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 λ) pode ser gerada por uma GLC cujas produções têm a forma (Forma Normal de Chomsky):

  $A \rightarrow BC$ ou $A \rightarrow a$ onde $A,B,C \in Vn$ e $a \in Vt$

Temos que:

1. Eliminar símbolos inúteis
2. Eliminar as λ-produções
3. Eliminar as produções unitárias

1. Eliminação de Símbolos Inúteis

- **Def:** um símbolo X de Vn ou Vt é útil para uma gramática G = (Vn,Vt,P,S) se existe alguma derivação da forma $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$ tal que $w \in Vt^*$; α e $\beta \in (Vn \cup Vt)^*$. 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 =>* w para alguma cadeia de terminais w. Todo terminal é fértil pois w =>* w.

2) X é acessível se existe uma derivação
S =>* $\alpha X \beta$ para algum α e β. 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 -> AB. A gramática fica:

S -> a

A -> b

- S e a são acessíveis a partir de S.
- Eliminando A e b ficamos com S -> 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 -> a

A -> b

que ainda possui A e b como inúteis.

Algorítmo para exclusão de símbolos inúteis

O algoritmo faz a exclusão em 2 etapas:
Considere G = (Vn,Vt,P,S)

1) Resulta em uma G1 = (Vn1,Vt,P1,S) somente com símbolos férteis.

Vn1 = $\varnothing$

Repita

Vn1 = Vn1 $\cup$ {A | A -> $\alpha \in$ P e $\alpha \in$ {Vt $\cup$ Vn1)*}

Até que o nro de elementos de Vn1 não aumente

P1 = P – {Y-> $\alpha X \beta$ | X não está em Vn1}

Algorítmo para exclusão de símbolos inúteis

O algoritmo faz a exclusão em 2 etapas:
Considere G = (Vn,Vt,P,S)

1) Resulta em uma G1 = (Vn1,Vt,P1,S) somente com símbolos férteis.

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

Vn1 = ∅

Repita

Vn1 = Vn1 ∪ {A | A -> α ∈ P e α ∈ {Vt ∪ Vn1)*}

Até que o nro de elementos de Vn1 não aumente

P1 = P – {Y-> αXβ | X não está em Vn1}

Vn1={S,A} P1={S->a; A->b}

2) Resulta em uma G2 = (Vn2,Vt2,P2,S) somente com símbolos acessíveis a partir de S

Vt2 = ∅

Vn2 = {S}

Vn1={S,A} P1={S->a; A->b}

Repita

Vn2 = Vn2 ∪ {A | X -> αAβ ∈ P1, X ∈ Vn2}

Vt2 = Vt2 ∪ {a | X -> αaβ ∈ P1, X ∈ Vn2}

Até que o nro de elementos de Vn2 e Vt2 não aumentem

P2 = P1 – {Y-> α | Y não está em Vn2}

Vn2={S} Vt2 = {a} P1={S->a}

Exemplo

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

P = { S -> aAa | bBb

A -> a | S

C -> c}

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

1) Iteração	Variáveis
inicio	∅
1	{A,C} A -> a e C -> c
2	{A,C,S} S -> aAa
3	{A,C,S}

A produção S -> bBb foi excluída de G1.

2) Iteração Variáveis Terminais

início	{S}	∅
1	{S,A}	{a}
2	{S,A}	{a}

A produção C -> c foi excluída pois C e c não pertencem à nova gramática G2.

Gramática final:

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

P = { S -> aAa

A -> a | 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-{λ} tem uma GLC sem λ-produções.
- A idéia é primeiro descobrir todas as variáveis anuláveis, ou seja, todo A de Vn tal que A=>* λ.
- Se A é anulável, sempre que A aparecer do lado de direito de uma regra, digamos B -> CAD, A poderá (ou não) derivar λ.
- Criamos 2 versões da produção:

  B -> CD (correspondente ao caso em que A teria sido usada para derivar λ), e

  B -> CAD
- E eliminamos todas as produções com lado direito λ.

Ex:. G:

S -> AB

A -> aAA | λ

B -> bBB | λ

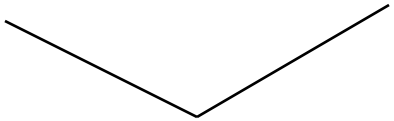
(1) Símbolos Anuláveis: S, A, B

(2) S -> AB => S -> AB | A | B

A -> aAA | λ => A -> aAA | aA | a

B -> bBB | λ => B -> bBB | bB | b

(3) Como S =>* λ => S -> λ é acrescida



G1 equivalente a G

3. Eliminação de Produções Unitárias (A -> B)

A produção A -> B não adiciona informação nenhuma e, se B -> α então podemos trocar A -> B por A -> α.

O algoritmo faz a exclusão em 2 etapas.

Seja G=(Vn, Vt, P, S):

1) Construção do fecho de cada variável

Para toda $A \in Vn$,

Fecho-A = {$B \in Vn$ | A <> B e $A \Rightarrow^{+} B$ usando só produções unitárias X -> Y}

Ex: E -> E + T | T Fecho-E = {T,F}

T -> T * F | F Fecho-T = {F}

F -> (E) | a Fecho-F = Ø

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

Seja P1 = P – {produções unitárias}

e

Para todo $A \in Vn$ e $B \in$ Fecho-A

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

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

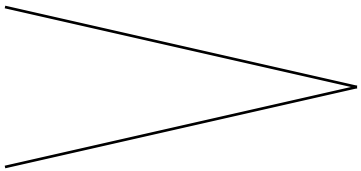
No exemplo:

E -> E + T | T
T -> T * F | F
F -> (E) | a

Fecho-E = {T,F}
Fecho-T = {F}

P1: E -> E + T
T -> T * F
F -> (E) | a

E -> E + T | T * F | (E) | a
T -> T * F | (E) | a
F -> (E) | 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 λ, tem uma gramática G em que todas as produções são de uma das 2 formas:

- A -> BC com A,B,C $\in$ Vn, ou
- A -> a com a $\in$ Vt

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 λ-produções; (o resultado são produções do tipo *A* -> 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 ->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

$A \rightarrow B_1B_2...B_k$, para k >2, em um grupo de produções com 2 variáveis em cada corpo. Introduzimos k-2 novas variáveis, $C_1, C_2, ...C_{k-2}$, em k-1 produções:

$A \rightarrow B_1C_1$, $C_1 \rightarrow B_2C_2$, ..., $C_{k-2} \rightarrow B_{k-1}B_k$

Exemplo

E -> E + T | T * F | (E) | a

T -> T * F | (E) | a

F -> (E) | a

(a):

E -> E M T | T V F | P E Q | a

T -> T V F | P E Q | a

F -> P E Q | a

P -> (

Q ->)

M -> +

V -> *

E -> E M T | T V F | P E Q | a

T -> T V F | P E Q | a

F -> P E Q | a

P -> (

Q ->)

M -> +

V -> *

(b):

E -> E X | T Y | P Z | a

X -> MT

Y -> VF

Z -> EQ

T -> T Y | P Z | a

F -> P Z | a

P -> (

Q ->)

M -> +

V -> *

Forma Normal de Greibach

- Toda LLC não vazia e sem λ tem uma gramática G em que todas as produções são da forma:

- A -> aα onde a $\in$ Vt e α 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.

GLC e LR

- As gramáticas (Vn, Vt, P e S) tal que toda produção de P seja da forma:

  $A \rightarrow a$ ou $A \rightarrow aB$

  onde A e $B \in Vn$ e $a \in Vt$

  são chamadas de Gramáticas Regulares.
- Teorema: as Gramáticas Regulares (GR) geram exatamente o conjunto das Linguagens Regulares. *(prove mostrando a equivalência entre GR e AF)*

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 ={$0^n1^n2^n$}. Suponha L livre de contexto. Então, existe um inteiro n dado pelo LB. Vamos escolher z = $0^n1^n2^n$. Podemos desmembrar z como z=uvwxy, onde |vwx| $\leq$ n e onde v e x não são ambos λ. 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:
  - $L = \{0^i1^j2^i3^j \mid i \geq 1 \text{ e } j \geq 1\}$
  - $L = \{ww \mid w \text{ está em } \{0,1\}^*\}$

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 $L = \{0^n1^n2^n\}$ não é LLC. Porém, $L = L1 \cap L2$, com

$L1 = \{0^n1^n2^i | n \geq 1, i \geq 1\}$ e

$L2 = \{0^i1^n2^n | n \geq 1, i \geq 1\}$, e L1 e L2 são LLCs:

Gramáticas para

L1: S -> AB
A -> 0A1 | 01
B -> 2B | 2

L2: S -> AB
A -> 0A | 0
B -> 1B2 | 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 AP em GLC

- Teorema: Existe um algoritmo de $O(n^3)$ que toma um AP_N, P, cuja representação tem comprimento n e produz uma GLC de comprimento no máximo igual a $O(n^3)$. Opcionalmente, podemos fazer G gerar L(P) por estado final.

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 → AB | BC

  A → BA | a

  B → CC | b

  C → AB | a

Vamos testar a pertinência de *baaba*

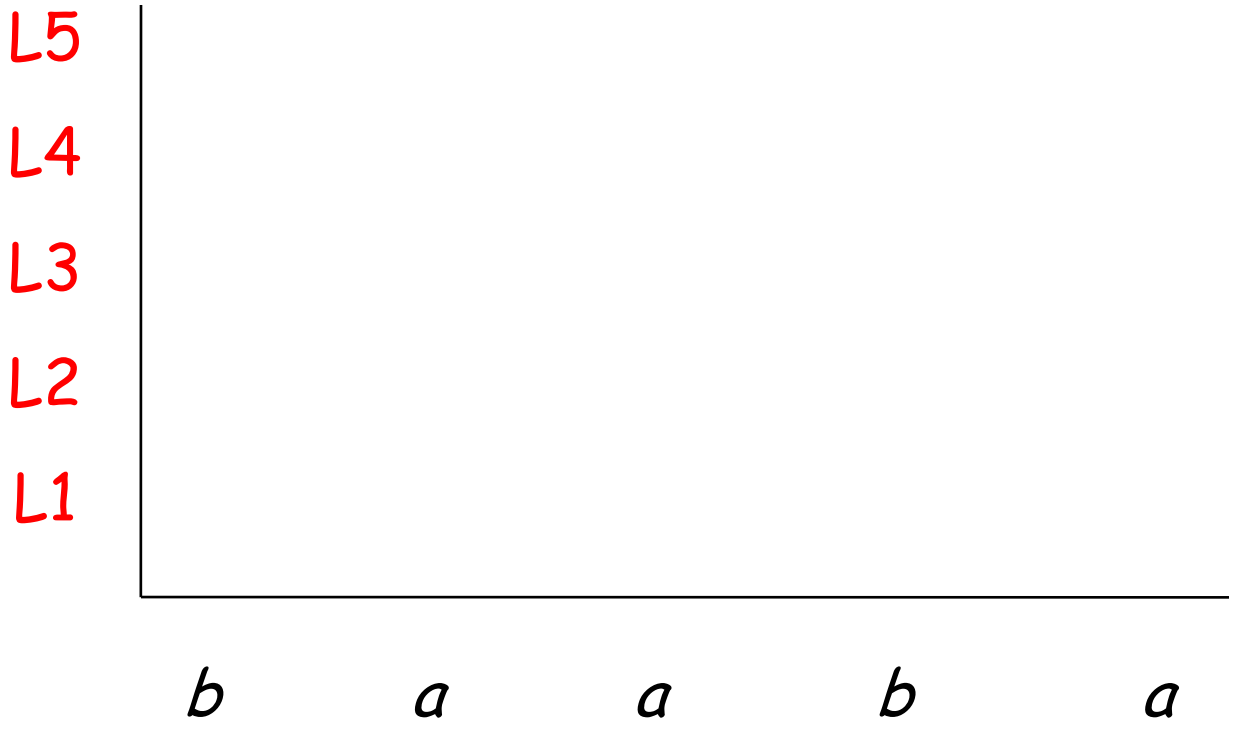
Devemos preencher a seguinte tabela:

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 Xij é o conjunto de variáveis A, de Vn, tais que

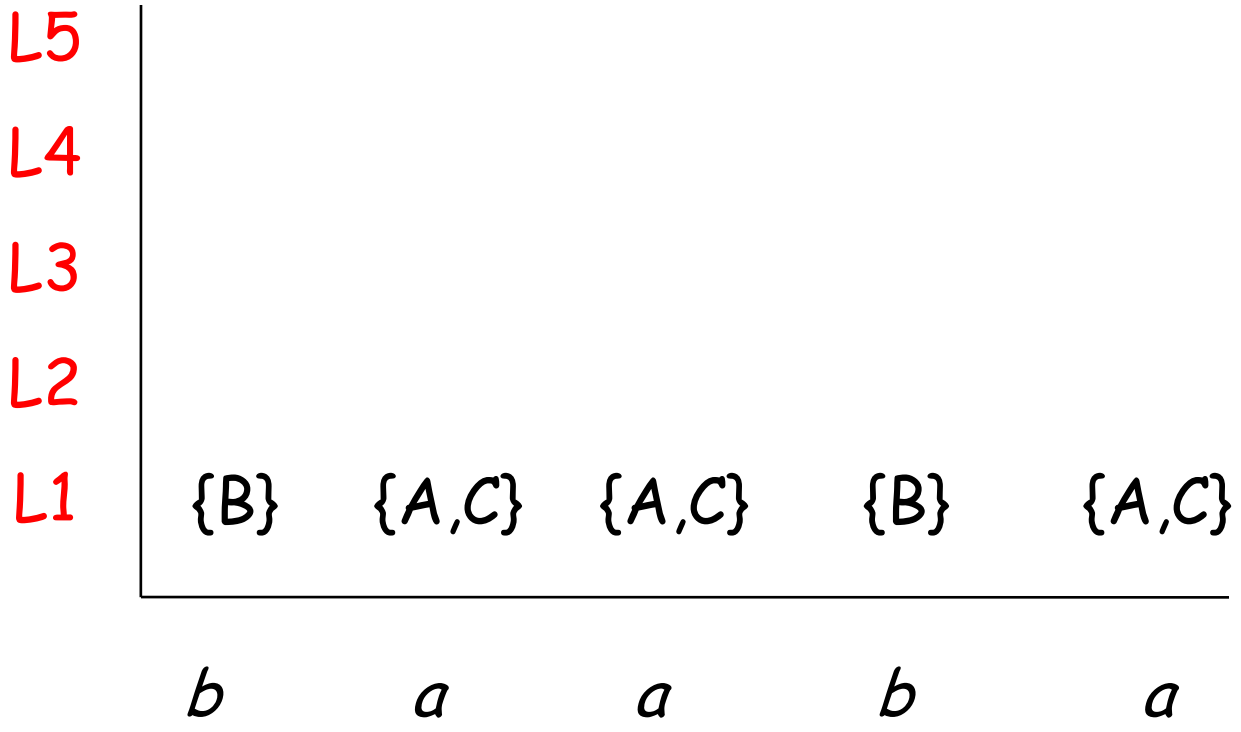
$A \rightarrow^* w_i\, w_{i+1} \ldots w_j$. Assim, queremos saber se $S \in X1n$, ou seja, se $S \rightarrow^* w$

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, Xii é o conjunto de variáveis A tais que $A \rightarrow w_i$ é uma produção de G.



S → AB | BC
A → BA | a
B → CC | b
C → AB | 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, Xii é o conjunto de variáveis A tais que $A \rightarrow w_i$ é uma produção de G.



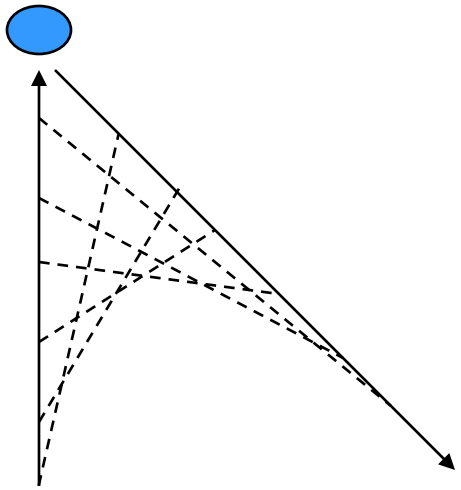
S → AB | BC
A → BA | a
B → CC | b
C → AB | a

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

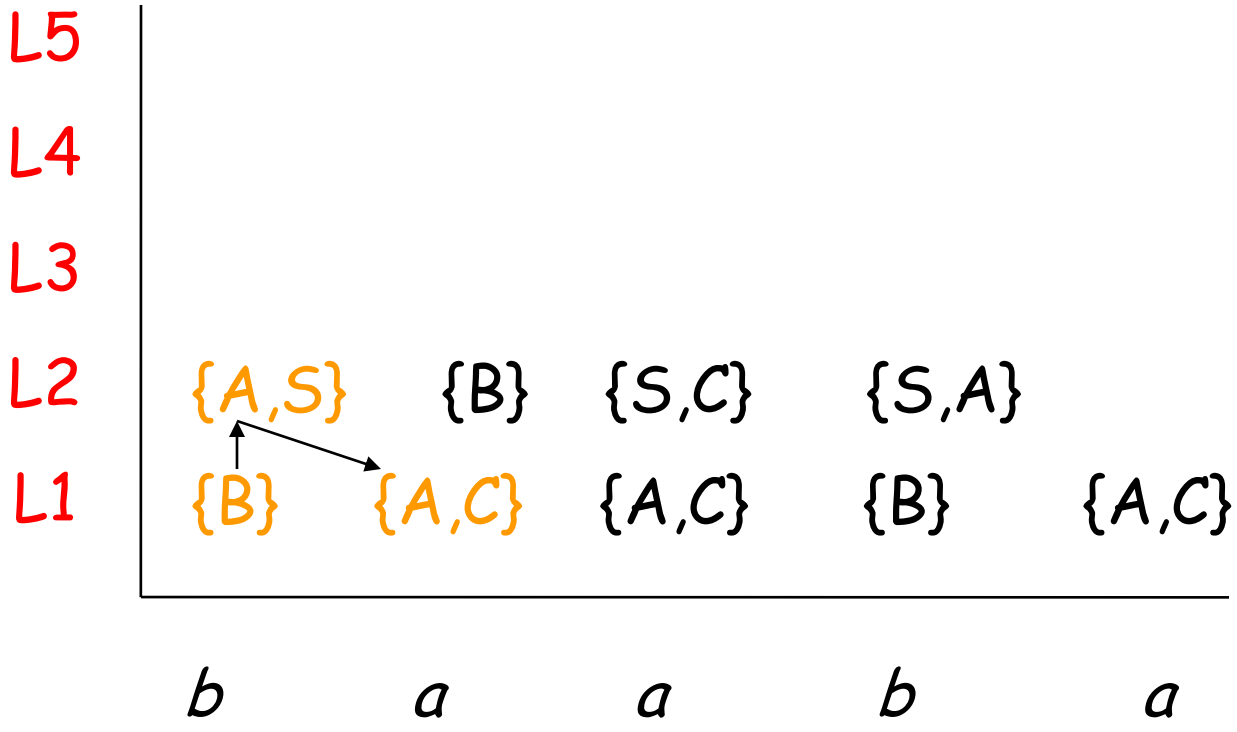
Concluímos que, para A estar em Xij, devemos encontrar variáveis B e C, e um inteiro k, tais que:

1. $i \leq k < j$
2. B está em Xik
3. C está em Xk+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}, $X_{i+1,j}$), ($X_{i,i+1}$, $X_{i+2,j}$) ...($X_{i,j-1}$, X_{jj}):

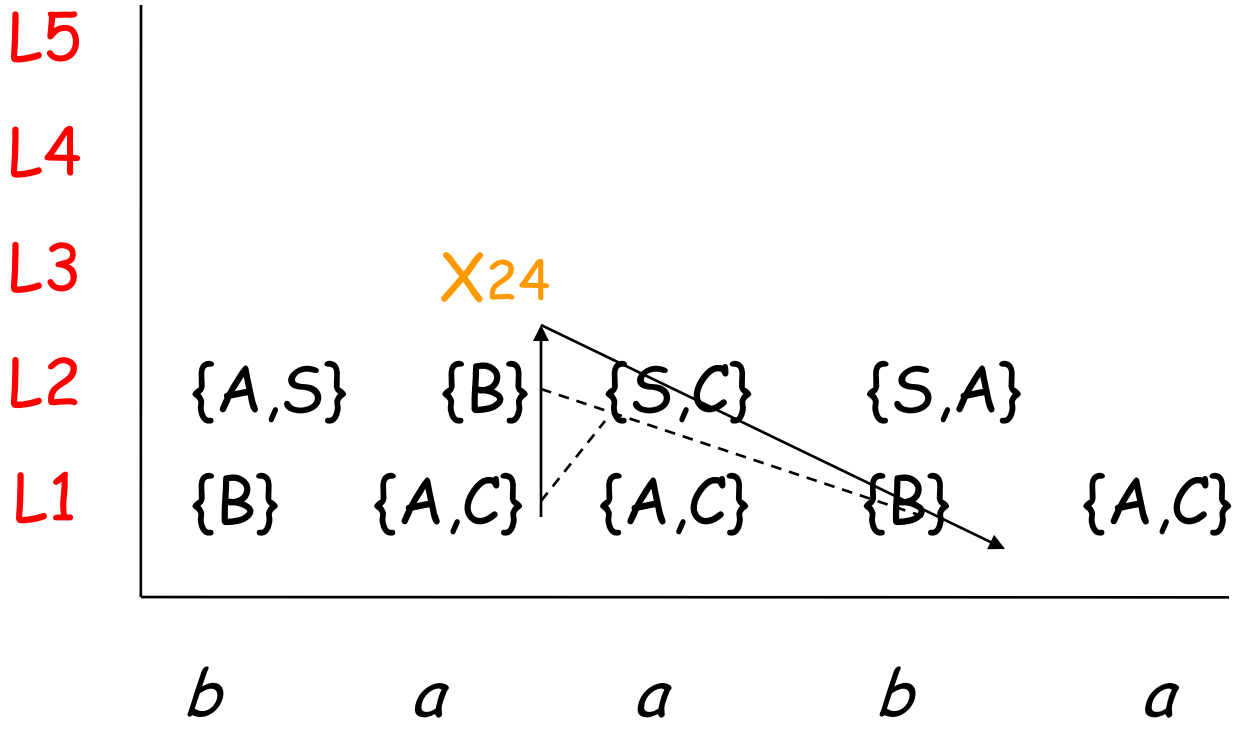


2ª. Linha L2: X12: Para uma variável gerar *ba*, ela deve ter um corpo cuja primeira variável esteja em X11 = {B} (i.e., ela gera o *b*) e cuja segunda variável esteja em X22 = {*A*,*C*} (i.e., ela gera o *a*). Esse corpo só pode ser BA ou BC. As únicas regras de G possíveis são A→ BA e S → BC. Portanto,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



S → AB | BC
A → BA | a
B → CC | b
C → AB | 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 X22 X34 U X23 X44. Esse conjunto é {A,C}{S,C} U {B}{B} = {AS, AC, CS, CC, BB}. Desses, somente CC é um corpo e sua variável é B. Portanto, X24 = {B}



S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Como S está em X15, então S ➔* *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 → AB | BC
A → BA | a
B → CC | b
C → AB | 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 Σ^*, onde Σ 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.