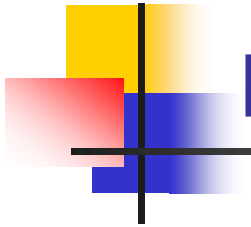
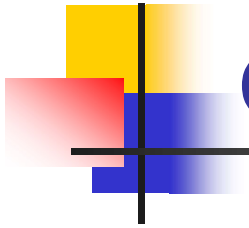
Linguagens Livres de Contexto



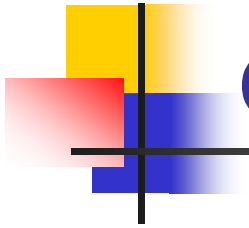
Roteiro

- **Gramáticas livres de contexto**
- Representação de linguagens livres de contexto
- Formas normais para gramáticas livres de contexto
- Gramáticas ambíguas
- Autômatos de Pilha
- Conversão de gramáticas livres de contexto para autômatos de pilha
- Exercícios



Gramáticas livres de contexto (GLC)

- GLC
  - Formalização sintática das linguagens de programação de alto nível;
  - Capacidade de representação de construções aninhadas:
    - construção de expressões aritméticas, em que subexpressões são delimitadas, através do uso de parênteses;
    - na estruturação do fluxo de controle, em que comandos internos são inseridos como parte integrante de outros externos;
    - na estruturação do programa, em que blocos, módulos, procedimentos e funções são empregados para criar diferentes escopos, etc.



GLC - definição

- GLC é uma quádrupla G = (V, T, P, S), onde:
  - V: conjunto (finito e não-vazio) dos símbolos não-terminais;
  - T: conjunto (finito e não-vazio) dos símbolos terminais; corresponde ao alfabeto da linguagem definida pela gramática;
  - P: conjunto (finito e não-vazio) das regras de produção, todas no formato $\alpha \rightarrow \beta$, com $\alpha \in (V - T)$ e $\beta \in (V,T)^*$;
  - S: raiz da gramática, $S \in (V - T)$.

GLC

- Regras de produção P
  - Conjunto P

$$E \rightarrow T + E$$
$$E \rightarrow T$$
$$T \rightarrow F * T$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow a$$

Exemplo de sentença que pode ser reconhecida pela gramática:

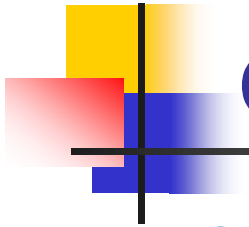
a * (a + a)

GLC

- Regras de produção P
  - Conjunto P

$$E \rightarrow T + E$$
$$E \rightarrow T$$
$$T \rightarrow F * T$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow a$$

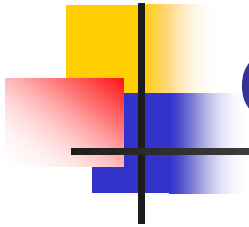
A linguagem gerada pela GLC admite o aninhamento de expressões - operando ***a*** com os operadores "$*$" e "$+$" - através do uso de parênteses como delimitadores.



GLC

- As GLC permitem impor restrições adicionais àquelas que se podem construir em gramáticas regulares, podendo assim caracterizar subconjuntos das linguagens regulares que façam o uso da propriedade determinada pelos aninhamentos sintáticos;

- Dessa forma, as GLC tornam-se muito úteis para a especificação de linguagens de programação, a maioria das quais exige aninhamentos sintáticos.



GLC - aplicação

- Podemos entender o analisador sintático (parser) de um compilador como
  - um dispositivo que tenta determinar se existe uma derivação da sentença de entrada de acordo com alguma GLC.
    - Processo de derivação mais à esquerda ou mais à direita OU árvore de derivação

GLC – derivação mais à esq./dir.

- Dada a gramática

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

- Dada a cadeia: w = -(id) ➔ verifique se w pertence a G.

- Como:

  E $\Rightarrow$- E $\Rightarrow$ - (E) $\Rightarrow$ - (**id**)

- Então: w $\in$ *L(G)*

GLC - derivação mais à esq./dir.

- Uma derivação é construída em dois passos:
  1. Escolha do não-terminal a ser substituído
  2. Escolha da alternativa de substituição
- Dada a cadeia **-(id+id)** e a gramática *G* podemos obter sua derivação:

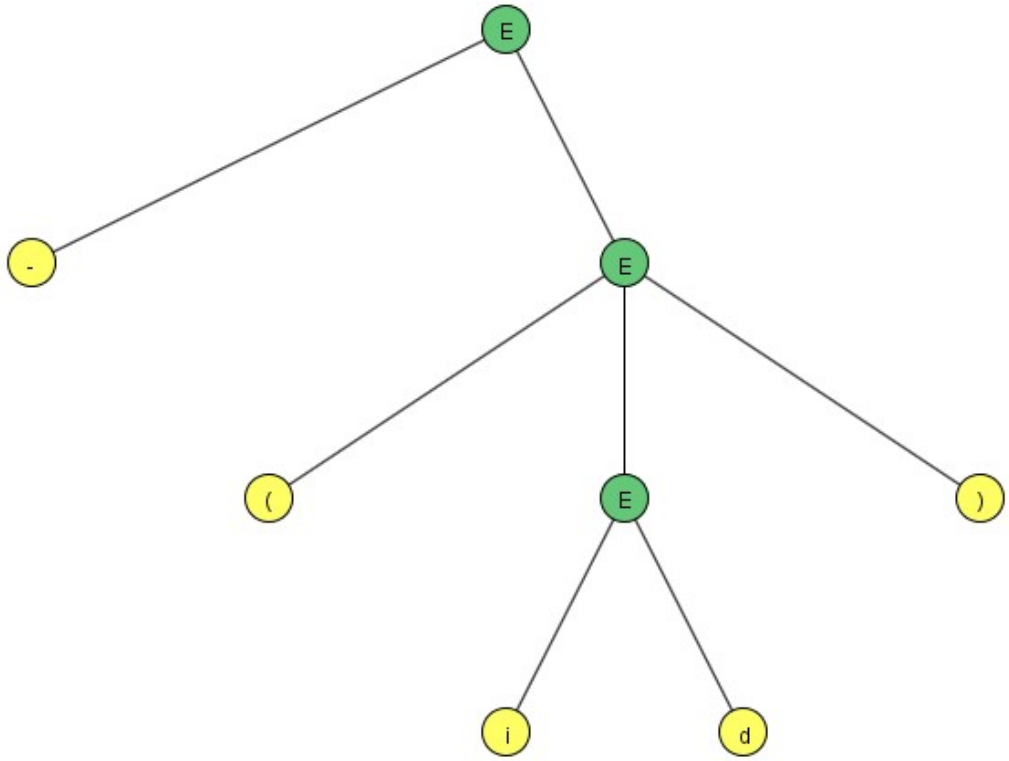
$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E)$$

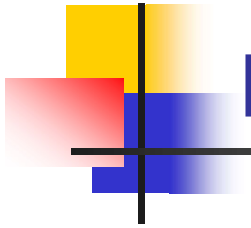
- A partir deste ponto temos duas alternativas:
  - (1) $E \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$ **derivação mais à esq.**
  - (2) $E \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$ **derivação mais à dir.**

GLC – árvore de derivação

- JFlap → *Grammar* → *Brute Force Parser*

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





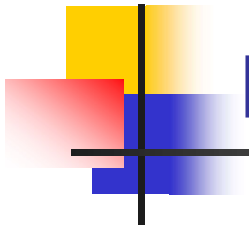
Roteiro

- Gramáticas livres de contexto
- **Representação de linguagens livres de contexto**
- Formas normais para gramáticas livres de contexto
- Gramáticas ambíguas
- Autômatos de Pilha
- Conversão de gramáticas livres de contexto para autômatos de pilha
- Exercícios

Representação de linguagens livres de contexto

- Notação BNF (*Backus-Naur Form*)
  - os não-terminais (V) são representados por textos delimitados pelos metasímbolos "<" e ">";
  - para distingui-los dos símbolos terminais, o metasímbolo "→" é substituído por "::=" e, finalmente, todas as alternativas de substituição para um mesmo não-terminal são agrupadas, separando-se umas das outras com o metasímbolo "|";
  - os terminais são denotados sem delimitadores.
- Exemplo: BNF Java

  http://www.cs.au.dk/~stm/RegAut/JavaBNF.html



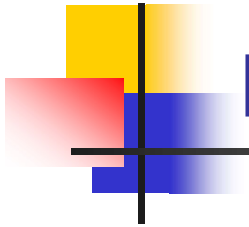
BNF

- Exemplo

GLC

$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow F * T$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow a$

BNF



BNF

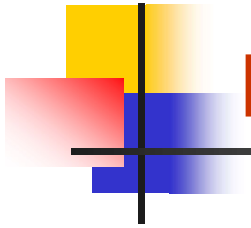
- Exemplo

GLC

$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow F * T$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow a$

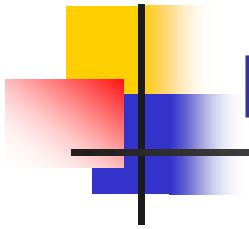
BNF

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



EBNF (BNF estendida)

- Resulta da fusão das definições da BNF e da ER estendida
  - Uma ER estendida admite como operandos os símbolos não-terminais da gramática, em adição aos terminais.
- Um conjunto de regras gramaticais representado através da notação EBNF é um conjunto de ER's estendidas, cada uma delas associada a um símbolo não-terminal distinto.



Exemplo

GLC

P = {S → XYZ | g
X → aX | a
Y → Sb
Z → cdZ | eZ | f }

BNF

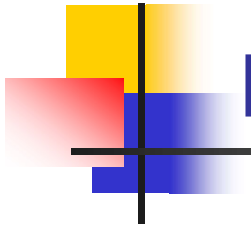
<S> ::= <X><Y><Z> | g
<X> ::= a<X> | a
<Y> ::= <S>b
<Z> ::= cd<Z> | e<Z> | f

BNF estendido

<S> ::= <X><Y><Z> | g
<X> ::= aa*
<Y> ::= <S>b
<Z> ::= (cd | e)*|f

resultando em:

<S> ::= aa*<S>b(cd | e)*|f | g



EBNF

- Vantagem de utilização quando comparada a BNF
  - Possibilidade de representar a repetição de formas sintáticas sem a necessidade de definições gramaticais recursivas (aquelas em que o símbolo não-terminal que estiver sendo definido ressurge, direta ou indiretamente, em formas sentenciais derivadas do mesmo), substituindo-as pela definição de iterações explícitas (através do uso do operador fechamento).
    - Isso proporciona um entendimento mais fácil da linguagem por ela definida, sendo ainda útil em determinados métodos de construção de reconhecedores sintáticos a partir de GLC

Extensões incorporadas na EBNF

- O uso dos metasímbolos "[" e "]" para agrupar termos opcionais, dispensando assim o uso do símbolo ε para representar a cadeia vazia.

- Exemplo:

$$S \rightarrow aS \mid \varepsilon$$

pode ser reescrito como

<S> ::= [a<S>]

Extensões incorporadas na EBNF

- O uso dos metasímbolos "{" e "}" para agrupar termos que se repetem zero ou mais vezes, dispensando assim o uso de recursões explícitas e também o uso de parênteses no agrupamento do termo que se repete

- Exemplo:

  S → aS | b é substituído por

  S ::= {a}b

Extensões incorporadas na EBNF

- O uso do metasímbolo "+" para representar a repetição de um termo uma ou mais vezes.

- Exemplo

  $S \rightarrow aS \mid a$, cuja solução é

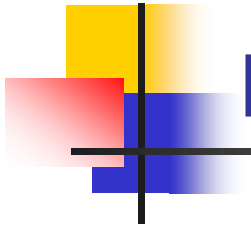
  $S ::= a^{+}$

Extensões incorporadas na EBNF

- O emprego de aspas duplas para delimitar os símbolos terminais da gramática (às vezes também o negrito ou o sublinhado), mantendo-se os símbolos não-terminais sem qualquer tipo de destaque nas regras gramaticais.

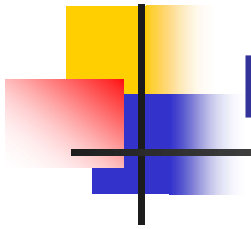
- Exemplo:

S ::= "a"S, S ::= **a**S, ou ainda S ::= a̲S para denotar S → aS



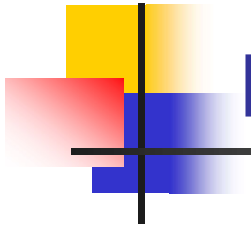
Roteiro

- Gramáticas livres de contexto
- Representação de linguagens livres de contexto
- **Formas normais para gramáticas livres de contexto**
- Gramáticas ambíguas
- Autômatos de Pilha
- Conversão de gramáticas livres de contexto para autômatos de pilha
- Exercícios



Formas normais para GLC

- Forma normal mais utilizada → Forma Normal de *Chomsky*



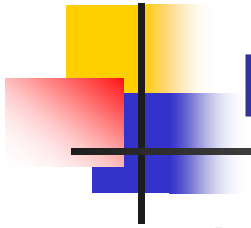
Forma Normal de *Chomsky*

- Uma gramática G = (V, T, P, S) obedece à Forma Normal de *Chomsky* se todas as produções p ∈ P forem de uma das duas formas seguintes:

  1. $A \rightarrow BC$, ou
  2. $A \rightarrow a$

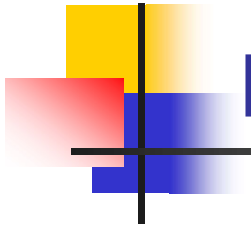
  com $A, B, C \in V$ e $a \in \Sigma^*$.

- Se $\varepsilon \in L(G)$, então admite-se $S \rightarrow \varepsilon$ como única produção em que ε aparece do lado direito.



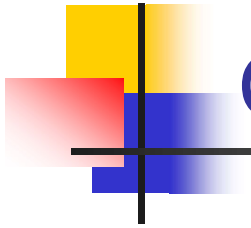
Forma Normal de *Chomsky*

- Toda linguagem livre de contexto L pode ser gerada por uma gramática livre de contexto na Forma Normal de Chomsky.



Roteiro

- Gramáticas livres de contexto
- Representação de linguagens livres de contexto
- Formas normais para gramáticas livres de contexto
- **Gramáticas ambíguas**
- Autômatos de Pilha
- Conversão de gramáticas livres de contexto para autômatos de pilha
- Exercícios



Gramáticas ambíguas

- Um requisito importante de uma linguagem de programação é que ela não seja ambígua ou, no mínimo, que qualquer ambiguidade na linguagem seja evidente ou facilmente evitada.

Gramáticas ambíguas

G: E → E + E | E * E | (E) | - E | **id**

- Uma gramática é dita ambígua se para uma dada sentença podem existir duas ou mais derivações (mais à esq ou mais à dir.) OU árvores distintas
- Considerando a gramática G e a sentença **id+id*id** podemos ter:

E ⇒ E+E ⇒ id+E ⇒ id+E*E ⇒ id+id*E ⇒ id+id*id

ou

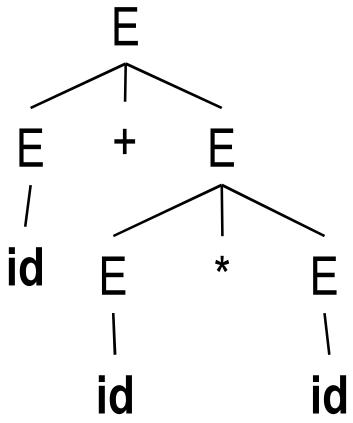
E ⇒ E*E ⇒ E+E*E ⇒ id+E*E ⇒ id+id*E ⇒ id+id*id

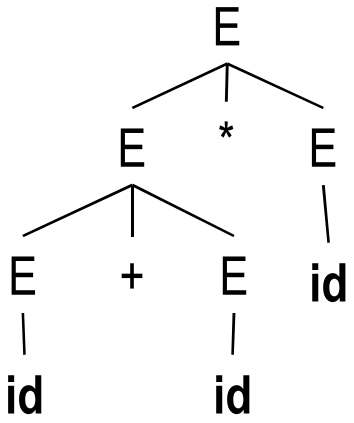
- Para cada uma destas derivações é possível a construção de uma árvore sintática distinta.

Gramáticas ambíguas

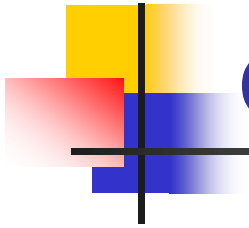
- Árvores de derivação mostrando a ambiguidade

sentença **id+id*id**





- Uma forma conveniente de eliminarmos a ambiguidade de uma gramática é reescrevendo tal gramática



Gramáticas ambíguas – exemplo clássico

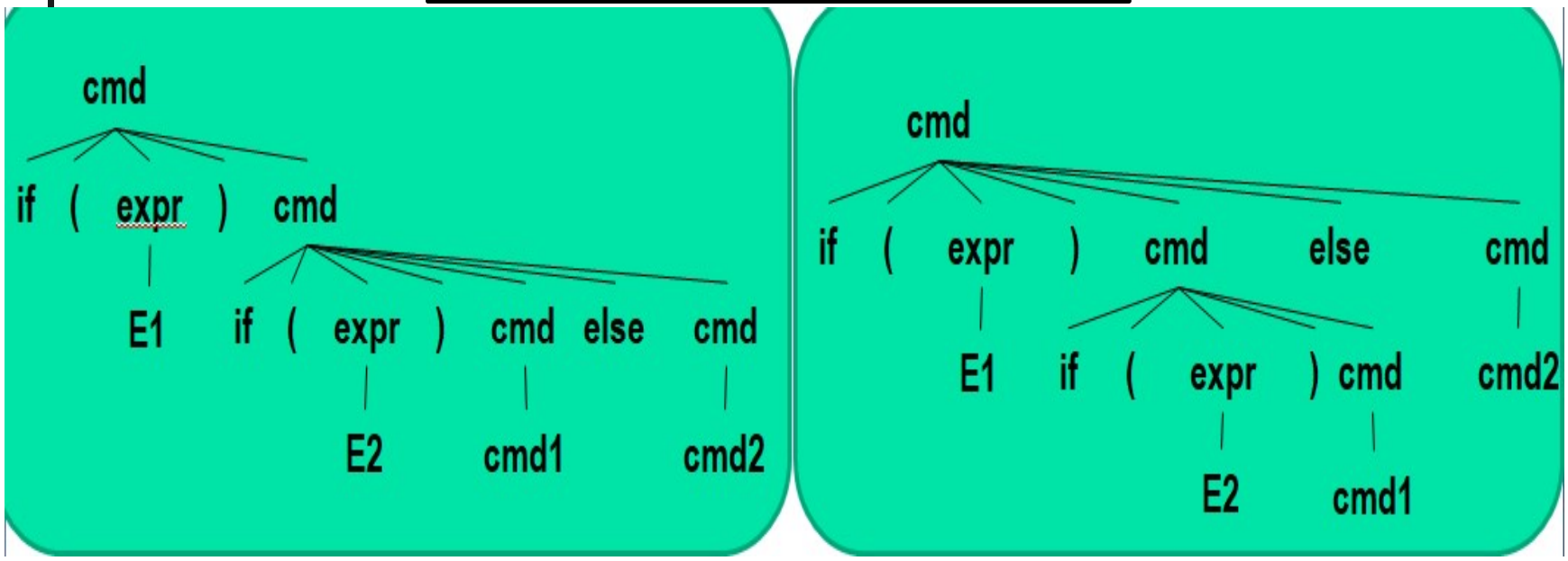
- Um problema clássico é o *"else-vazio"*. Considere a gramática abaixo:

  cmd → if (expr) cmd | if (expr) cmd else cmd
  | outro

- Dada a sentença **if** (E_1) **if** (E_2) cmd_1 **else** cmd_2 é possível obter duas árvores de derivação

Gramáticas ambíguas – exemplo clássico

if (E_1) **if** (E_2) cmd_1 **else** cmd_2



- Ou seja, o *else* "pertence" ao primeiro ou ao segundo *if*?
- Embora todas as linguagens de programação prefiram a primeira árvore, a gramática dada é incapaz de determinar isto.

Reescrevendo a gramática para eliminar a ambiguidade

if (E_1) if (E_2) cmd_1 **else** cmd_2

cmd → cmdIsol | cmdAssoc

cmdAssoc → if (expr) cmdAssoc else cmdAssoc | outro

cmdIsol → if (expr) cmd | if (expr) cmdAssoc else cmdIsol

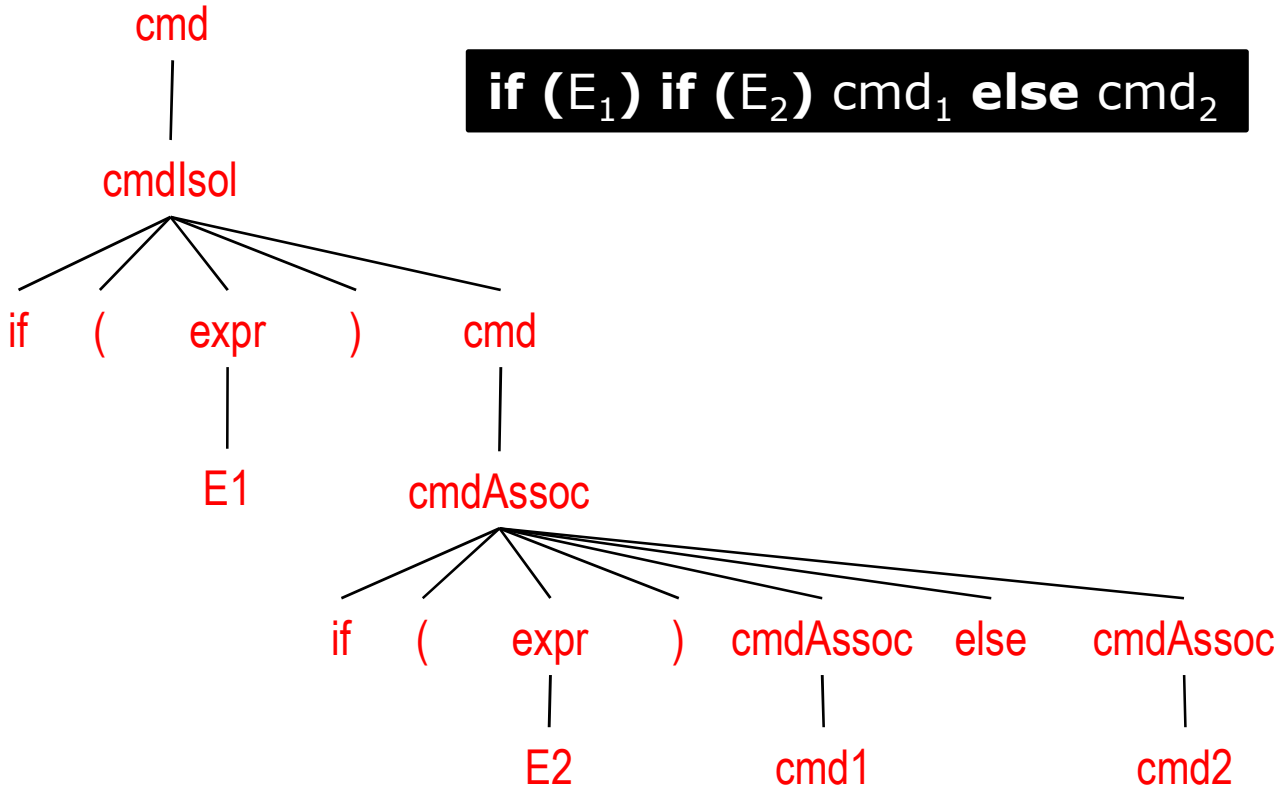
- Esta gramática resolve o problema do "*else-vazio*" pois:
  - Todo comando deve ser inicialmente tratado como um comando isolado
  - Um comando figurando entre um *if* e um *else* deve ser tratado como comando associado
  - Comandos associados ou são *if-else* ou são outros comandos
- Devemos notar que neste caso existem "regras" de uso associada a gramática

Reescrevendo a gramática para eliminar a ambiguidade

cmd → cmdIsol | cmdAssoc

cmdAssoc → if (expr) cmdAssoc else cmdAssoc | outro

cmdIsol → if (expr) cmd | if (expr) cmdAssoc else cmdIsol



if (E_1) if (E_2) cmd_1 else cmd_2

Gramáticas ambíguas – exemplo 2

- Seja G1 = ({E,T,F,U},(0,1,2,...,9},P,E)

  P = { E -> E + T | T
  T -> T * F | F
  F -> U | (E)
  U -> 0 | 1 | ... | 9}

- uma gramática para expressões aritméticas para inteiros de 0..9.

- Por que as LP não usam a gramática G2:

  P = { E -> E + E | E * E | (E) | F
  F -> 0 | 1 | ...| 9}

- que contém um menor número de símbolos e produções?

Gramáticas ambíguas – exemplo 2

- A razão é que:
  1. G2 é ambígua e G1 não é.
  2. Em G2 **+** e ***** tem a mesma prioridade de resolução enquanto que na matemática eles não tem.

- Dada a expressão 2 + 3 * 5
- Ela é interpretada como?

  (2 + (3 * 5)) = 17
- Ou

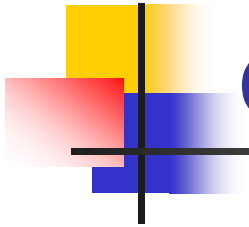
  ((2 + 3) * 5) = 25

G1

P = { E -> E + T | T
T -> T * F | F
F -> U | (E)
U -> 0 | 1 | ... | 9}

G2

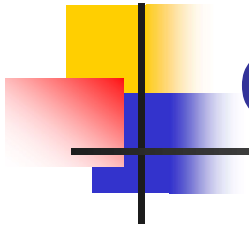
P = { E -> E + E | E * E | (E) | F
F -> 0 | 1 | ...| 9}



Gramáticas ambíguas

- Causas da ambiguidade

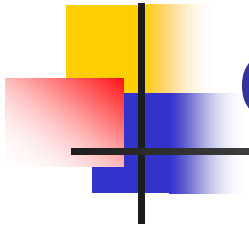
- A prioridade não é respeitada
  - Precisamos forçar somente uma forma de estruturar o uso de vários operadores na árvore de derivação
- Uma sequência de operadores idênticos podem se agrupar a partir da esquerda ou da direita
  - Precisamos forçar somente uma forma de se associar os operadores idênticos



Gramáticas ambíguas

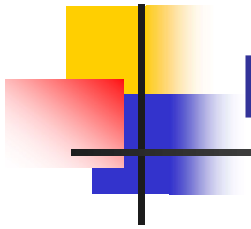
- Para mostrar que uma gramática é ambígua é suficiente:
  - Para uma dada sentença:
    - existir duas ou mais derivações (mais à esq ou mais à dir.) para representá-la

      OU
    - mais de uma árvore de derivação distinta



Gramáticas ambíguas

- Para retirar a ambiguidade:
  - introduzir várias variáveis e criar novas regras, quando temos operadores com várias prioridades de resolução.
  - Para resolver a ambiguidade vinda do uso de vários operadores idênticos, forçamos o uso da recursão para esquerda ou direita



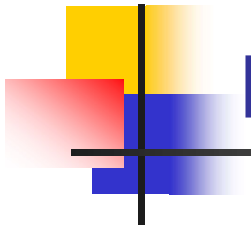
Exercício (1)

Verifique (a partir da sentença aabbab) se a gramática abaixo é ambígua. Em caso afirmativo, retire a ambiguidade

S -> bA | aB

A -> a | aS | bAA

B -> b | bS | aBB

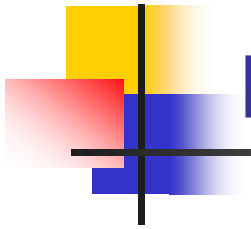


Exercício (2)

- Dada a gramática que define os operadores lógicos and e or

  E -> E or E | E and E | (E) | a

- Verifique a partir de árvores de derivação e com a sentença a or a and a se a gramática é ambígua. Em caso afirmativo, retire a ambiguidade



Exercício (3)

Verifique (a partir da sentença **( ) ( ) ( )**) se a gramática abaixo é ambígua.
Em caso afirmativo, retire a ambiguidade

S -> (S) | ( ) | SS

Exercício

4. Mostre a linguagem gerada pela gramática livre de contexto abaixo .

G = ({S,A,B},{a,b}, P, S), cujo P ={S--> AB, A --> aA | aBB, B --> Bb | b}

5. Seja G = ({<cad>, <meio>}, {a,b}, P, <cad>) uma gramática com as seguintes regras de produção:

<cad> --> ab | a <meio>b<meio> --> a<meio> | <meio>b | a | b

Pergunta-se: Trata-se de uma gramática ambígua ou não? Justifique. Qual a linguagem gerada por G?

6. A gramática G = ({E}, {a,b,b,(,),OR,AND}, P, E) cujo P = {E --> a | b | c | E OR E | E AND E | (E) } é ambígua? Justifique. Se for, encontre uma gramática não ambígua para ela.

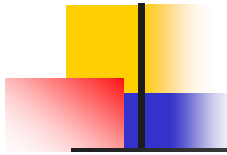
7. A gramática a seguir gera a linguagem 0*1(0+1)*:

S →A1B

A →0A | ε

B →0B | 1B | ε

- Forneça derivações mais à esquerda, mais à direita e árvore de análise sintática para as cadeias: (a) 00101 (b) 1001 (c) 00011



Exercício

8. Dado o comando de repetição **for** estilo Pascal. Considere que o corpo pode ser composto apenas por atribuições e/ou outros comandos **for**, e o início e fim podem ser expressões aritméticas quaisquer. Dois exemplos:

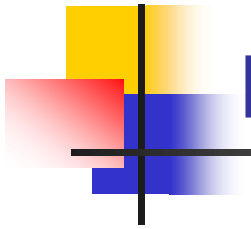
- 1ª sentença:

```
for x := 0 to 50 do
begin
    x := 3;
    Valor := Valor + 1;
end
```

- 2ª sentença:

```
for x := 10 downto 5 do
    for y := 1 to 10 do
      w := w + x + y;
```

Construa a GLC utilizando a notação BNF



Exercício (1) - resolução

Verifique (a partir da sentença aabbab) se a gramática abaixo é ambígua. Em caso afirmativo, retire a ambiguidade

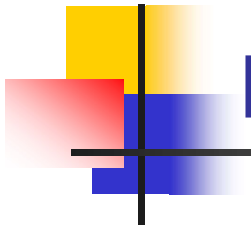
S -> bA | aB

A -> a | aS | bAA

B -> b | bS | aBB

- S => aB => aaBB => aabB => aabbS => aabbaB => aabbab
- S => aB => aaBB =>aabSB => aabbAB => aabbaB => aabbab

- L(G) = {w | w consiste de um número igual de a's e b's}. A palavra aabbab tem 2 derivações mais à esquerda, sinalizando a ambiguidade da gramática.



Exercício (1) - resolução

Verifique (a partir da sentença aabbab) se a gramática abaixo é ambígua. Em caso afirmativo, retire a ambiguidade

S -> bA | aB

A -> a | aS | bAA

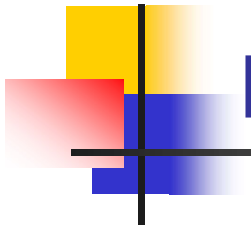
B -> b | bS | aBB

- G não ambígua que gera a mesma linguagem:

S -> aBS | aB | bAS | bA

A -> bAA | a

B -> aBB | b



Exercício (2) - resolução

- Dada a gramática que define os operadores lógicos and e or

  E -> E or E | E and E | (E) | a

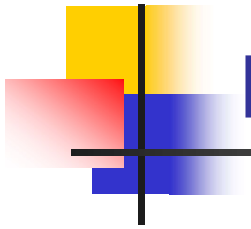
- Verifique a partir de árvores de derivação e com a sentença a or a and a se a gramática é ambígua. Em caso afirmativo, retire a ambiguidade

- Gramática não ambígua:

  E -> E or T | T

  T -> T and F | F

  F -> a | (E)



Exercício (3) - resolução

Verifique (a partir da sentença **( ) ( ) ( )**) se a gramática abaixo é ambígua. Em caso afirmativo, retire a ambiguidade

S -> (S) | ( ) | SS

Gramática não ambígua

S -> (S) | ( ) | SA
A -> ( ) | (S)