ITC: Introdução à Teoria da Computação

Marcos Castilho

DInf/UFPR

15 de junho de 2021

Alguns teoremas sobre GLC's

A imposição de certas restrições na forma das regras de uma Gramática Livre de Contexto garantem algumas propriedades interessantes, por exemplo para analisadores sintáticos que terminam ou algumas outras caracterizações teóricas importantes.

Eliminação de regras-λ

Veremos como construir uma gramática G_L a partir de uma gramática G de tal maneira que as produções em G_L não contêm regras-λ. Na verdade, pode haver uma única.

Mas antes precisamos eliminar a recursividade do símbolo de partida.

Eliminação de recursividade no símbolo de partida

Lema 1: Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto. Existe uma outra GLC $G' = (V', \Sigma, P', S')$ que satisfaz:

(i) $L(G) = L(G')$

(ii) As regras de P são da forma $A \to w$, onde $A \in V'$ e $w \in ((V - \{S'\}) \cup \Sigma)^*$

Prova

Se S não ocorre no lado direito de nenhuma regra de G então $G = G'$. Portanto, considere que S é recursiva. Devemos fazer a seguinte alteração:

$$G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \to S\}, S')$$

Onde S' é o novo símbolo de partida ($S' \notin V$). S' serve apenas para iniciar a derivação.
Obviamente $L(G) = L(G')$ pois se $w \in L(G)$ então $S \overset{*}{\underset{G}{\Rightarrow}} w$.

Mas $S' \overset{1}{\underset{G'}{\Rightarrow}} S \overset{*}{\underset{G'}{\Rightarrow}} w$.

Exemplo

$$G : \begin{cases} S \to aS \mid AB \mid AC \\ A \to aA \mid \lambda \\ B \to bB \mid bS \\ C \to cC \mid \lambda \end{cases}$$

$$G' : \begin{cases} S' \to S \\ S \to aS \mid AB \mid AC \\ A \to aA \mid \lambda \\ B \to bB \mid bS \\ C \to cC \mid \lambda \end{cases}$$

Observações

Uma propriedade interessante para GLC's é que as derivações sempre aumentem de tamanho, ou que pelo menos mantenha o tamanho das formas sentenciais;

Um tipo de regra que permite reduzir o tamanho das formas sentenciais é uma regra-λ;

Chamamos de *variáveis que constituem produções* λ*, ou produções vazias*:

> Inicialmente, as produções-λ ($A \to \lambda$)
> Sucessivamente, as variáveis que geram λ indiretamente (por exemplo $B \to A$; $A \to \lambda$)

Uma GLC que não contém estas variáveis é chamada de *não contrativa* (do inglês *non contracting*).

Eliminação de regras-λ

Algoritmo para obtenção do conjunto de variáveis que constituem produções vazias:

Entrada: Uma GLC $G = (V, \Sigma, P, S)$

$NULL := \{A \mid A \to \lambda \in P\}$
REPEAT
    $PREV := NULL$
    FOR "toda variável $A \in V$" DO
        IF "existe $A \to w \in P$" E $w \in PREV^*$ THEN
            $NULL := NULL \cup \{A\}$
UNTIL $NULL = PREV$

Exemplo

$$G : \begin{cases} S \to ACA \\ A \to aAa \mid B \mid C \\ B \to bB \mid b \\ C \to cC \mid \lambda \end{cases}$$

Iteração	*NULL*	*PREV*
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Lema

Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto. O algoritmo anterior gera o conjunto de variáveis que constituem produções vazias.

Prova

Mostrar que, ao final do algoritmo, toda variável em *NULL* deriva λ;

Mostrar que toda variável que constitui produções λ será inserida em *NULL*.

Prova, continuação

A prova do primeiro ponto é por indução no número de iterações.

Base: Suponha que uma variável A seja inserida em *NULL* no passo 1 do algoritmo. Então G contém $A \to \lambda$ e a derivação é $A \Rightarrow \lambda$.

HI: Suponha que todas as variáveis que pertencem à *NULL* após n iterações do algoritmo constituem produções vazias.

Passo indutivo: Seja A uma variável inserida em *NULL* na iteração $n + 1$.

Prova, continuação

Então, por construção, existe uma regra do tipo:

$$A \to A_1 A_2 \ldots A_k$$

Onde cada $A_i \in PREV$ na iteração $n+1$. Mas $A_i \stackrel{*}{\Rightarrow} \lambda$, para $i = 1, \ldots k$, pela HI.

Assim:

$$A \Rightarrow A_1 A_2 \ldots A_k \stackrel{*}{\Rightarrow} A_2 \ldots A_k \stackrel{*}{\Rightarrow} \ldots \stackrel{*}{\Rightarrow} A_k \stackrel{*}{\Rightarrow} \lambda$$

O que mostra que A constitui produções vazias.

Prova, continuação

Agora provaremos a parte 2, isto é, que toda variável que constitui produções λ será inserida em *NULL*.

Suponha que $A \stackrel{n}{\Rightarrow} \lambda$. Vamos mostrar que A é inserida em *NULL* no máximo na iteração n. De fato, por indução no tamanho da derivação:

BASE: Se $A \stackrel{1}{\Rightarrow} \lambda$, então A é inserida em *NULL* no passo 1 do algoritmo.

HI: Suponha agora que todas as variáveis que derivam λ em no máximo n aplicações da regra foram inseridas em *NULL* no máximo na iteração n.

Prova, continuação

Passo indutivo: Seja A uma variável que deriva λ em uma derivação de tamanho $n+1$, isto é:

$$A \Rightarrow A_1 A_2 \ldots A_k \stackrel{n}{\Rightarrow} \lambda.$$

Cada A_i deriva λ em no máximo n passos (por construção). Logo, cada A_i satisfaz a HI e assim está em *NULL* antes da iteração $n+1$. Seja $m \leq n$ a iteração em que todos os A_i's estão em *NULL*. Na iteração $m+1$ a regra $A \rightarrow A_1 A_2 \ldots A_k$ permite inserir A em *NULL*.

Exemplo

$$G : \begin{cases} S \to aS \mid AB \mid AC \\ A \to aA \mid \lambda \\ B \to bB \mid bS \\ C \to cC \mid \lambda \end{cases}$$

$$NULL = \{S, A, C\}$$

$$G_1 : \begin{cases} S' \to S \\ S \to aS \mid AB \mid AC \\ A \to aA \mid \lambda \\ B \to bB \mid bS \\ C \to cC \mid \lambda \end{cases}$$

$$NULL = \{S', S, A, C\}$$

Lema 2

Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto:

Se $A \overset{*}{\underset{G}{\Rightarrow}} w$, então $G' = (V, \Sigma, P \cup \{A \to w\}, S)$ é equivalente à G $(L(G) = L(G'))$.

Prova

Toda produção de G está em G'. Assim, é fácil ver que $L(G) \subseteq L(G')$.
Veremos agora a prova de que $L(G') \subseteq L(G)$

Prova, continuação

Seja $w \in L(G')$ e que a produção $A \to w$ foi usada na derivação. Podemos obter w em G assim:

$$S \overset{*}{\underset{G}{\Rightarrow}} A \overset{*}{\underset{G}{\Rightarrow}} w.$$

Portanto, $w \in L(G)$.

Aplicação do lema

Uma gramática sem regras-λ é *não contrativa*.

Suponha que $B \in V$ constitui produções λ;

Suponha uma regra do tipo: $A \to BAa$;

Se B pode derivar λ, então:
$A \Rightarrow BAa \stackrel{*}{\Rightarrow} Aa \stackrel{*}{\Rightarrow} w$;

Assim, basta substituir as regras-λ por $A \to Aa$;

Se temos outra regra do tipo $A \to BABa$, basta substituir por quatro outras regras:

$A \to BABa$; $A \to ABa$; $A \to BAa$ e $A \to Aa$.

Teorema

Seja $G = (V, \Sigma, P, S)$ uma gramática livre de contexto. Existe um algoritmo que constrói uma GLC $G_L = (V_L, \Sigma, P_L, S_L)$ tal que:

(i) $L(G_L) = L(G)$

(ii) S_L não é recursiva

(iii) $A \to \lambda \in P_L$ se, e somente se, $\lambda \in L(G)$ e $A = S_L$.

Prova

Aplica-se a técnica do lema 1, resolvemos (ii). Assim, $V_L = V \cup \{S'\}$ ou $V_L = V$.

Quanto às produções de G_L, construimos:

(i) Se $\lambda \in L(G)$, então $S_L \to \lambda \in P_L$

(ii) Seja $A \to w \in P$. Suponha que $w = w_1 A_1 w_2 A_2 \ldots w_k A_k w_{k+1}$, onde $A_1, A_2, \ldots, A_k$ são subconjuntos do conjunto de variáveis que constituem produções vazias que ocorrem em w. Assim:

$$A \to w_1 w_2 \ldots w_k w_{k+1} \in P_L.$$

(iii) $A \to \lambda \in P_L$ apenas se $\lambda \in L(G)$ e $A = S_L$.

Devemos provar que a gramática G_L assim construída é tal que $L(G) = L(G_L)$.

Prova, continuação

($\Rightarrow$) $L(G_L) \subseteq L(G)$, pois as derivações em G_L usam regras de G e outras criadas em (ii), e cada uma destas regras é derivável em G.

Prova, continuação

($\Leftarrow$) $L(G) \subseteq L(G_L)$, pois toda palavra não nula em $L(G)$ é derivável em $L(G_L)$. Seja $A \overset{n}{\underset{G}{\Rightarrow}} w$ uma derivação em G tal que $w \in \Sigma^+$.

Se $n = 1$, $A \to w \in P$ e como $w \neq \lambda$ também está em P_L, assuma que toda palavra derivável a partir de A por n ou menos regras pode ser derivada a partir de A em G_L.

Seja $A \overset{n+1}{\underset{G}{\Longrightarrow}} w \mid w \in \Sigma^+$. Isto é:

$$A \Rightarrow w_1 A_1 w_2 A_2 \ldots w_k A_k w_{k+1} \overset{n}{\underset{G}{\Rightarrow}} w, \text{ onde } A_i \in V \text{ e } w_i \in \Sigma^*.$$

Prova, continuação

Pelo lema da aula anterior:

$$w = w_1 p_1 w_2 p_2 \ldots w_k p_k w_{k+1},$$

Onde $A_i \overset{m}{\underset{G}{\Rightarrow}} p_i, m \leq n$. Para cada $p_i \in \Sigma^+$, a hipótese de indução garante que $A_i \overset{*}{\underset{G_L}{\Rightarrow}} p_i$.

Se $p_j = \lambda$, A_j constitui produção vazia e a construção (ii) gera a regra:

$$A \rightarrow w_1 A_1 w_2 A_2 \ldots w_k A_k w_{k+1},$$

em que cada A_j que deriva λ é deletado.

Uma derivação em G_L pode ser construída começando-se por esta regra e derivando cada $p_i \in \Sigma^+$ usando as derivações que existem dada a hipótese de indução.

Exemplo

$$G : \begin{cases} S \to ACA \\ A \to aAa \mid B \mid C \\ B \to bB \mid b \\ C \to cC \mid \lambda \end{cases}$$

$NULL = \{S, A, C\}$, portanto, por (i),
$S_L \to \lambda \in P_L$
(ii)

$C \to cC$, onde $\{C\} \subseteq NULL$, assim, $C \to c \in P_L$

$A \to aAa$, onde $\{A\} \subseteq NULL$, assim, $A \to aa \in P_L$

$S \to ACA$, onde $\{A, C\} \subseteq NULL$, assim,
$S \to AC \mid CA \mid AA \mid A \mid C \in P_L$

$C \to \lambda$, mas $C \neq S_L$, portanto deleta a regra!

Exemplo, continuação

$$G_L : \begin{cases} S_L \to S \mid \lambda \\ S \to ACA \mid AC \mid CA \mid AA \mid C \mid A \\ A \to aAa \mid aa \mid B \mid C \\ B \to bB \mid b \\ C \to cC \mid c \end{cases}$$

Exemplo de derivação em G para aba:

$S \Rightarrow ACA \Rightarrow aAaCA$
$\Rightarrow aBaCA \Rightarrow abaCA$
$\Rightarrow abaA \Rightarrow abaC$
$\Rightarrow aba$

Exemplo de derivação em G_L para aba:

$S \Rightarrow A \Rightarrow aAa \Rightarrow aBa \Rightarrow aba$

Exemplo 2: $a^*b^*c^*$

$$G : \begin{cases} S \to ABC \\ A \to aA \mid \lambda \\ B \to bB \mid \lambda \\ C \to cC \mid \lambda \end{cases}$$

$$G_L : \begin{cases} S_L \to S \mid \lambda \\ S \to ABC \mid AB \mid AC \mid BC \mid A \mid B \mid C \\ A \to aA \mid a \\ B \to bB \mid b \\ C \to cC \mid c \end{cases}$$

Eliminação de regras de cadeia

Regras de cadeia são as que têm a forma $A \to B$

Exemplo:

G:

$A \to aA \mid a \mid B$
$B \to bB \mid b \mid c$

G_1:

$A \to aA \mid a \mid bB \mid b \mid c$
$B \to bB \mid b \mid c$

Construção do conjunto $cadeia(A)$

Entrada: Uma GLC $G = (V, \Sigma, P, S)$ essencialmente não contrativa

1. $cadeia(A) := \{A\}$
2. $PREV := \emptyset$
3. REPEAT
   - 3.1 $NEW := cadeia(A) - PREV$
   - 3.2 $PREV := cadeia(A)$
   - 3.3 FOR "toda variável $B \in NEW$" DO
     - FOR "toda regra $B \to C$" DO
       - $cadeia(A) := cadeia(A) \cup \{C\}$

UNTIL $cadeia(A) = PREV$

Lema

Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto essencialmente não contrativa. O algoritmo anterior gera o conjunto de variáveis que são deriváveis de A apenas usando regra de cadeia.

Teorema

Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto essencialmente não contrativa. Existe um algoritmo que constrói uma GLC G_c tal que:

(i) $L(G_c) = L(G)$;

(ii) G_c não tem regras de cadeia.

Prova

Construímos P_c assim: Para cada variável $A \in V$, acrescentamos em P_c as regras:

$$A \to w$$

Se existe uma variável B tal que:

(i) $B \in cadeia(A)$

(ii) $B \to w \in P$

(iii) $w \notin V$.

Ainda:

$$V_c = V,\ \Sigma_c = \Sigma,\ S_c = S.$$

Mostraremos que $L(G) = L(G_c)$

Prova, continuação

O lema 2 garante que as palavras deriváveis em G_c são também deriváveis em G.

Seja $w \in L(G)$ e $A \overset{*}{\underset{G}{\Rightarrow}} B$ uma sequência maximal de regras de cadeia usadas na derivação de w:

$$S \overset{*}{\underset{G}{\Rightarrow}} uAv \overset{*}{\underset{G}{\Rightarrow}} uBv \underset{G}{\Rightarrow} upv \overset{*}{\underset{G}{\Rightarrow}} w.$$

Onde $B \to p$ é uma regra que não é de cadeia.
A regra $A \to p$ pode substituir a sequência de derivações em cadeia. Usando-se esta técnica repetidas vezes, podemos remover todas as aplicações em cadeia na derivação de w, obtendo uma derivação de w em G_c.

Exemplo

$$G : \begin{cases} S \to ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda \\ A \to aAa \mid aa \mid B \mid C \\ B \to bB \mid b \\ C \to cC \mid c \end{cases}$$

G_c é essencialmente não contrativa, portanto:

$cadeia(S) = \{S, A, B, C\}$

$cadeia(A) = \{A, B, C\}$

$cadeia(B) = \{B\}$

$cadeia(C) = \{C\}$

Exemplo, continuação

$$G : \begin{cases} S \to ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda \\ A \to aAa \mid aa \mid B \mid C \\ B \to bB \mid b \\ C \to cC \mid c \end{cases}$$

Portanto:

$$P_c : \begin{cases} S \to ACA \mid CA \mid AA \mid AC \mid aAa \mid aa \mid bB \mid b \mid cC \mid c \mid \lambda \\ A \to aAa \mid aa \mid bB \mid b \mid cC \mid c \\ B \to bB \mid b \\ C \to cC \mid c \end{cases}$$

Observações

Eliminar regras de cadeia aumenta o número de regras, mas reduz o tamanho das derivações;

G_c também é essencialmente não contrativa;

Cada regra em G_c tem uma das seguintes formas:

(i) $S \to \lambda$
(ii) $A \to a$
(iii) $A \to w$, onde $w \in (V \cup \Sigma)^*$ e $|w| \geq 2$.

Os analisadores *bottom-up* são completos para verificar se $w \in L(G_c)$.

Símbolos inúteis

Definição:
Seja G uma Gramática Livre de Contexto. Um símbolo $x \in (V \cup \Sigma)$ é *útil* se existe uma derivação

$$S \underset{G}{\overset{*}{\Rightarrow}} uxv \underset{G}{\overset{*}{\Rightarrow}} w,$$

Onde $u, v \in (\Sigma \cup V)^*$ e $w \in \Sigma^*$.
Um símbolo que não é útil é *inútil*.

Exemplo: $L(G) = b^+$

$$G : \begin{cases} S \to AC \mid BS \mid B \\ A \to aA \mid aF \\ B \to CF \mid b \\ C \to cC \mid D \\ D \to aD \mid BD \mid C \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

Algoritmo para encontrar as variáveis que geram terminais

Entrada: Uma GLC $G = (V, \Sigma, P, S)$

1. $TERM := \{A \mid \text{existe } A \to w \in P \text{ tal que } w \in \Sigma^*\}$
2. REPEAT
   2.1 $PREV := TERM$
   2.2 FOR "toda variável $A \in V$" DO
   IF "existe $A \to w$" e $w \in (PREV \cup \Sigma)^*$ THEN
   $TERM := TERM \cup \{A\}$

UNTIL $PREV = TERM$

Teorema

Seja $G = (V, \Sigma, P, S)$ uma gramática livre de contexto. Existe um algoritmo que constrói uma GLC $G_T = (V_T, \Sigma, P_T, S)$ tal que:

(i) $L(G_T) = L(G)$

(ii) Toda variável em G_T deriva uma palavra terminal em G_T.

Prova

P_T é obtido assim: remove-se as regras que contêm variáveis de G que estão em $V - TERM$.

$$V_T = TERM$$

$$P_T = \{A \to w \in P \mid A \in TERM \text{ e } w \in (TERM \cup \Sigma)^*\}$$

$$\Sigma_T = \{a \in \Sigma \mid \exists A \to uav \in P_T\}$$

Resta provar a dupla inclusão: $L(G_T) \subseteq L(G)$ e $L(G) \subseteq L(G_T)$

Prova, continuação

($\Rightarrow$) $L(G_T) \subseteq L(G)$. Seja $w \in L(G_T)$. Então $S \overset{*}{\underset{G_T}{\Longrightarrow}} w$. Mas $P_T \subseteq P$, e portanto $S \overset{*}{\underset{G}{\Rightarrow}} w$.

Prova, continuação

($\Leftarrow$) $L(G) \subseteq L(G_L)$. Seja $w \in L(G)$. Então $S \overset{*}{\underset{G}{\Rightarrow}} w$. Suponha, por contradição, que $S \overset{*}{\underset{G_T}{\not\Longrightarrow}} w$.

Então existe $A \in (V - TERM)$ que ocorre em um passo intermediário da derivação de w (em G). Mas uma derivação de A não pode gerar uma palavra terminal, pela construção do algoritmo.

Assim, todas as regras aplicadas na derivação estão em P_T e portanto $w \in L(G_T)$.

Exemplo

$$G : \begin{cases} S \to AC \mid BS \mid B \\ A \to aA \mid aF \\ B \to CF \mid b \\ C \to cC \mid D \\ D \to aD \mid BD \mid C \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

iteração	*TERM*	*PREV*
0	$\{B, F\}$	
1	$\{B, F, A, S\}$	{ B,F }
2	$\{B, F, A, S, E\}$	{ B,F,A,S }
3	$\{B, F, A, S, E\}$	{ B,F,A,S,E }

Exemplo, continuação

$$G : \begin{cases} S \to AC \mid BS \mid B \\ A \to aA \mid aF \\ B \to CF \mid b \\ C \to cC \mid D \\ D \to aD \mid BD \mid C \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

$$G_T : \begin{cases} S \to BS \mid B \\ A \to aA \mid aF \\ B \to b \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

Algoritmo para encontrar as variáveis atingíveis

Entrada: Uma GLC $G = (V, \Sigma, P, S)$

1. $REACH := \{S\}$
2. $PREV := \emptyset$
3. REPEAT
   - 3.1 $NEW := REACH - PREV$
   - 3.2 $PREV := REACH$
   - 3.3 FOR "$A \in NEW$" DO
     - FOR "$A \to w \in P$" DO
       - "adiciona variáveis em w a $REACH$

UNTIL $REACH = PREV$

Lema

Seja $G = (V, \Sigma, P, S)$ uma Gramática Livre de Contexto O algoritmo anterior gera o conjunto de variáveis atingíveis a partir de S.

Prova

(I) Toda variável em *REACH* é derivável de S

(II) Toda variável atingível a partir de S é adicionada ao conjunto *REACH*.

Prova, continuação

I- Indução no número de iterações do algoritmo.
BASE: na linha 1, $REACH = \{S\}$, e S é derivável a partir de S.
HI: Suponha que todas as variáveis em $REACH$ após n iterações são deriváveis de S.
Passo indutivo: Seja B uma variável adicionada em $REACH$ após $n+1$ iterações. Então existe $A \to uBv \in P \mid A \in REACH$ após n iterações. Pela HI, existe uma derivação tal que $S \stackrel{*}{\Rightarrow} xAy$.
Aplicando a regra $A \to uBv$ mostramos a derivação de B.

Prova, continuação

II- Se $S \stackrel{n}{\Rightarrow} uAv$ então A é adicionada em $REACH$ no máximo na iteração n. Se A é atingível por uma derivação de tamanho zero, então ela entra em S no passo 1 do algoritmo.

Suponha que se uma variável é atingível por uma derivação de tamanho n (ou menos) então $A \in REACH$ na derivação n (ou antes).

Seja $S \stackrel{n}{\Rightarrow} xAy \Rightarrow xuBvy$ uma derivação de G. Pela HI, $A \in REACH$ na derivação n e B é adicionado na iteração seguinte.

Teorema

Seja $G = (V, \Sigma, P, S)$ uma GLC. Existe um algoritmo que produz uma GLC G_U tal que:

(i) $L(G_U) = L(G)$;

(ii) G_U não tem símbolos inúteis.

Prova

Construa G_T a partir de G. Elimine de G_T as variáveis que não são atingíveis de S, e consequentemente as regras que as contêm, assim obtendo G_U.

Exemplo, continuação

$$G: \begin{cases} S \to AC \mid BS \mid B \\ A \to aA \mid aF \\ B \to CF \mid b \\ C \to cC \mid D \\ D \to aD \mid BD \mid C \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

$$G_T: \begin{cases} S \to BS \mid B \\ A \to aA \mid aF \\ B \to b \\ E \to aA \mid BSA \\ F \to bB \mid b \end{cases}$$

$$G_U: \begin{cases} S \to BS \mid B \\ B \to b \end{cases}$$

Observação

A ordem de aplicação dos algoritmos é relevante. Primeiro deve-se eliminar as variáveis não geram terminais, e somente então encontrar os símbolos que não são atingíveis a partir de S.

O exemplo a seguir ilustra este fato.

Exemplo

$$G : \begin{cases} S \to a \mid AB \\ A \to b \end{cases}$$

$$G_T : \begin{cases} S \to a \\ A \to b \end{cases} \quad \Rightarrow \quad G_U : \{ \; S \to a$$

$$G'_T : \begin{cases} S \to a \mid AB \\ A \to b \end{cases} \quad \Rightarrow \quad G'_U : \begin{cases} S \to a \\ A \to b \end{cases}$$

Licença

Slides feitos em LaTeX usando beamer e tikz, editados com vim.

Licença

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`