

# Propriedades de LLCs

Transformações úteis para *GLC*:

(1) eliminação de produções nulas;

(2) eliminação de produções unitárias;

(3) Eliminação de símbolos estéreis e inacessíveis;

Forma Normal de Chomsky (FNC) e Forma Normal de Greibach (FNG)

(4) Lema do Bombeamento para LLCs

(5) Propriedades das LLCs

# Formas normais para GLC

- É mais fácil provar certas propriedades das LLC se simplificarmos as GLC, evidentemente sem alterar as linguagens geradas por elas.
- Para mostrar que toda LLC (sem  $\lambda$ ) pode ser gerada por uma GLC cujas produções têm a forma (**Forma Normal de Chomsky**):

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a \quad \text{onde } A, B, C \in V_n \text{ e } a \in V_t$$

Temos que:

1. Eliminar símbolos inúteis
2. Eliminar as  $\lambda$ -produções
3. Eliminar as produções unitárias

# 1. Eliminação de Símbolos Inúteis

- **Def:** um símbolo  $X$  de  $V_n$  ou  $V_t$  é **útil** para uma gramática  $G = (V_n, V_t, P, S)$  se existe alguma derivação da forma  $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$  tal que  $w \in V_t^*$ ;  $\alpha$  e  $\beta \in (V_n \cup V_t)^*$ . Ou seja, se  $X$  faz parte de alguma forma sentencial de  $G$ . Se  $X$  não é útil, chamamos de **inútil**.
- Omitir símbolos inúteis não altera a linguagem gerada.
- Para eliminá-los vamos usar duas características de símbolos úteis:
  - capacidade de gerar sentenças (**fértil ou gerador**) e
  - ser **acessível** a partir do símbolo inicial.

- 1)  $X$  é fértil se  $X \Rightarrow^* w$  para alguma cadeia de terminais  $w$ . Todo terminal é fértil pois  $w \Rightarrow^* w$ .
  
- 2)  $X$  é acessível se existe uma derivação  $S \Rightarrow^* \alpha X \beta$  para algum  $\alpha$  e  $\beta$ . Isto é, acessível a partir de  $S$ .

Se eliminarmos

**PRIMEIRO** os símbolos não férteis (estéreis)  
e

**DEPOIS** os não-acessíveis (inacessíveis),  
deixaremos a gramática somente com símbolos úteis.

# Exemplo

$$G = (\{S, A, B\}, \{a, b\}, P, S)$$

$$P = \{S \rightarrow AB \mid a$$

$$A \rightarrow b\}$$

Qual é a  $L(G)$  ?

$$G = (\{S, A, B\}, \{a, b\}, P, S)$$

$$P = \{S \rightarrow AB \mid a$$

$$A \rightarrow b\}$$

Todos os símbolos, menos B, são férteis. Para eliminar B devemos eliminar a produção  $S \rightarrow AB$ . A gramática fica:

$$S \rightarrow a$$

$$A \rightarrow b$$

- S e a são acessíveis a partir de S.
- Eliminando A e b ficamos com  $S \rightarrow a$
- e a linguagem gerada é  $\{a\}$  justamente igual a  $L(G)$  inicial.
  
- Se iniciássemos com a checagem dos inacessíveis primeiro, não encontraríamos nenhum, e terminaríamos com a gramática:

$$S \rightarrow a$$

$$A \rightarrow b$$

que ainda possui A e b como inúteis.

# Algoritmo para exclusão de símbolos inúteis

O algoritmo faz a exclusão em 2 etapas:

Considere  $G = (V_n, V_t, P, S)$

- 1) Resulta em uma  $G_1 = (V_{n1}, V_t, P_1, S)$  somente com símbolos férteis.

$$V_{n1} = \emptyset$$

Repita

$$V_{n1} = V_{n1} \cup \{A \mid A \rightarrow \alpha \in P \text{ e } \alpha \in \{V_t \cup V_{n1}\}^*\}$$

Até que o nro de elementos de  $V_{n1}$  não aumente

$$P_1 = P - \{Y \rightarrow \alpha X \beta \mid X \text{ não está em } V_{n1}\}$$

# Algoritmo para exclusão de símbolos inúteis

O algoritmo faz a exclusão em 2 etapas:

Considere  $G = (V_n, V_t, P, S)$

1) Resulta em uma  $G_1 = (V_{n1}, V_t, P_1, S)$  somente com símbolos férteis.

$$V_{n1} = \emptyset$$

Repita

$$V_{n1} = V_{n1} \cup \{A \mid A \rightarrow \alpha \in P \text{ e } \alpha \in \{V_t \cup V_{n1}\}^*\}$$

Até que o nro de elementos de  $V_{n1}$  não aumente

$$P_1 = P - \{Y \rightarrow \alpha X \beta \mid X \text{ não está em } V_{n1}\}$$

$$V_{n1} = \{S, A\} \quad P_1 = \{S \rightarrow a; A \rightarrow b\}$$

$$G = (\{S, A, B\}, \{a, b\}, P, S) \\ P = \{S \rightarrow AB \mid a \\ A \rightarrow b\}$$



2) Resulta em uma  $G_2 = (V_{n2}, V_{t2}, P_2, S)$  somente com símbolos acessíveis a partir de  $S$

$$V_{t2} = \emptyset$$

$$V_{n2} = \{S\}$$

$$V_{n1} = \{S, A\} \quad P_1 = \{S \rightarrow a; A \rightarrow b\}$$

Repita

$$V_{n2} = V_{n2} \cup \{A \mid X \rightarrow \alpha A \beta \in P_1, X \in V_{n2}\}$$

$$V_{t2} = V_{t2} \cup \{a \mid X \rightarrow \alpha a \beta \in P_1, X \in V_{n2}\}$$

Até que o nro de elementos de  $V_{n2}$  e  $V_{t2}$  não aumentem

$$P_2 = P_1 - \{Y \rightarrow \alpha \mid Y \text{ não está em } V_{n2}\}$$

$$V_{n2} = \{S\} \quad V_{t2} = \{a\} \quad P_1 = \{S \rightarrow a\}$$

# Exemplo

$$G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$$

$$P = \{ S \rightarrow aAa \mid bBb$$

$$A \rightarrow a \mid S$$

$$C \rightarrow c\}$$

$$L(G) = (aaa)^+$$

1) Iteração

Variáveis

início

$\emptyset$

1

$\{A, C\}$   $A \rightarrow a$  e  $C \rightarrow c$

2

$\{A, C, S\}$   $S \rightarrow aAa$

3

$\{A, C, S\}$

A produção  $S \rightarrow bBb$  foi excluída de  $G_1$ .

| 2) Iteração | Variáveis | Terminais   |
|-------------|-----------|-------------|
| início      | {S}       | $\emptyset$ |
| 1           | {S,A}     | {a}         |
| 2           | {S,A}     | {a}         |

A produção  $C \rightarrow c$  foi excluída pois  $C$  e  $c$  não pertencem à nova gramática  $G_2$ .

Gramática final:

$$G = (\{S,A\}, \{a\}, P, S)$$

$$P = \{ S \rightarrow aAa$$

$$A \rightarrow a \mid S \}$$

$$L(G) = (aaa)^+$$

## 2. Eliminação de Produções Nulas

- Produções nulas não são essenciais para as gramáticas, a não ser para gerar a cadeia nula quando esta pertence à linguagem.
- Se  $L$  tem uma  $GLC$ , então  $L - \{\lambda\}$  tem uma  $GLC$  sem  $\lambda$ -produções.
- A idéia é primeiro descobrir todas as variáveis anuláveis, ou seja, todo  $A$  de  $V_n$  tal que  $A \Rightarrow^* \lambda$ .
- Se  $A$  é anulável, sempre que  $A$  aparecer do lado de direito de uma regra, digamos  $B \rightarrow CAD$ ,  $A$  poderá (ou não) derivar  $\lambda$ .
- Criamos 2 versões da produção:
  - $B \rightarrow CD$  (correspondente ao caso em que  $A$  teria sido usada para derivar  $\lambda$ ), e
  - $B \rightarrow CAD$
- E eliminamos todas as produções com lado direito  $\lambda$ . 12

Ex.:  $G$ :

$S \rightarrow AB$

$A \rightarrow aAA \mid \lambda$

$B \rightarrow bBB \mid \lambda$

(1) Símbolos Anuláveis:  $S, A, B$

(2)  $S \rightarrow AB \quad \Rightarrow \quad S \rightarrow AB \mid A \mid B$

$A \rightarrow aAA \mid \lambda \quad \Rightarrow \quad A \rightarrow aAA \mid aA \mid a$

$B \rightarrow bBB \mid \lambda \quad \Rightarrow \quad B \rightarrow bBB \mid bB \mid b$

(3) Como  $S \Rightarrow^* \lambda \Rightarrow S \rightarrow \lambda$  é acrescentada



$G_1$  equivalente a  $G$

### 3. Eliminação de Produções Unitárias ( $A \rightarrow B$ )

A produção  $A \rightarrow B$  não adiciona informação nenhuma e, se  $B \rightarrow \alpha$  então podemos trocar  $A \rightarrow B$  por  $A \rightarrow \alpha$ .

O algoritmo faz a exclusão em 2 etapas.

Seja  $G=(V_n, V_t, P, S)$ :

1) Construção do fecho de cada variável

Para toda  $A \in V_n$ ,

$\text{Fecho-}A = \{B \in V_n \mid A \leftrightarrow B \text{ e } A \Rightarrow^+ B \text{ usando só produções unitárias } X \rightarrow Y\}$

Ex:  $E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a$

$\text{Fecho-}E = \{T, F\}$

$\text{Fecho-}T = \{F\}$

$\text{Fecho-}F = \emptyset$

2) Exclusão das produções da forma  $A \rightarrow B$ . A gramática resultante é  $G1 = (Vn, Vt, P1, S)$ .

Seja  $P1 = P - \{\text{produções unitárias}\}$

e

Para todo  $A \in Vn$  e  $B \in \text{Fecho-}A$

faça Se  $B \rightarrow \alpha \in P$  e  $\alpha \notin Vn$

então  $P1 = P1 \cup \{A \rightarrow \alpha\}$

No exemplo:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid a$

$\text{Fecho-}E = \{T, F\}$

$\text{Fecho-}T = \{F\}$

$P1: E \rightarrow E + T$

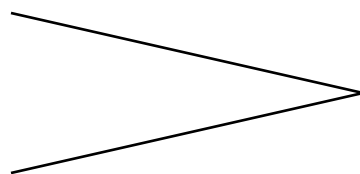
$T \rightarrow T * F$

$F \rightarrow (E) \mid a$

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

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

$F \rightarrow (E) \mid a$



$G1$  equivalente a  $G$

# Ordem segura de aplicação das simplificações

- (1) eliminação de produções nulas;
- (2) eliminação de produções unitárias;
- (3) Eliminação de símbolos estéreis e inacessíveis, isto é, de símbolos inúteis.



# Forma Normal de Chomsky

- Toda LLC não vazia, sem  $\lambda$ , tem uma gramática  $G$  em que todas as produções são de uma das 2 formas:
- $A \rightarrow BC$  com  $A, B, C \in V_n$ , ou
- $A \rightarrow a$  com  $a \in V_t$

Além disso,  $G$  não tem nenhum símbolo inútil.

Dizemos que  $G$  está na *Forma Normal de*

*Chomsky (FNC)* Noan Chomsky: Lingüista que propôs a descrição de línguas naturais com *GLC*, e que provou que toda *GLC* poderia ser convertida na *FNC*.

# Conversão de uma GLC em FNC

- Eliminar símbolos inúteis, produções unitárias e  $\lambda$ -produções; (o resultado são produções do tipo  $A \rightarrow a$  ou terá um corpo de comprimento 2 ou mais)
- Organizar todos os corpos de comprimento 2 ou mais para que consistam apenas em variáveis; (a)
- Desmembrar os corpos de comprimento 3 ou mais em uma cascata de produções, cada uma com um corpo consistindo em duas variáveis. (b)

(a) Para todo terminal  $a$  que aparecer em um corpo de comprimento 2 ou mais, crie uma nova variável  $A$ . Essa variável terá apenas a produção  $A \rightarrow a$ . Usamos  $A$  em lugar de  $a$  em todo lugar em que  $a$  aparecer em um corpo de comprimento 2 ou mais. Assim, toda produção terá um corpo que será um único terminal ou pelo menos 2 variáveis e nenhum terminal.

(b) Desmembramos as produções

$A \rightarrow B_1 B_2 \dots B_k$ , para  $k > 2$ , em um grupo de produções com 2 variáveis em cada corpo. Introduzimos  $k-2$  novas variáveis,  $C_1, C_2, \dots, C_{k-2}$ , em  $k-1$  produções:

$A \rightarrow B_1 C_1, \quad C_1 \rightarrow B_2 C_2, \dots, \quad C_{k-2} \rightarrow B_{k-1} B_k$

# Exemplo

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

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

$F \rightarrow (E) \mid a$

(a):

$E \rightarrow E M T \mid T V F \mid P E Q \mid a$

$T \rightarrow T V F \mid P E Q \mid a$

$F \rightarrow P E Q \mid a$

$P \rightarrow ($

$Q \rightarrow )$

$M \rightarrow +$

$V \rightarrow *$

$E \rightarrow EMT \mid TVF \mid PEQ \mid a$

$T \rightarrow TVF \mid PEQ \mid a$

$F \rightarrow PEQ \mid a$

$P \rightarrow ($

$Q \rightarrow )$

$M \rightarrow +$

$V \rightarrow *$

(b):

$E \rightarrow EX \mid TY \mid PZ \mid a$

$X \rightarrow MT$

$Y \rightarrow VF$

$Z \rightarrow EQ$

$T \rightarrow TY \mid PZ \mid a$

$F \rightarrow PZ \mid a$

$P \rightarrow ($

$Q \rightarrow )$

$M \rightarrow +$

$V \rightarrow *$

# Forma Normal de Greibach

- Toda LLC não vazia e sem  $\lambda$  tem uma gramática  $G$  em que todas as produções são da forma:
- $A \rightarrow a\alpha$  onde  $a \in V_t$  e  $\alpha$  é uma cadeia de zero ou mais variáveis.

Algoritmo de transformação em (Menezes, 2002).

Desde que cada uso de uma produção introduz exatamente 1 terminal numa forma sentencial, uma sentença de tamanho  $n$  tem uma derivação de exatamente  $n$  passos.

# Lema do Bombeamento para LLC

De forma análoga ao LB para LR, esse lema permite mostrar que uma linguagem não é LLC.

Idéia: Em qualquer cadeia suficientemente longa de uma LLC, é possível encontrar no máximo duas subcadeias curtas e próximas, que podemos bombear em conjunto. Isto é, podemos repetir ambas as cadeias  $i$  vezes, para qualquer inteiro  $i$ , e a cadeia resultante ainda estará na linguagem.



Teorema (Lema do Bombeamento para LLC): Seja  $L$  uma LLC. Então, existe uma constante  $n$  tal que, para toda cadeia  $z$  de  $L$ , com  $|z| \geq n$ , podemos escrever

$z = uvwxy$ , sujeita às seguintes condições:

1.  $|vwx| \leq n$ . Ou seja, a porção intermediária não é muito longa.
2.  $vx \neq \varepsilon$ . Tendo em vista que  $v$  e  $x$  são os fragmentos a serem bombeados, essa condição diz que pelo menos uma das cadeias que bombeamos não deve ser vazia.
3. Para todo  $i \geq 0$ ,  $uv^iwx^iy$  está em  $L$ . Isto é, as duas cadeias  $v$  e  $x$  podem ser bombeadas qualquer número de vezes, incluindo 0, e a cadeia resultante ainda será um elemento de  $L$ .

- Seja  $L = \{0^n 1^n 2^n\}$ . Suponha  $L$  livre de contexto. Então, existe um inteiro  $n$  dado pelo LB. Vamos escolher  $z = 0^n 1^n 2^n$ . Podemos desmembrar  $z$  como  $z = uvwxy$ , onde  $|vwx| \leq n$  e onde  $v$  e  $x$  não são ambos  $\lambda$ . Assim, sabemos que  $vwx$  não pode envolver ao mesmo tempo  $0s$  e  $2s$ , pois o último  $0$  e o primeiro  $2$  estão separados por  $n+1$  posições. Provaremos que  $L$  contém alguma cadeia que reconhecidamente não está em  $L$ , contradizendo a hipótese.

Os casos possíveis são:

1.  $vwx$  não tem nenhum 2. Então  $vx$  consiste apenas de 0s e 1s, e tem pelo menos um desses símbolos. Portanto,  $uwy$ , que deveria estar em  $L$  pelo LB, tem  $n$  2s, mas tem menos de  $n$  0s ou menos de  $n$  1s, ou ambos. Assim, ele não pertence a  $L$ , e concluímos que  $L$  não é uma LLC nesse caso.
2.  $vwx$  não tem nenhum 0. De modo semelhante,  $uwy$  tem  $n$  0s, mas tem um número menor de 1s ou 2s. Portanto, ele não está em  $L$ .

Qualquer que seja o caso, concluímos que  $L$  tem uma cadeia que sabemos que não está em  $L$ . Essa contradição nos permite concluir que nossa hipótese estava errada;  $L$  não é uma LLC.

- Verifique na bibliografia as provas, usando o LB, de que as seguintes linguagens não são LLC:

- $L = \{0^i 1^j 2^i 3^j \mid i \geq 1 \text{ e } j \geq 1\}$

- $L = \{ww \mid w \text{ está em } \{0,1\}^*\}$

# Propriedades de LLC

Teorema: As LLCs são fechadas sob as seguintes operações:

1. União
2. Concatenação
3. Fechamento ( $*$ ) e fechamento positivo ( $+$ )
4. Reversão

# Propriedades de LLC

Teorema: As LLCs não são fechadas sob a interseção.

Contra-exemplo: Vimos que  $L = \{0^n 1^n 2^n\}$  não é LLC. Porém,  $L = L1 \cap L2$ , com

$L1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$  e

$L2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$ , e  $L1$  e  $L2$  são LLCs:

Gramáticas para

$L1: S \rightarrow AB$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

$L2: S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 1B2 \mid 12$

# Propriedades de LLC

Teorema: As LLCs são fechadas sob a operação "interseção com uma linguagem regular". Se  $L$  é uma LLC e  $R$  é uma LR, então  $L \cap R$  é uma LLC.

Teorema: As afirmativas a seguir são verdadeiras a respeito das LLCs  $L$ ,  $L1$  e  $L2$ , e para uma LR  $R$ :

1.  $\overline{L - R}$  é uma linguagem livre de contexto.
2.  $\overline{L}$  não é necessariamente uma LLC.
3.  $L1 - L2$  não é necessariamente uma LLC

# Complexidade da conversão entre *GLC* e *AP*

- O tempo da conversão é parte do custo dos algoritmos de decisão sobre *LLCs*, sempre que a linguagem é dada numa representação diferente daquela para a qual o algoritmo é projetado.



São Lineares no tamanho da entrada (e portanto rápidos e comparáveis à entrada):

- Converter uma GLC em um AP
- Converter um  $AP_F$  num  $AP_N$
- Converter um  $AP_N$  num  $AP_F$

# Conversão de GLC para FNC

- Teorema: Dada uma gramática  $G$  de comprimento  $n$ , podemos encontrar uma gramática equivalente na Forma Normal de Chomsky para  $G$  no tempo  $O(n^2)$ ; a gramática resultante tem comprimento  $O(n^2)$ .

# Testando o caráter vazio das LLCs

- Estratégia: Dada uma GLC, verifique se seu símbolo inicial,  $S$ , é gerador, isto é, se  $S$  deriva pelo menos uma cadeia.  $L$  é vazia se e somente se  $S$  não é gerador.

O algoritmo visto antes é  $O(n^2)$ , mas é possível sofisticá-lo de modo que fique  $O(n)$ .

# Teste de Pertinência de uma cadeia $w$ a uma LLC

- Algoritmo de tabulação CYK:  $O(n^3)$ , onde  $|w| = n$
- Exemplo: Seja  $G$  na FNC:

$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

Vamos testar a pertinência de *baaba*

Devemos preencher a seguinte tabela:

|    |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|
| L5 | X15 |     |     |     |     |
| L4 | X14 | X25 |     |     |     |
| L3 | X13 | X24 | X35 |     |     |
| L2 | X12 | X23 | X34 | X45 |     |
| L1 | X11 | X22 | X33 | X44 | X55 |

$b$        $a$        $a$        $b$        $a$

Onde  $X_{ij}$  é o conjunto de variáveis  $A$ , de  $V_n$ , tais que

$A \xrightarrow{*} w_i w_{i+1} \dots w_j$ . Assim, queremos saber se  $S \in X_{1n}$ , ou seja, se  $S \xrightarrow{*} w$

1ª. Linha L1 (BASE): Como a cadeia que se inicia e termina na posição  $i$  é apenas o terminal  $w_i$ , e como a gramática está na FNC, a única maneira de derivar a cadeia  $w_i$  é usar uma produção  $A \rightarrow w_i$ . Desse modo,  $X_{ii}$  é o conjunto de variáveis  $A$  tais que  $A \rightarrow w_i$  é uma produção de  $G$ .

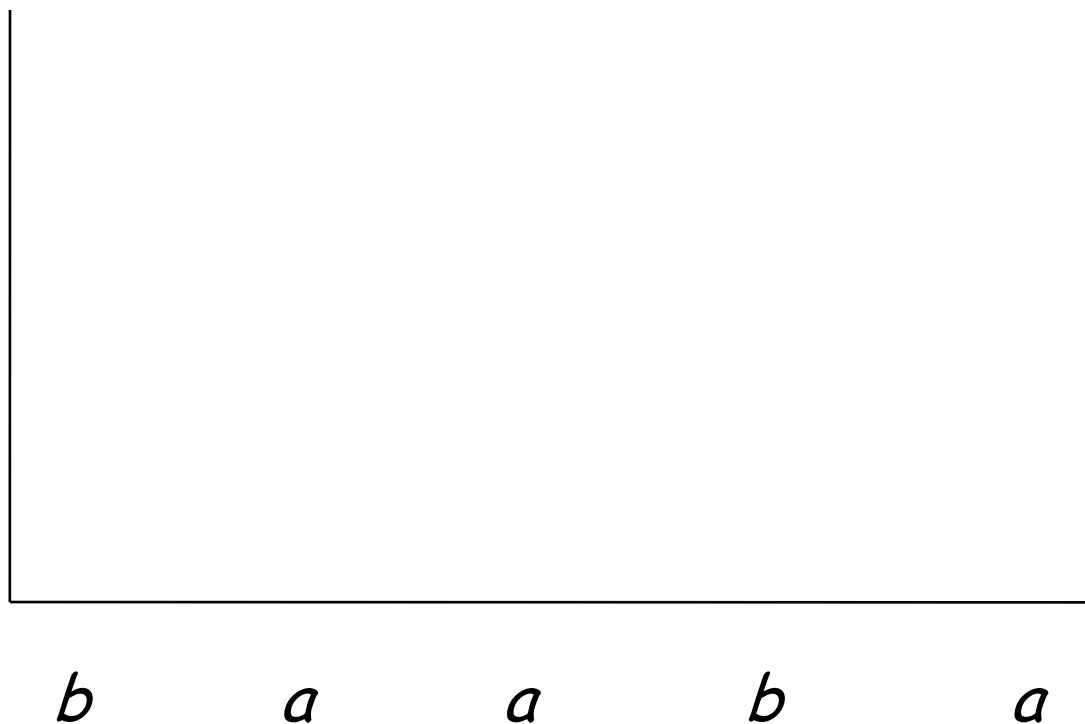
L5

L4

L3

L2

L1



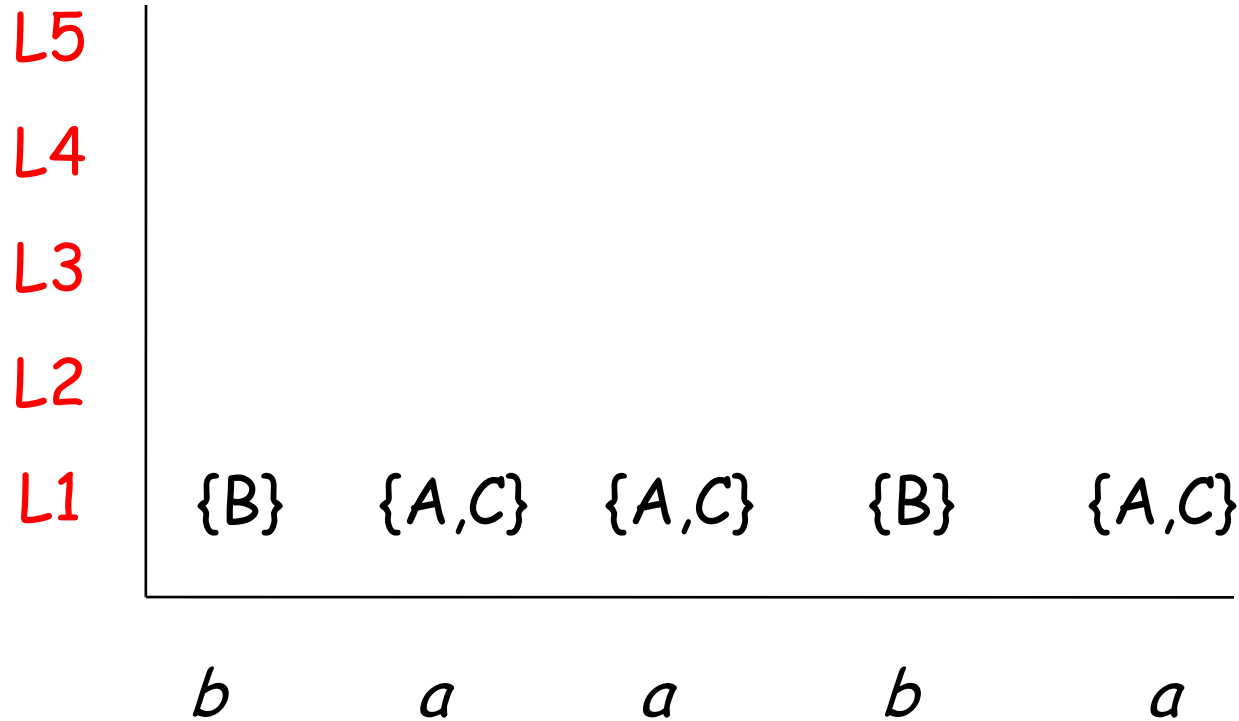
$S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

1ª. Linha L1 (BASE): Como a cadeia que se inicia e termina na posição  $i$  é apenas o terminal  $w_i$ , e como a gramática está na FNC, a única maneira de derivar a cadeia  $w_i$  é usar uma produção  $A \rightarrow w_i$ . Desse modo,  $X_{ii}$  é o conjunto de variáveis  $A$  tais que  $A \rightarrow w_i$  é uma produção de  $G$ .



$S \rightarrow AB \mid BC$   
 $A \rightarrow BA \mid a$   
 $B \rightarrow CC \mid b$   
 $C \rightarrow AB \mid a$

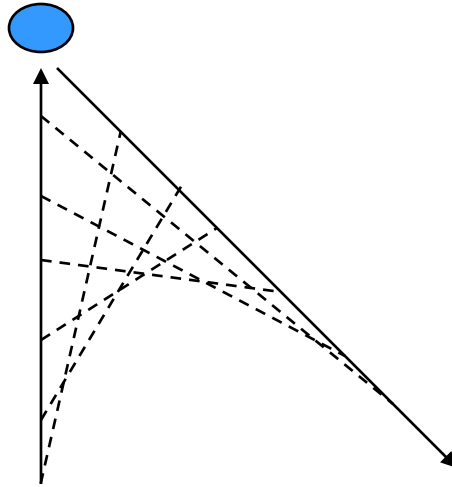
**Indução:** Suponha que queremos calcular  $X_{ij}$  que está na linha  $j-i+1$ , e que calculamos todos os  $X$ s nas linhas inferiores. Ou seja, conhecemos todas as cadeias mais curtas que  $w_i w_{i+1} \dots w_j$  e, em particular, conhecemos todos os prefixos e sufixos próprios dessa cadeia. Como  $j-i > 0$ , sabemos que qualquer derivação  $A \rightarrow^* w_i w_{i+1} \dots w_j$  deve começar com alguma etapa  $A \rightarrow BC$ . Então  $B$  deriva algum prefixo de  $w_i w_{i+1} \dots w_j$ , digamos  $B \rightarrow^* w_i w_{i+1} \dots w_k$ , para algum  $k < j$ . Além disso,  $C$  deve derivar então o restante de  $w_i w_{i+1} \dots w_j$ , isto é,  $C \rightarrow^* w_{k+1} w_{k+2} \dots w_j$ .

Concluimos que, para  $A$  estar em  $X_{ij}$ , devemos encontrar variáveis  $B$  e  $C$ , e um inteiro  $k$ , tais que:

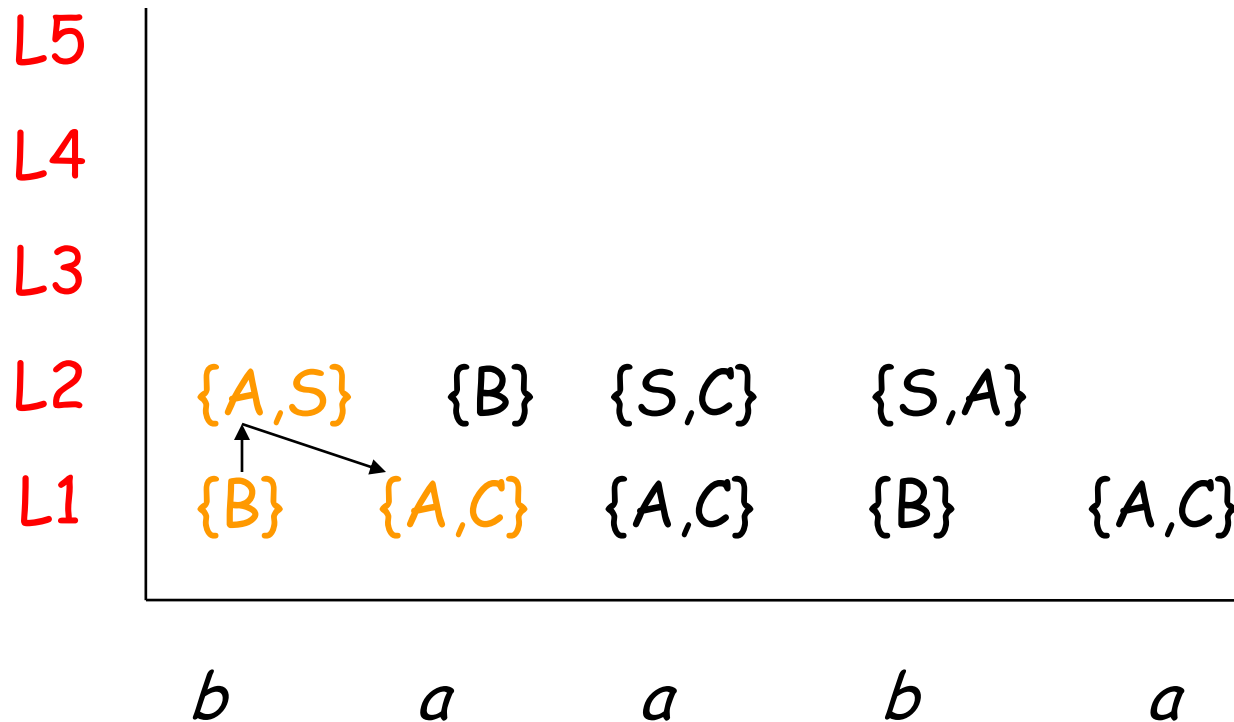
1.  $i \leq k < j$
2.  $B$  está em  $X_{ik}$
3.  $C$  está em  $X_{k+1, j}$
4.  $A \rightarrow BC$  é uma produção de  $G$ .



Encontrar as variáveis  $A$  exige a comparação de no máximo  $n$  pares de conjuntos calculados anteriormente  $(X_{ii}, X_{i+1,j})$ ,  $(X_{i,i+1}, X_{i+2,j}) \dots (X_{i,j-1}, X_{jj})$ :

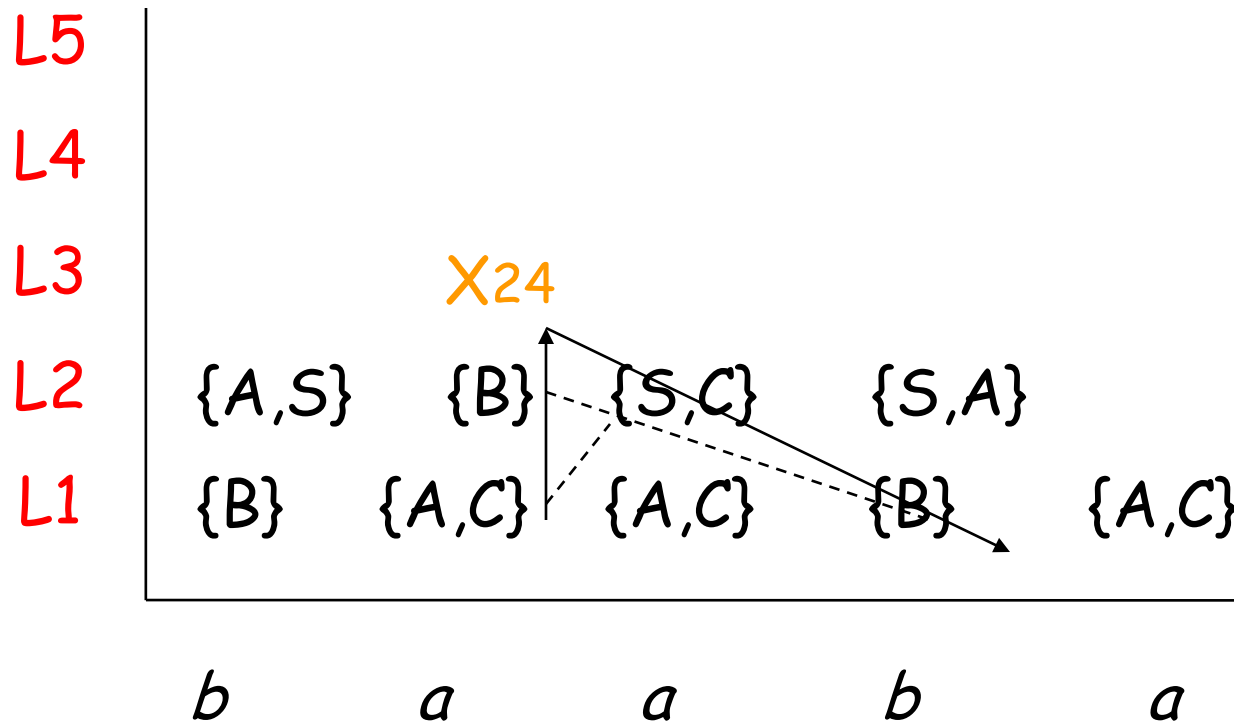


2º. Linha L2: X12: Para uma variável gerar  $ba$ , ela deve ter um corpo cuja primeira variável esteja em  $X11 = \{B\}$  (i.e., ela gera o  $b$ ) e cuja segunda variável esteja em  $X22 = \{A,C\}$  (i.e., ela gera o  $a$ ). Esse corpo só pode ser  $BA$  ou  $BC$ . As únicas regras de  $G$  possíveis são  $A \rightarrow BA$  e  $S \rightarrow BC$ . Portanto,  $X12 = \{A,S\}$



$S \rightarrow AB \mid BC$   
 $A \rightarrow BA \mid a$   
 $B \rightarrow CC \mid b$   
 $C \rightarrow AB \mid a$

3ª. Linha L3: X24: podemos desmembrar a cadeia aab que ocupa as posições 2 a 4 concluindo a primeira subcadeia depois da posição 2 ou da posição 3. Ou seja, podemos escolher  $k=2$  ou  $3$  na definição de X24. Assim, devemos considerar todos os corpos em  $X_{22} \cup X_{34} \cup X_{23} \cup X_{44}$ . Esse conjunto é  $\{A,C\}\{S,C\} \cup \{B\}\{B\} = \{AS, AC, CS, CC, BB\}$ . Desses, somente  $CC$  é um corpo e sua variável é  $B$ . Portanto,  $X_{24} = \{B\}$



$S \rightarrow AB \mid BC$   
 $A \rightarrow BA \mid a$   
 $B \rightarrow CC \mid b$   
 $C \rightarrow AB \mid a$

Como  $S$  está em  $X_{15}$ , então  $S \rightarrow^* baaba$

|    |         |         |       |       |       |     |
|----|---------|---------|-------|-------|-------|-----|
| L5 | {S,A,C} |         |       |       |       |     |
| L4 | -       | {S,A,C} |       |       |       |     |
| L3 |         | {B}     | {B}   |       |       |     |
| L2 | {A,S}   | {B}     | {S,C} | {S,A} |       |     |
| L1 | {B}     | {A,C}   | {A,C} | {B}   | {A,C} |     |
|    |         | $b$     | $a$   | $a$   | $b$   | $a$ |

$S \rightarrow AB \mid BC$   
 $A \rightarrow BA \mid a$   
 $B \rightarrow CC \mid b$   
 $C \rightarrow AB \mid a$

# Questões Indecidíveis sobre LLCs

Não existem algoritmos para responder as seguintes perguntas:

1. Uma dada LLC é inerentemente ambígua?
2. A interseção de duas LLCs é vazia?
3. Duas LLCs são iguais?
4. Uma dada LLC é igual a  $\Sigma^*$ , onde  $\Sigma$  é o alfabeto dessa linguagem?
5. Uma GLC  $G$  é ambígua?

# Resumo

- *Eliminação de símbolos inúteis:* uma variável pode ser eliminada de uma *GLC*, a menos que derive alguma cadeia de terminais e também apareça em pelo menos uma cadeia derivada do axioma. Para eliminar símbolos inúteis, primeiro devemos testar se uma variável deriva uma cadeia de terminais e eliminar aquelas que não o fazem, juntamente com todas suas produções. Só então eliminaremos variáveis que não são deriváveis do axioma.
- *Eliminação de produções nulas e unitárias:* dada uma *GLC*, podemos encontrar outra *GLC* que gere a mesma linguagem, com exceção da cadeia nula, ainda que não tenha nenhuma produção nula ou produções unitárias (um única variável como corpo).

# Resumo

- *Forma Normal de Chomsky*: dada uma GLC que derive ao menos uma cadeia não nula, podemos encontrar outra GLC que gere a mesma linguagem com exceção da cadeia nula, e que esteja na FNC': não há nenhum símbolo inútil e todo corpo de produção consiste em duas variáveis ou em um terminal.
- *O lema do bombeamento*: em qualquer LLC é possível encontrar, em qualquer cadeia suficientemente longa, uma subcadeia curta tal que os dois extremos dessa subcadeia possam ser "bombeados" em conjunto: i.e., cada um possa se repetir qualquer número de vezes desejado. As cadeias bombeadas não são ambas nulas. Esse lema nos permite provar que muitas linguagens não são livres de contexto.

# Resumo

- *Operações que preservam LLCs:* as LLCs são fechadas sob as operações de substituição, união, concatenação, clausura, inversão e homomorfismos inversos. As LLCs não são fechadas sob a interseção ou a complementação, mas a interseção de uma LLC e uma LR é sempre uma LLC.
- *Teste do caráter vazio de uma LLC:* dada uma GLC, existe um algoritmo para descobrir se ela gera alguma cadeia. Uma implementação cuidadosa permite que esse teste seja conduzido em tempo proporcional ao tamanho da própria gramática.
- *Teste de pertinência a uma LLC:* o algoritmo CYK informa se uma dada cadeia está em uma determinada LLC. Para uma LLC fixa, esse teste demora  $O(n^3)$ , se  $n$  é o comprimento da cadeia sendo testada.