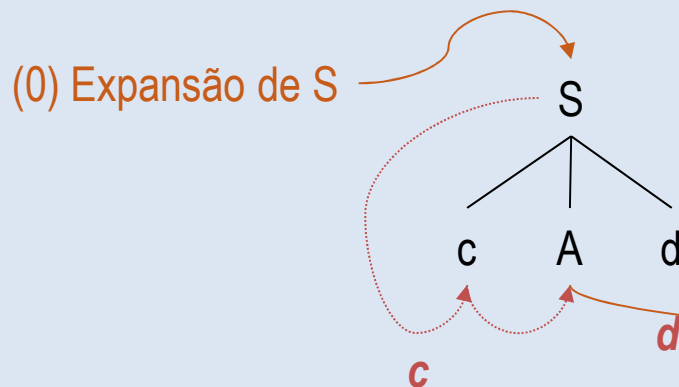
ASD com retrocesso

- ❑ Considere a gramática:
 $S \rightarrow cAd$
 $A \rightarrow ab \mid a$
- ❑ Para a cadeia **w = cad** temos:

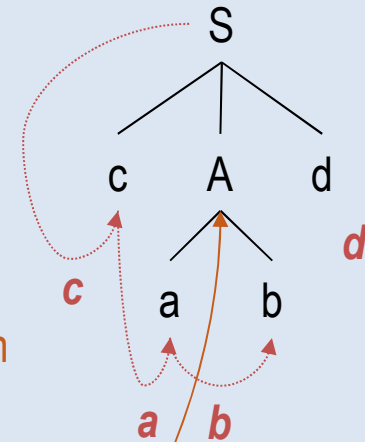


ASD com retrocesso

- ❑ Considere a gramática:
 $S \rightarrow cAd$
 $A \rightarrow ab \mid a$
- ❑ Para a cadeia **w = cad** temos:

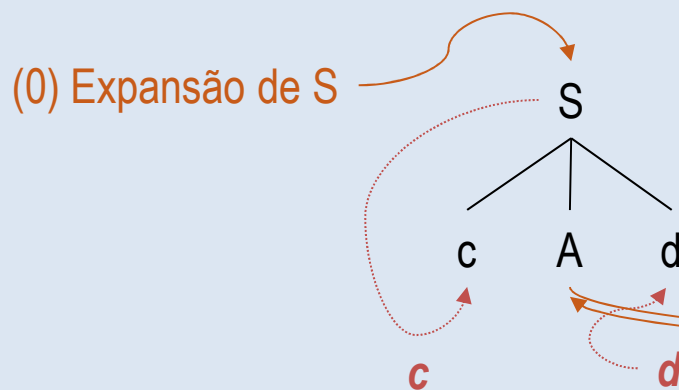


(1) Expansão A com 1a. produção



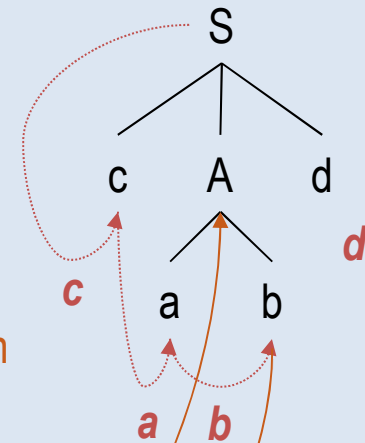
ASD com retrocesso

- ❑ Considere a gramática:
 $S \rightarrow cAd$
 $A \rightarrow ab \mid a$
- ❑ Para a cadeia **w = cad** temos:



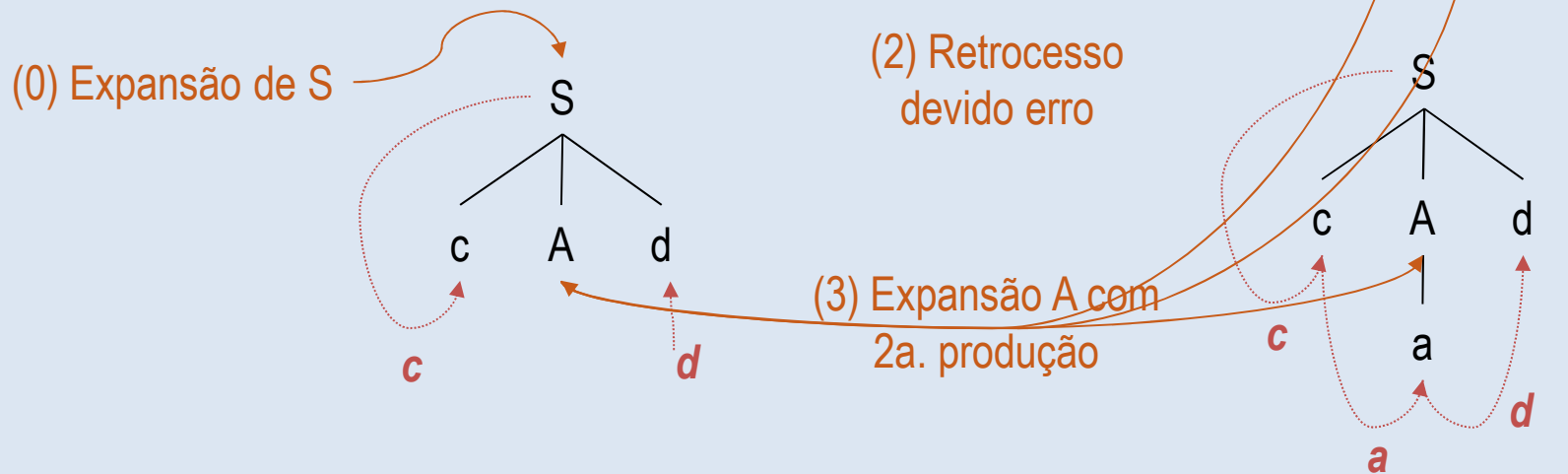
(1) Expansão A com 1a. produção

(2) Retrocesso devido erro



ASD com retrocesso

- ❑ Considere a gramática:
 $S \rightarrow cAd$
 $A \rightarrow ab \mid a$
- ❑ Para a cadeia **w = cad** temos:



ASD com retrocesso

- ❑ Como visto no exemplo anterior:
 - a construção da árvore gramatical começa de cima para baixo usando o símbolo de partida;
 - a medida em que são encontrados os símbolos *não-terminais* eles são expandidos;
 - da mesma forma são consumidos, um a um, e os símbolos *terminais* encontrados;
 - quando um símbolo *terminal* não tem correspondência com a produção em uso ocorre o retrocesso - *tentativa e erro*.
- ❑ O número de **derivações** pode ser uma função **exponencial** do tamanho da cadeia

ASD com retrocesso

- ❑ Usualmente é implementada como um conjunto de funções cooperativas
- ❑ Cada construção da linguagem (uma regra BNF) é reconhecida pelo código de uma função

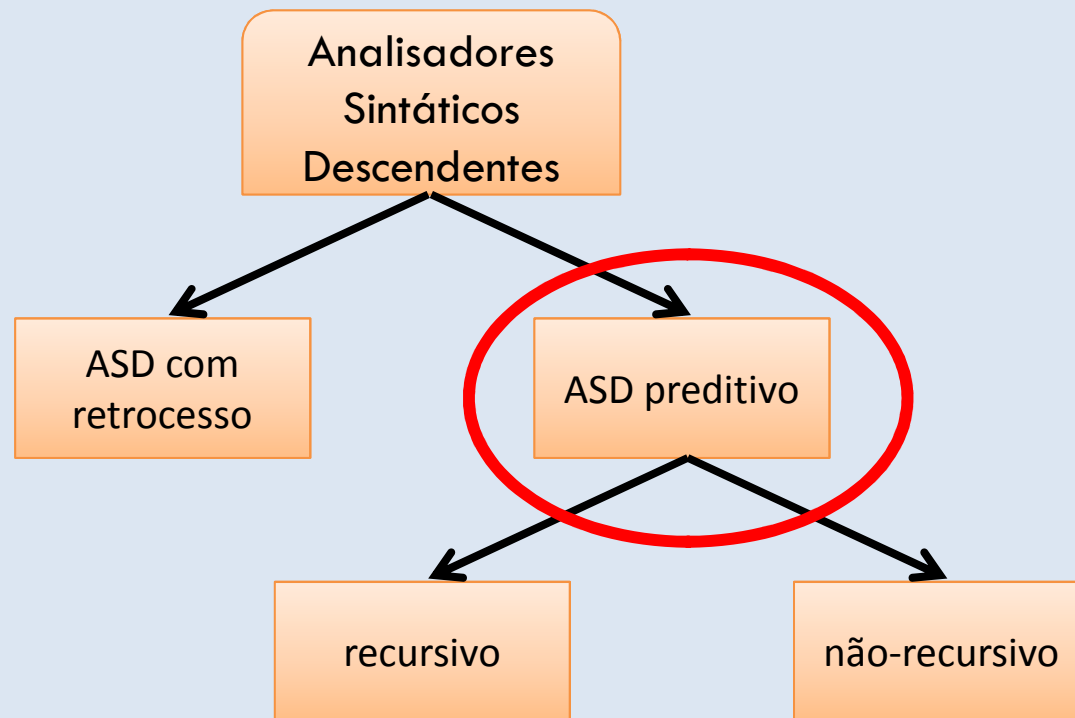
ASD

- ❑ Se a gramática for cuidadosamente escrita de forma que:
 - não haja recursão à esquerda e
 - suas produções estejam adequadamente fatoradas.

- ❑ Pode-se construir um ASD preditivo, ou seja, que não necessite operações de retrocesso.



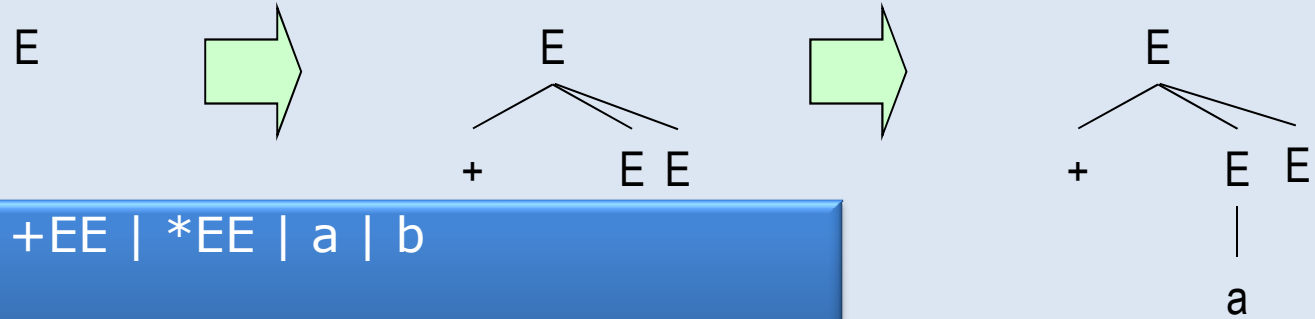
ASD



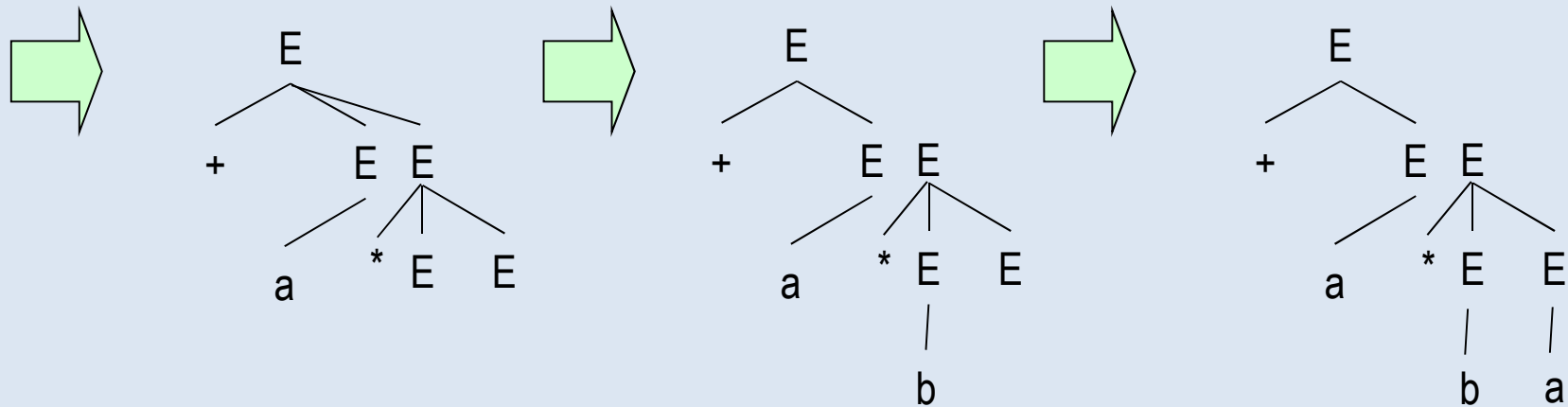
ASD Preditiva

- ❑ Dada uma gramática sem recursividade à esquerda e fatorada é possível:
 - ❑ Olhar para o primeiro símbolo da entrada e saber qual regra aplicar

ASD Preditiva



$G: E \rightarrow +EE \mid *EE \mid a \mid b$
 Reconhecer a entrada: $+a*ba$



+a*ba

☐ Restrições

1. **não haja recursão à esquerda** → O lado direito das produções devem começar por terminais
2. **suas produções estejam adequadamente fatoradas** → Para um não-terminal qualquer, não devem existir duas regras que comecem com um mesmo terminal

ASD Preditiva

1. **não haja recursão à esquerda** → O lado direito das produções devem começar por terminais

□ Caso a gramática apresente nos lados direitos regras de produção que iniciam com não-terminais, seus conjuntos **Primeiro (*First*)** devem ser disjuntos



ASD Preditiva

- Dada a gramática em EBNF - e os conjuntos **Primeiro** - que obedece a restrição(1), mostre o reconhecimento para a cadeia: **abcdad**

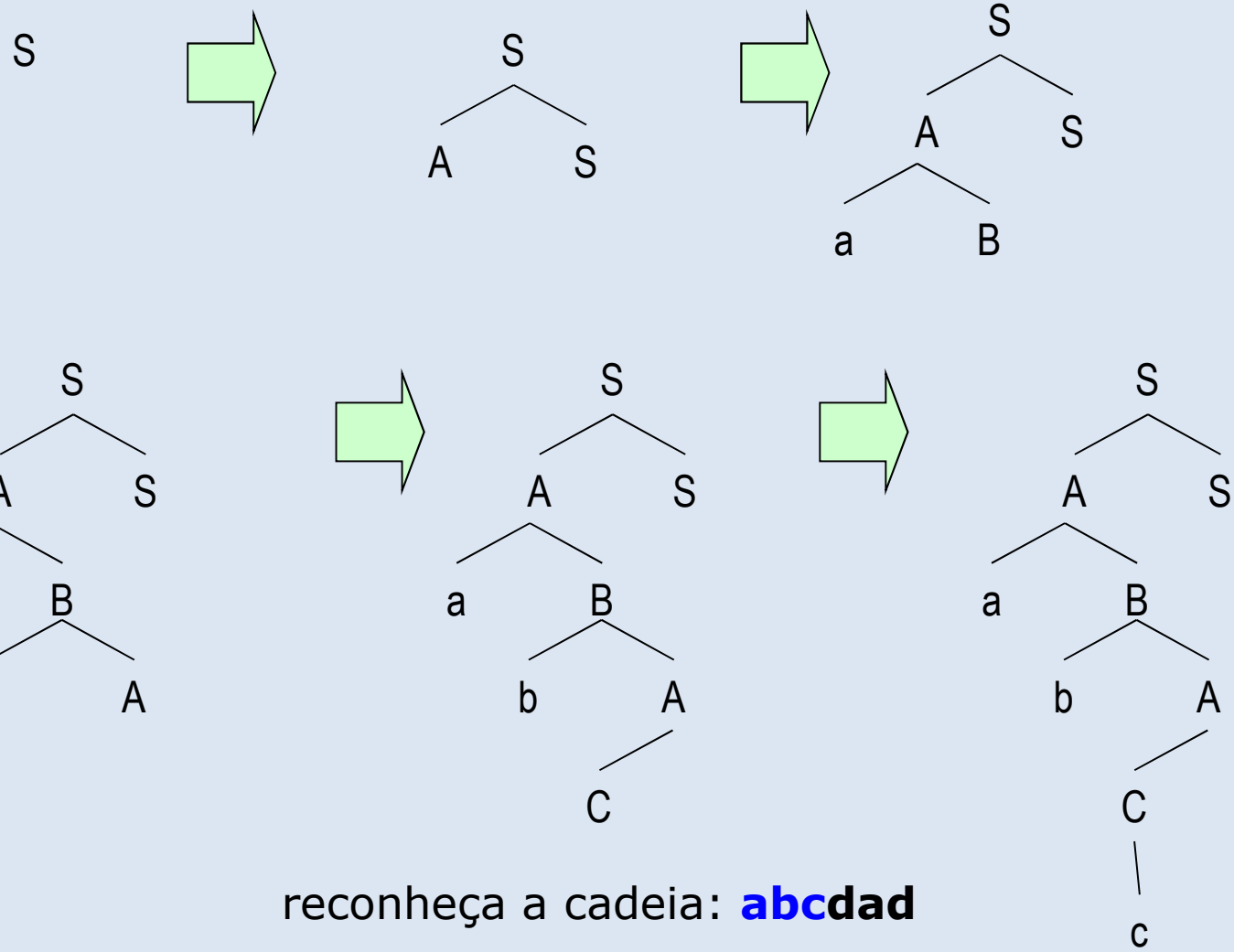
```
<S> ::= <A><S> | <B><A>  
<A> ::= a<B> | <C>  
<B> ::= b<A> | d  
<C> ::= c
```

```
Primeiro(S) = {a,b,c,d}  
Primeiro(A) = {a, c}  
Primeiro(B) = {b, d}  
Primeiro(C) = {c}
```

$\langle S \rangle ::= \langle A \rangle \langle S \rangle \mid \langle B \rangle \langle A \rangle$
 $\langle A \rangle ::= a \langle B \rangle \mid \langle C \rangle$
 $\langle B \rangle ::= b \langle A \rangle \mid d$
 $\langle C \rangle ::= c$

ASD Preditiva

$\text{Primeiro}(S) = \{a, b, c, d\}$
 $\text{Primeiro}(A) = \{a, c\}$
 $\text{Primeiro}(B) = \{b, d\}$
 $\text{Primeiro}(C) = \{c\}$



reconheça a cadeia: **abcdad**

$\langle S \rangle ::= \langle A \rangle \langle S \rangle \mid \langle B \rangle \langle A \rangle$

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

$\langle B \rangle ::= b \langle A \rangle \mid d$

$\langle C \rangle ::= c$

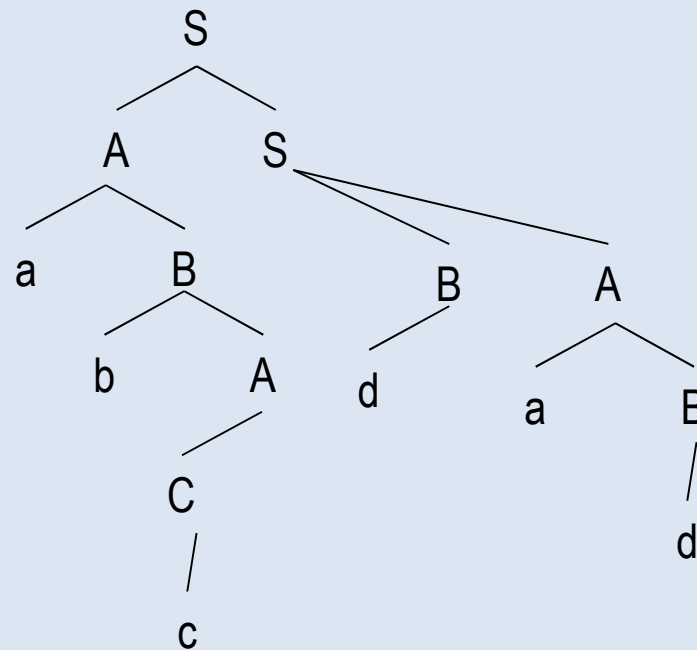
ASD Preditiva

Primeiro(S) = {a,b,c,d}

Primeiro(A) = {a, c}

Primeiro(B) = {b, d}

Primeiro(C) = {c}



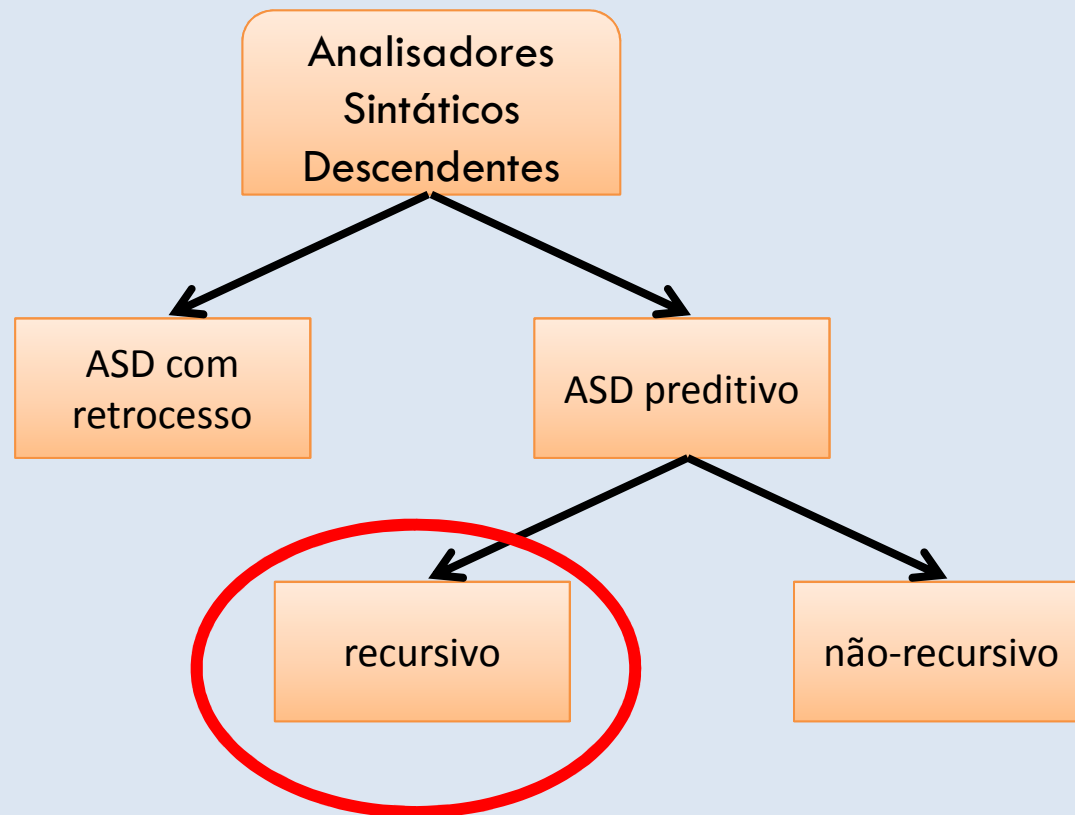
reconheça a cadeia: **abcdad**

ASD Preditiva

- ❑ As gramáticas que obedecem as restrições:
 1. não apresenta recursão à esquerda
 2. suas produções estejam adequadamente fatoradas → para um não terminal qualquer, não devem existir duas regras que comecem com um mesmo terminal
- ❑ São chamadas de:
 - **LL(1)** (*Left to Right - Leftmost derivation*) → 1 único símbolo a frente para determinar qual regra de produção utilizar



ASD



ASD Preditivo Recursivo

- ❑ Um **ASD preditivo recursivo** é um conjunto de procedimentos possivelmente recursivos, um para cada não terminal a ser derivado
 - ❑ Requer uma gramática LL(1)



unesp

ASD Preditivo Recursivo

Exemplo

```
<E> ::= <T> + <E> | <T>  
<T> ::= <F> * <T> | <F>  
<F> ::= a | b | (<E>)
```

Note que não é LL(1)

```
procedimento ASD  
begin  
  obter_simbolo;  
  E;  
  se fim-de-cadeia OK  
  então ERRO  
end
```

```
procedimento E  
begin  
  T;  
  se (símbolo='+') então  
    obter_simbolo;  
  E;  
end
```

```
Procedimento T  
begin  
  F;  
  se (símbolo='*') então  
    obter_simbolo;  
  T;  
end
```

```
Procedimento F  
begin  
  se (símbolo='(') então  
    obter_simbolo;  
  E;  
  se (símbolo=')') então  
    obter_simbolo  
  senão ERRO;  
  senão se (símbolo='a') ou (símbolo='b')  
  então obter_simbolo  
  senão ERRO;  
end
```

ASD Preditivo Recursivo

Exemplo

```

<E> ::= <T> + <E> | <T>
<T> ::= <F> * <T> | <F>
<F> ::= a | b | (<E>)
  
```

Note que não é LL(1)

```

procedimento ASD
begin
  obter_simbolo;
  E;
  se fim-de-cadeia OK
  então ERRO
end
  
```

```

procedimento E
begin
  T;
  se (símbolo='+') então
  obter_simbolo;
  E;
end
  
```

```

Procedimento T
begin
  F;
  se (símbolo='*') então
  obter_simbolo;
  T;
end
  
```

```

Procedimento F
begin
  se (símbolo='(') então
  obter_simbolo;
  E;
  se (símbolo=')') então
  obter_simbolo
  senão ERRO;
  senão se (símbolo='a') ou (símbolo='b')
  então obter_simbolo
  senão ERRO;
end
  
```

Testar para a cadeia

a+b



ASD Preditivo Recursivo

Exemplo

□ Método formal para gerar os procedimentos

1. **Regras de transformação:** mapeamento das regras de produção em grafos sintáticos
2. **Regras de tradução:** mapeamento dos grafos em procedimentos

Gramática LL(1)

□ Exemplo

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



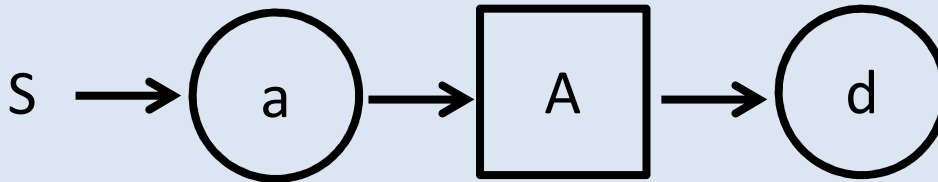
ASD Preditivo Recursivo: Grafos sintáticos

- ❑ Grafos sintáticos seguem a mesma ideia de diagramas de transição, e tem o objetivo de atuar no planejamento para a implementação de um analisador sintático preditivo.

ASD Preditivo Recursivo

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

Grafo sintático



Terminais são mapeados por um círculo

Não-terminais são mapeados por um quadrado

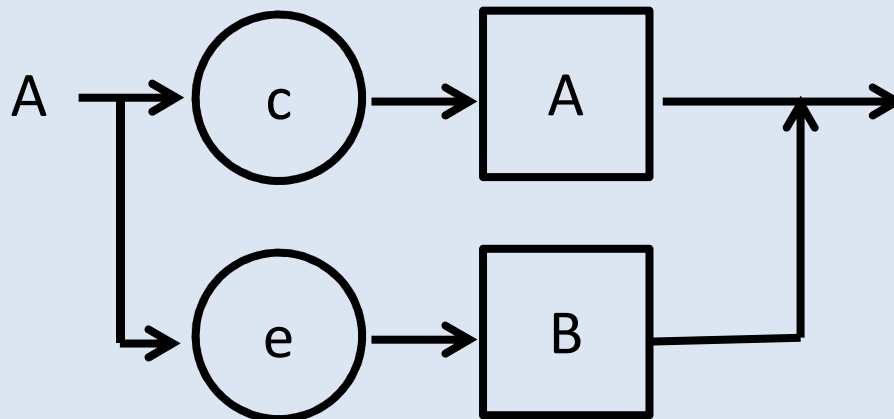
```
procedimento S
begin
  se (simbolo='a') então
    obter_simbolo;
  A;
  se (simbolo='d')
    então obter_simbolo
    senão ERRO;
  senão ERRO;
end
```

$\langle S \rangle ::= a \langle A \rangle d$
 $\langle A \rangle ::= c \langle A \rangle \mid e \langle B \rangle$
 $\langle B \rangle ::= f \mid g$

ASD Preditivo Recursivo

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

Grafo sintático



```

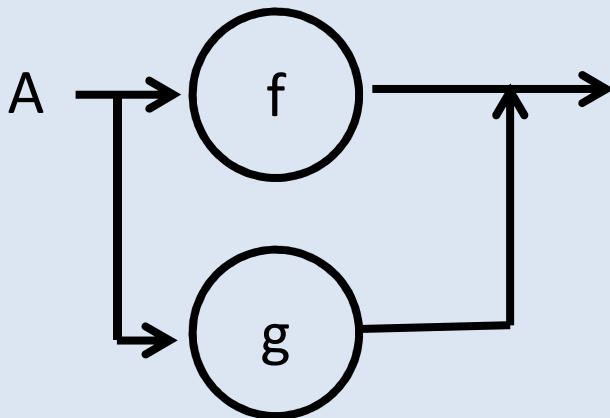
procedimento A
begin
  se (simbolo='c') então
    obter_simbolo;
    A;
  senão se (simbolo='e') então
    obter_simbolo;
    B;
  senão ERRO;
end
  
```

$\langle S \rangle ::= a\langle A \rangle d$
 $\langle A \rangle ::= c\langle A \rangle \mid e\langle B \rangle$
 $\langle B \rangle ::= f \mid g$

ASD Preditivo Recursivo

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

Grafo sintático



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

$\langle S \rangle ::= a\langle A \rangle d$
 $\langle A \rangle ::= c\langle A \rangle \mid e\langle B \rangle$
 $\langle B \rangle ::= f \mid g$

ASD Preditivo Recursivo

❑ Chamada recursiva – programa principal

```
procedimento ASD
begin
  obter_simbolo;
  S; //rotina referente ao primeiro grafo sintático
  se (terminou_cadeia)
    então SUCESSO
    senão ERRO
end
```

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

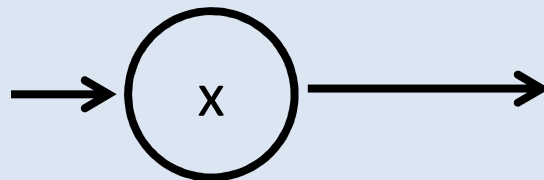
ASD Preditivo Recursivo

Exemplo

❑ Método formal para gerar os procedimentos

1. **Regras de transformação:** mapeamento das regras de um não terminal em grafos sintáticos → se possível realizar a união de grafos para maior simplicidade e eficiência

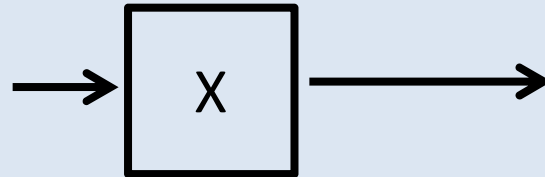
❑ Mapeando um terminal **x**



ASD Preditivo Recursivo

Exemplo

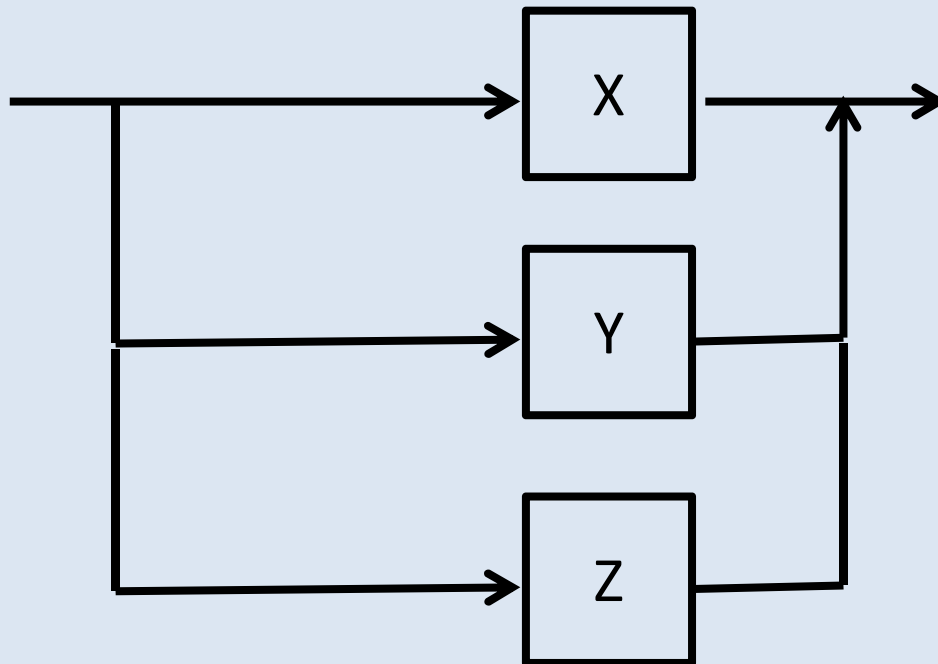
- ❑ Mapeando um não-terminal **X**



ASD Preditivo Recursivo

Exemplo

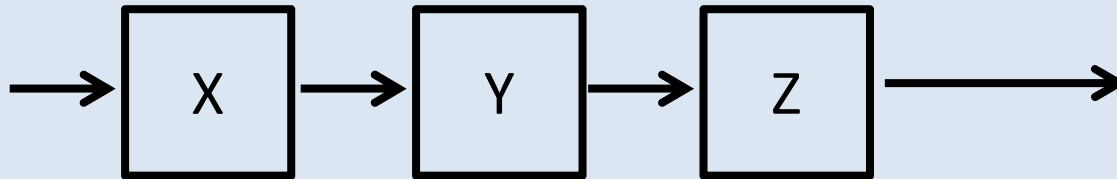
- Mapeando alternativas entre não-terminais **X**, **Y** e **Z**



ASD Preditivo Recursivo

Exemplo

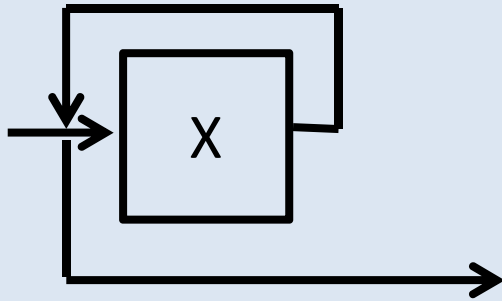
- Mapeando uma sequência de não-terminais **X**, **Y** e **Z**



ASD Preditivo Recursivo

Exemplo

- ❑ Mapeando o fechamento sobre o não-terminal **X** ou $\{X\}^*$ ou X^*





ASD Preditivo Recursivo

- ❑ Obtenha os grafos sintáticos para as regras abaixo

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

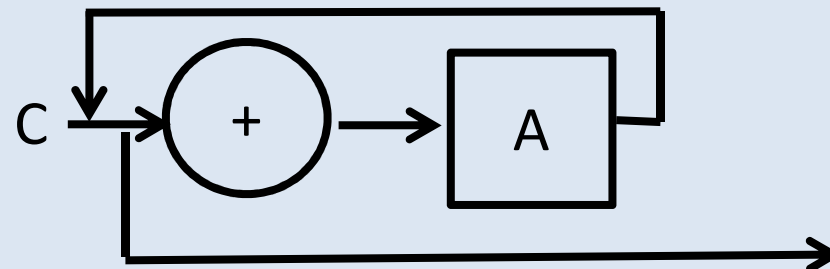
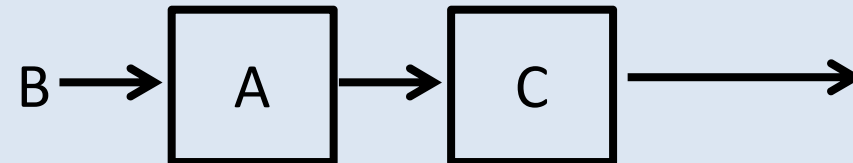
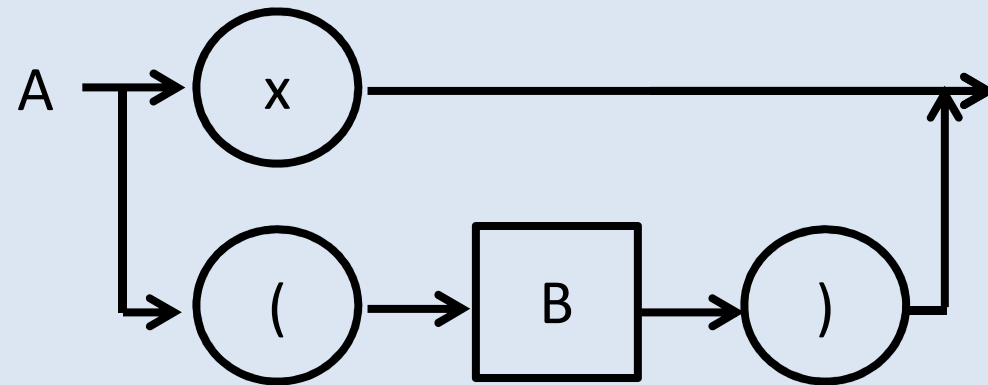
$\langle B \rangle ::= \langle A \rangle \langle C \rangle$

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

ASD Preditivo Recursivo

- Obtenha os grafos sintáticos para as regras abaixo

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





ASD Preditivo Recursivo

Exemplo

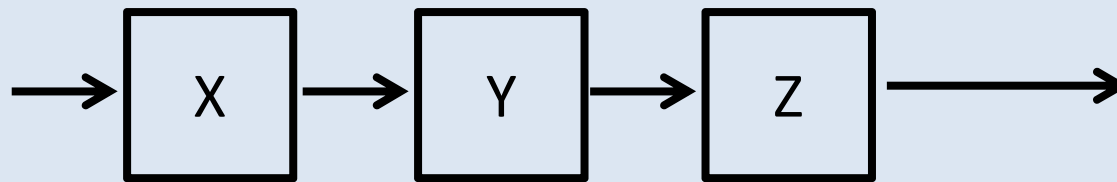
- ❑ Método formal para gerar os procedimentos
 - 2. Regras de tradução: mapeamento dos grafos em procedimentos

- ❑ Escrever um procedimento para cada grafo

ASD Preditivo Recursivo

Exemplo

- ❑ A sequência de não-terminais **X**, **Y** e **Z** gera o procedimento

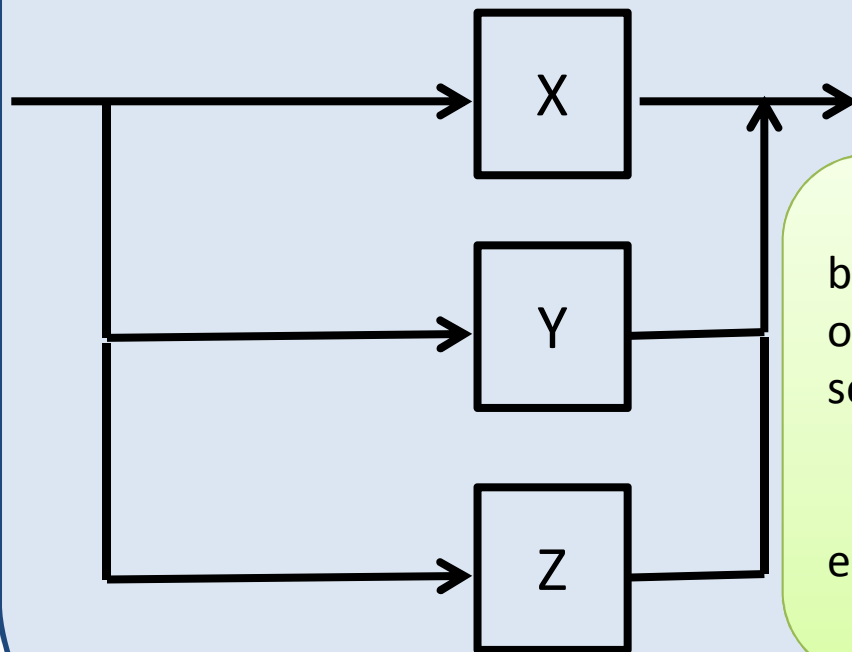


```
begin  
  X;  
  Y;  
  Z;  
end
```

ASD Preditivo Recursivo

Exemplo

- As alternativas entre não-terminais **X**, **Y** e **Z** origina o procedimento

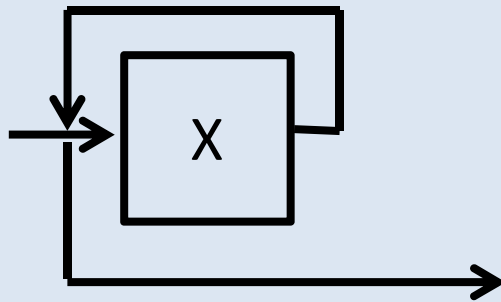


```
begin
obter_simbolo; //chamada ao analisador léxico
se (símbolo está em Primeiro(X)) então X
  senão se (símbolo está em Primeiro(Y)) então Y
  senão se (símbolo está em Primeiro(Z)) então Z
end
```

ASD Preditivo Recursivo

Exemplo

- ❑ Mapeando o fechamento sobre **X** ou $\{X\}^*$ ou X^* em procedimento

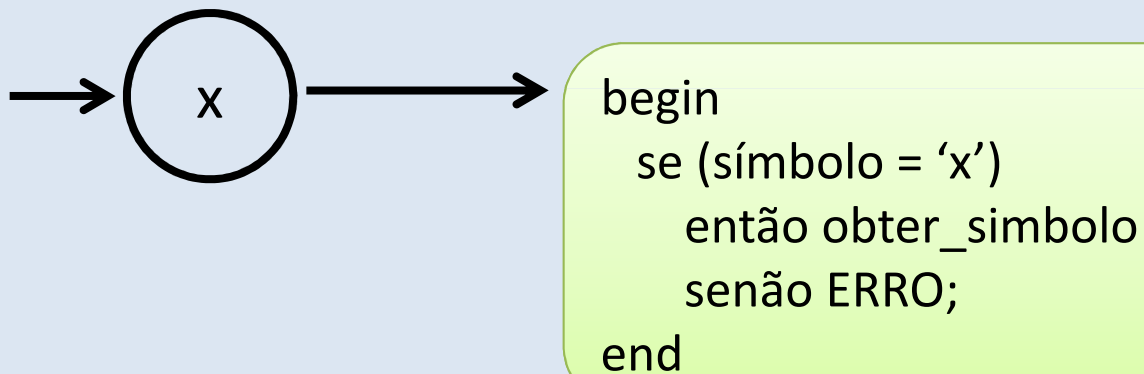


```
begin
  enquanto (símbolo está em Primeiro(X)) faça
    X;
end
```

ASD Preditivo Recursivo

Exemplo

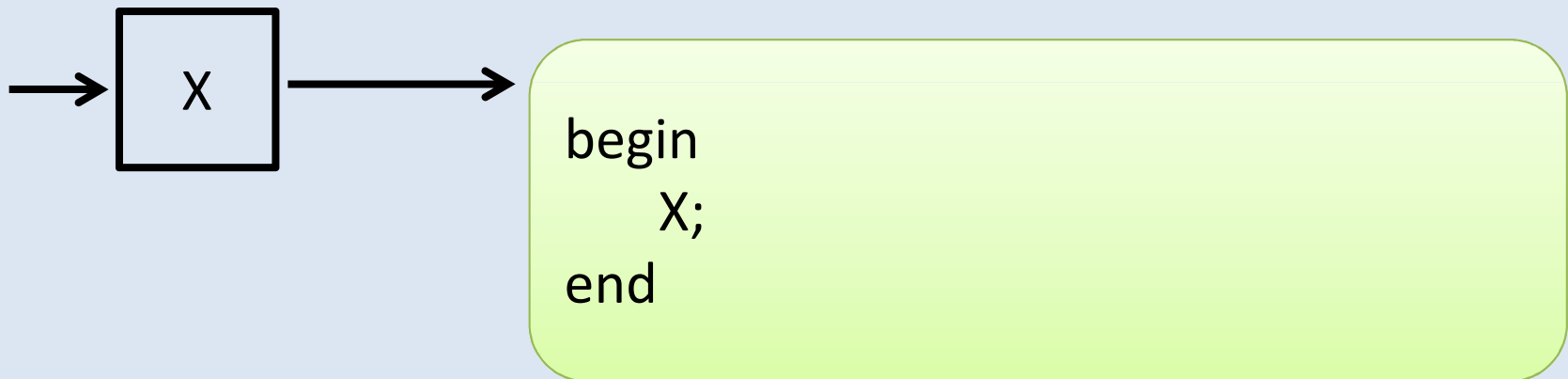
- ❑ Mapeando um terminal **x** em um procedimento



ASD Preditivo Recursivo

Exemplo

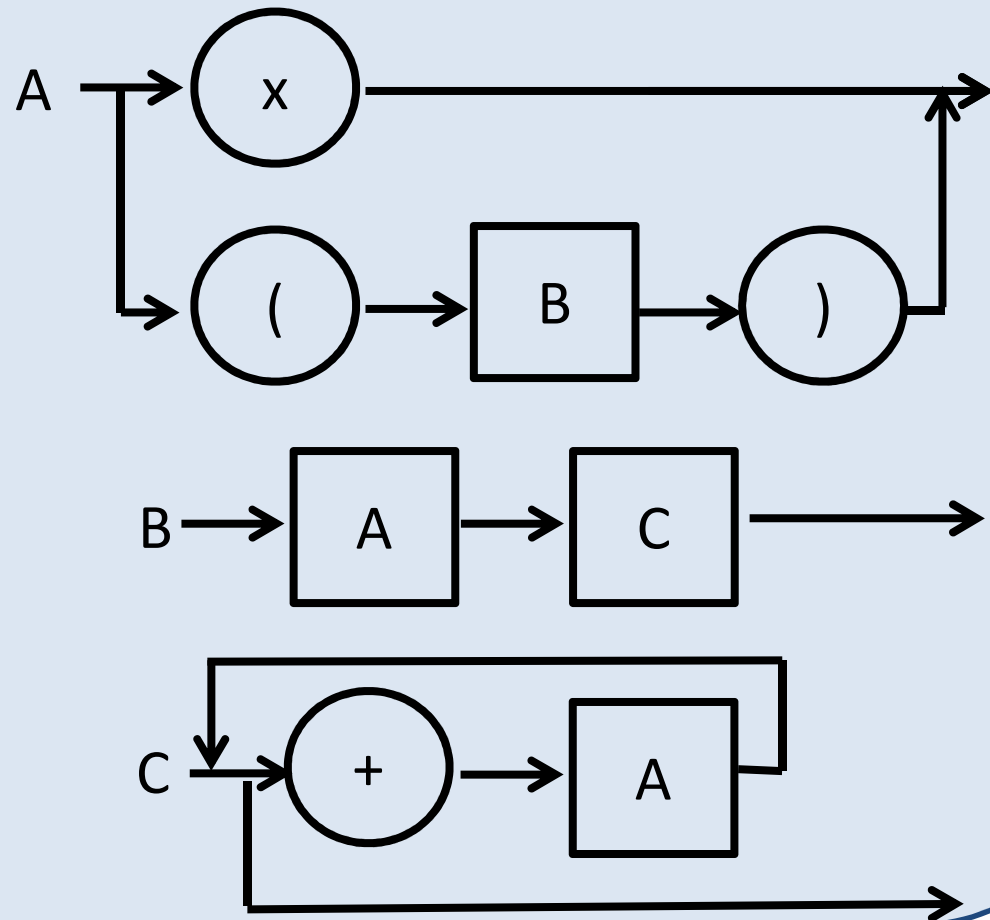
- Mapeando um não-terminal **X** em um procedimento



ASD Preditivo Recursivo

❑ Obtenha os procedimentos recursivos para os grafos abaixo

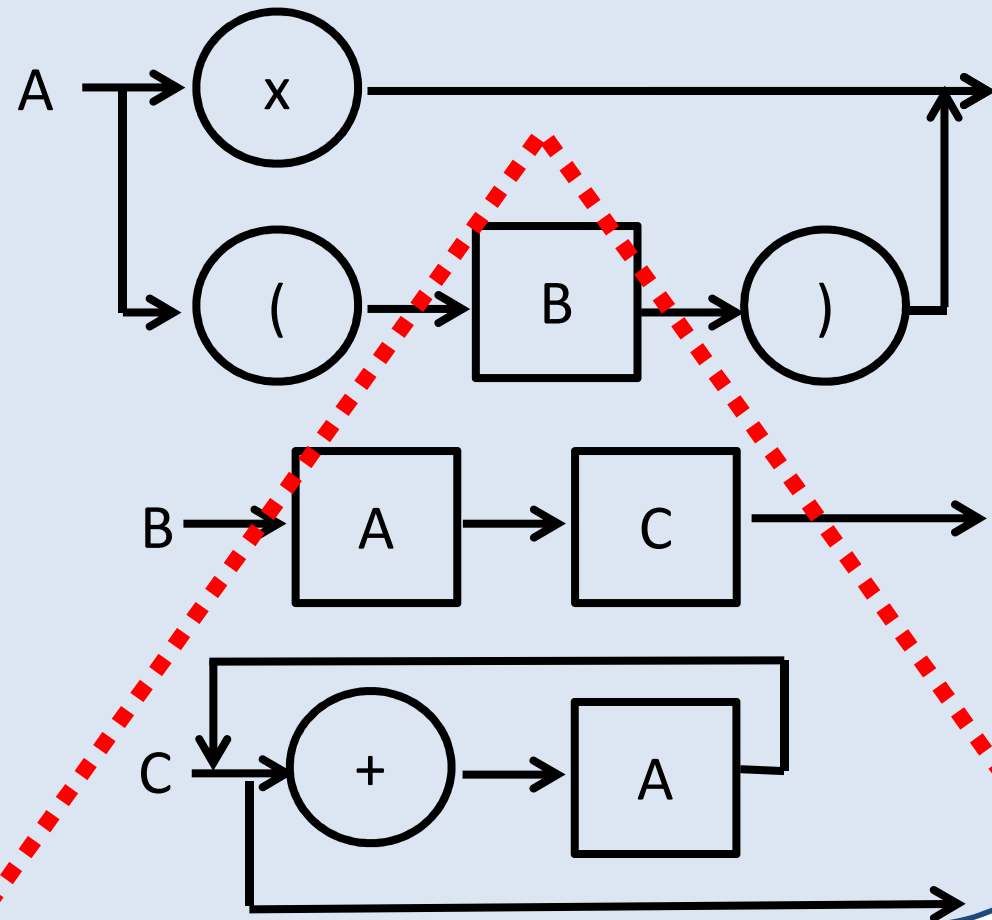
❑ O grafo pode ser simplificado?



ASD Preditivo Recursivo

❑ Obtenha os procedimentos recursivos para os grafos abaixo

❑ O grafo pode ser simplificado?





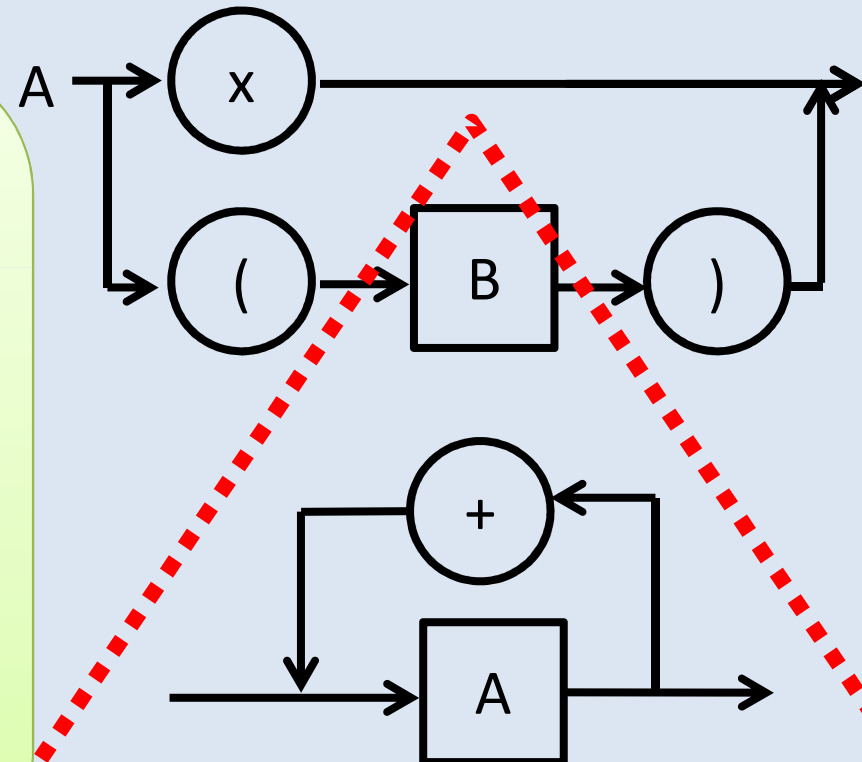
unesp

ASD Preditivo Recursivo

- Obtenha os procedimentos recursivos para os grafos

abaixo

```
procedimento A
begin
  se (simbolo='x') então obter_simbolo
  senão se (simbolo='(') então
    obter_simbolo;
    A;
  enquanto (simbolo='+') faça
    obter_simbolo;
    A;
  se (simbolo=')')
    então obter_simbolo
    senão ERRO;
  senão ERRO;
end
```



ASD Preditivo Recursivo

- ❑ Para utilizar o ASD Preditivo Recursivo é necessário reescrever a gramática:
 - ❑ eliminar a recursividade à esquerda
 - ❑ fatorar: eliminar regras com mesmo terminal em seus conjuntos primeiro
 - ❑ desta forma, a gramática passa ser LL(1)
 - ❑ a partir do próximo símbolo a ser lido, é possível determinar qual regra de produção aplicar.



ASD Preditivo Recursivo

- Operações sobre gramáticas
 - Eliminar a recursividade à esquerda
 - Fatoração



Eliminação da Recursão à Esquerda

- Gramáticas são recursivas à esquerda se possui um *não-terminal* A para o qual existam derivações $A \rightarrow A\alpha$ para uma cadeia α .

Eliminação da Recursão à Esquerda

- ❑ Para o par de produções recursivas à esquerda:
 $A \rightarrow A\alpha \mid \beta$

- ❑ A substituição abaixo elimina a recursão imediata à esquerda:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

- ❑ Nenhuma outra modificação é requerida a partir de A.

Exemplo de Eliminação da Recursão à Esquerda

- Gramática para expressões simples:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id}$$

- Eliminando a recursão à esquerda para E e T:

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \varepsilon$$

$$F \rightarrow (E) \mid \mathbf{id}$$

$$A \rightarrow A\alpha \mid \beta$$

- A substituição abaixo elimina a recursão imediata à esquerda:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$



Recursão não imediata à Esquerda

- ❑ Uma outra gramática:

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

- ❑ Note que tal gramática é recursiva à esquerda de forma não imediata pois:

$$S \Rightarrow Aa \Rightarrow Sda$$

- ❑ É necessário um algoritmo mais eficiente que sistematicamente elimine a recursão à esquerda de uma gramática



Algoritmo de Eliminação de Recursão à Esquerda

Algoritmo:

- ❑ Entrada: uma gramática G recursiva à esquerda.
- ❑ Saída: produção de gramática equivalente sem recursão à esquerda mas que *pode* conter produções- ϵ .
- ❑ Método: aplicação do algoritmo a seguir em G produzindo uma gramática G' equivalente.



Algoritmo de Eliminação de Recursão à Esquerda (ERE)

1. Colocar não terminais em qualquer ordem A_1, A_2, \dots, A_n

2. Para $i=1$ até n faça

 Para $j=1$ até $i-1$ faça

 substituir produções $A_i \rightarrow A_j\beta$

 por $A_i \rightarrow \alpha_1\beta \mid \alpha_2\beta \mid \dots \mid \alpha_n\beta$

 onde $A_j \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ são as produções A_j correntes.

 Elimine a recursão imediata à esquerda

 entre as produções A_j



Exemplo de Aplicação do Algoritmo ERE

- Dada a gramática:

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sd \mid \varepsilon$$

- Ordenamos os não-terminais S , A e efetuamos as substituições indicadas:

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$$

- Eliminando a recursão imediata à esquerda:

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bdA' \mid A'$$

$$A' \rightarrow cA' \mid adA' \mid \varepsilon$$

$$A \rightarrow A\alpha \mid \beta$$

- A substituição abaixo elimina a recursão imediata à esquerda:

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

Fatoração à Esquerda

- ❑ Transformação gramatical apropriada para a criação analisadores sintáticos preditivos.
- ❑ Dada a existência de produções alternativas - consiste na reescrita das produções de forma a adiar a escolha de uma destas até que seja possível a escolha certa - **Ideia: eliminar regras com mesmo terminal em seus conjuntos Primeiro**
- ❑ Desta forma o analisador pode atuar sem a necessidade de efetuar algum retrocesso.

Fatoração à Esquerda

- ❑ Exemplo:

cmd → **if (expr) cmd else cmd**
| **if (expr) cmd** | **a**

expr → **b**

- ❑ Obtendo o *token* if não é possível saber qual das duas produções utilizar. Em geral, se a cadeia começar com α , como começar?

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$

- ❑ Adiar a decisão fatorando A através de sua **expansão** temos:

$A \rightarrow \alpha A' \mid \gamma$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

Fatoração à Esquerda

- Voltando ao exemplo anterior agora compactado:

$$C \rightarrow i(E)CeC \mid i(E)C \mid a$$

$$E \rightarrow b$$

```
<cmd> ::= if (<expr>) <cmd> else <cmd>
        | if (<expr>) <cmd> | a
<expr> ::= b
```

- Temos: $\alpha = i(E)C$, $\beta_1 = eC$, $\beta_2 = \varepsilon$ e $\gamma = a$

- Assim:

$$C \rightarrow i(E)CC' \mid a$$

$$C' \rightarrow eC \mid \varepsilon$$

$$E \rightarrow b$$

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$$

- Adiar a decisão fatorando A através de sua **expansão** temos:

$$A \rightarrow \alpha A' \mid \gamma$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

ASD Preditivo Recursivo

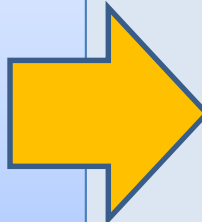
Exercício

□ A gramática é LL(1)? Em caso negativo, transforme-a

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

□ A gramática é LL(1) transforme-a

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



Fatoração à esquerda

□ Em geral, se a cadeia começar com α , como começar?

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$$

□ Adiar a decisão fatorando A através de sua **expansão** temos:

$$A \rightarrow \alpha A' \mid \gamma$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

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



ASD Preditivo Recursivo

Exercício

□ A gramática é LL(1)? Em caso negativo, transforme-a

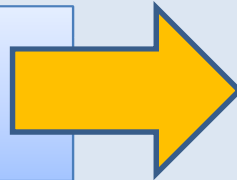
```
<E> ::= <T> + <E> | <T>  
<T> ::= a | b
```

ASD Preditivo Recursivo

Exercício

- A gramática é LL(1)? Em caso negativo, transforme-a

$\langle E \rangle ::= \langle T \rangle + \langle E \rangle \mid \langle T \rangle$
 $\langle T \rangle ::= a \mid b$



$\langle E \rangle ::= \langle T \rangle \langle E' \rangle$
 $\langle E' \rangle ::= + \langle E \rangle \mid \epsilon$
 $\langle T \rangle ::= a \mid b$



ASD Preditivo Recursivo

Exercício

□ Construa os grafos sintáticos

$\langle E \rangle ::= \langle T \rangle \langle E' \rangle$

$\langle E' \rangle ::= + \langle E \rangle \mid \varepsilon$

$\langle T \rangle ::= a \mid b$

ASD Preditivo Recursivo

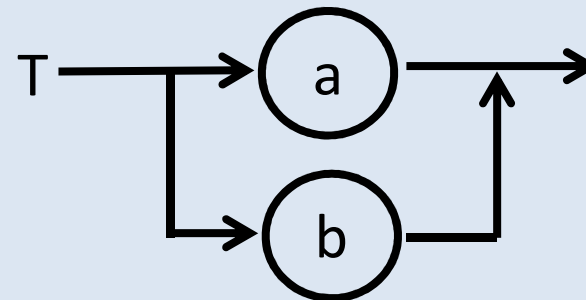
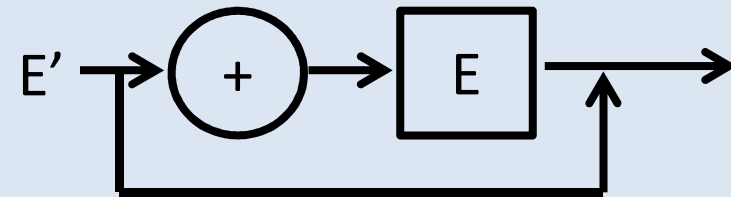
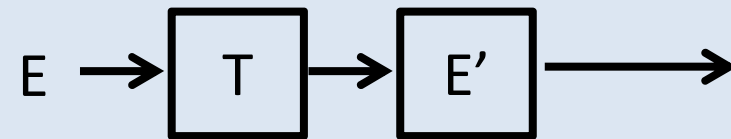
Exercício

□ Construa os grafos sintáticos

$\langle E \rangle ::= \langle T \rangle \langle E' \rangle$

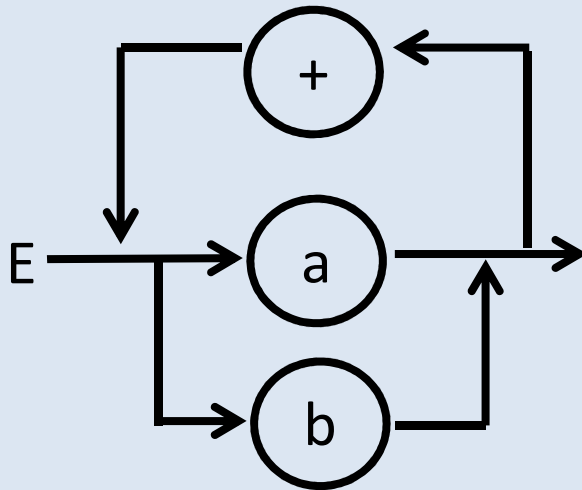
$\langle E' \rangle ::= + \langle E \rangle \mid \varepsilon$

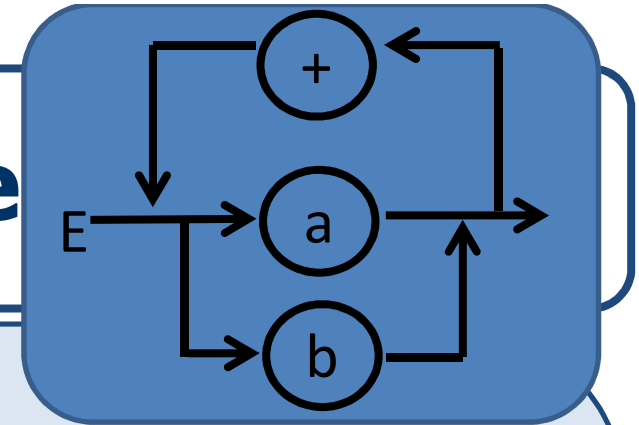
$\langle T \rangle ::= a \mid b$



ASD Preditivo Recursivo

□ Reduzindo o grafo





□ Programa principal e procedimento para E

```
procedimento ASD
begin
  obter_próximo();
  E();
  se (terminou_cadeia) então SUCESSO
  senão ERRO;
end
```

```
procedimento E
begin
  se (símbolo='a') ou (símbolo='b') então
    obter_próximo()
  senão ERRO;
  enquanto (símbolo='+') faça
    obter_próximo();
  se (símbolo='a') ou (símbolo='b') então
    obter_próximo()
  senão ERRO;
end
```



ASD Preditivo Recursivo

Projeto – Parte 2

- Verifique se LALG é LL(1) e, se necessário, transforme-a.
- Construa os grafos sintáticos (em número reduzido) e os procedimentos recursivos para declaração de variáveis em LALG

Na próxima aula

□ ASD Preditivo não-recursivo

