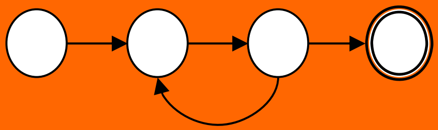
Automata e Linguagens Formais

CTC 34



6

Prof. Carlos H. C. Ribeiro
carlos@ita.br

Análise Sintática (*Parsing)*
GLCs ambíguas
Grafos de GLCs
Estratégias para *parsing*
Exemplos de *parsers*

Análise Sintática (*Parsing)*

Derivações em uma GLC: mecanismo para gerar cadeias de uma linguagem livre de contexto. Ok, mas...

Como determinar se uma dada cadeia pode ser gerada por uma dada gramática?

Em outras palavras:

Como determinar se uma dada cadeia é sintaticamente correta (ou seja, se está de acordo com a sintaxe definida pela gramática)?

Nosso objetivo: determinar algoritmo para produzir derivações das cadeias de uma linguagem de uma gramática dada. Se a cadeia não estiver na linguagem, o algoritmo deve descobrir que não existe derivação capaz de produzí-la.

Este algoritmo é conhecido como **analisador sintático** ou ***parser*.**

Parsers são usualmente variações de algoritmos de percurso em grafos.

Derivações à Direita e à Esquerda (revisão)

Derivação à Direita: cada passo de derivação aplicado à variável mais à direita.

Derivação à Esquerda: cada passo de derivação aplicado à variável mais à esquerda.

Exemplo:

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

P: S→aAS |a | b

A→SbA |SS| ba

À direita: S =>aAS=>aAb=> aSbAb=> aSbbab=> aabbab

À esquerda: S =>aAS=>aSbAS=>aabAS=> aabbaS=>aabbab

Um Teorema Óbvio

Seja G=(V, Σ, P,S) uma gramática livre de contexto.Uma sentença *w* está em L(G) sss existe uma derivação à esquerda de *w* a partir de S.

a) Para qualquer <u>sentença</u> *w*, existe uma derivação à esquerda. **Óbvio.**

b) Se existe derivação à esquerda de *w*, então *w* está em L(G). **Óbvio ululante**

Exemplo:

G: $S \rightarrow AB$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

Gera $L(G)=a^*b^*$. Para qualquer <u>sentença</u> de L(G), existe uma derivação à esquerda.

Por outro lado, **não há derivação à esquerda** para a <u>forma sentencial</u> A... Mas **existe derivação à direita!**

Existe um teorema idêntico para derivações à direita. Dada a dualidade, **vamos a partir de agora nos** concentrar apenas em derivações à esquerda.

Ambigüidade em GLCs

Restringir a atenção a derivações à esquerda é suficiente para estabelecer uma <u>derivação canônica para qualquer cadeia de linguagem</u> de uma gramática?

Infelizmente, não...

Exemplo

G:

$S \rightarrow AA$

$A \rightarrow AAA \mid bA \mid Ab \mid a$

$$S \underset{L}{\Rightarrow} AA \underset{L}{\Rightarrow} aA \underset{L}{\Rightarrow} aAAA \underset{L}{\Rightarrow} abAAA \underset{L}{\Rightarrow} abaAA \underset{L}{\Rightarrow} ababAA \underset{L}{\Rightarrow} ababaA \underset{L}{\Rightarrow} ababaa$$

$$S \underset{L}{\Rightarrow} AA \underset{L}{\Rightarrow} AAAA \underset{L}{\Rightarrow} aAAA \underset{L}{\Rightarrow} abAAA \underset{L}{\Rightarrow} abaAA \underset{L}{\Rightarrow} ababAA \underset{L}{\Rightarrow} ababaa$$

Ao menos duas derivações à esquerda para uma dada cadeia: **gramática ambígua**. É o mesmo conceito de ambigüidade em linguagens naturais.

Exemplo: *João ganhou um livro de Jorge Amado.*

Ambigüidade em GLCs

Def: Uma GLC é ambígua se existir alguma cadeia *w* em L (G) derivada por duas derivações à esquerda distintas.

Exemplo

G: S→ aS | Sa | a

Tem L(G) = a^+ e duas derivações à esquerda distintas para aa (quais?).

G: S→ aS | a

Tem L(G) = a^+ e é não-ambígua.

Observe portanto que a ambigüidade é **propriedade da gramática** e não da linguagem.

Pergunta: Dada uma linguagem livre de contexto, é sempre possível achar uma GLC não-ambígua que a gere?

Resposta: **Não!**

Grafos de GLCs e *Parsing*

Seja G = (V,Σ,P,S) uma GLC. O grafo da gramática G, g(G), é o grafo direcionado (N,P,A) onde:

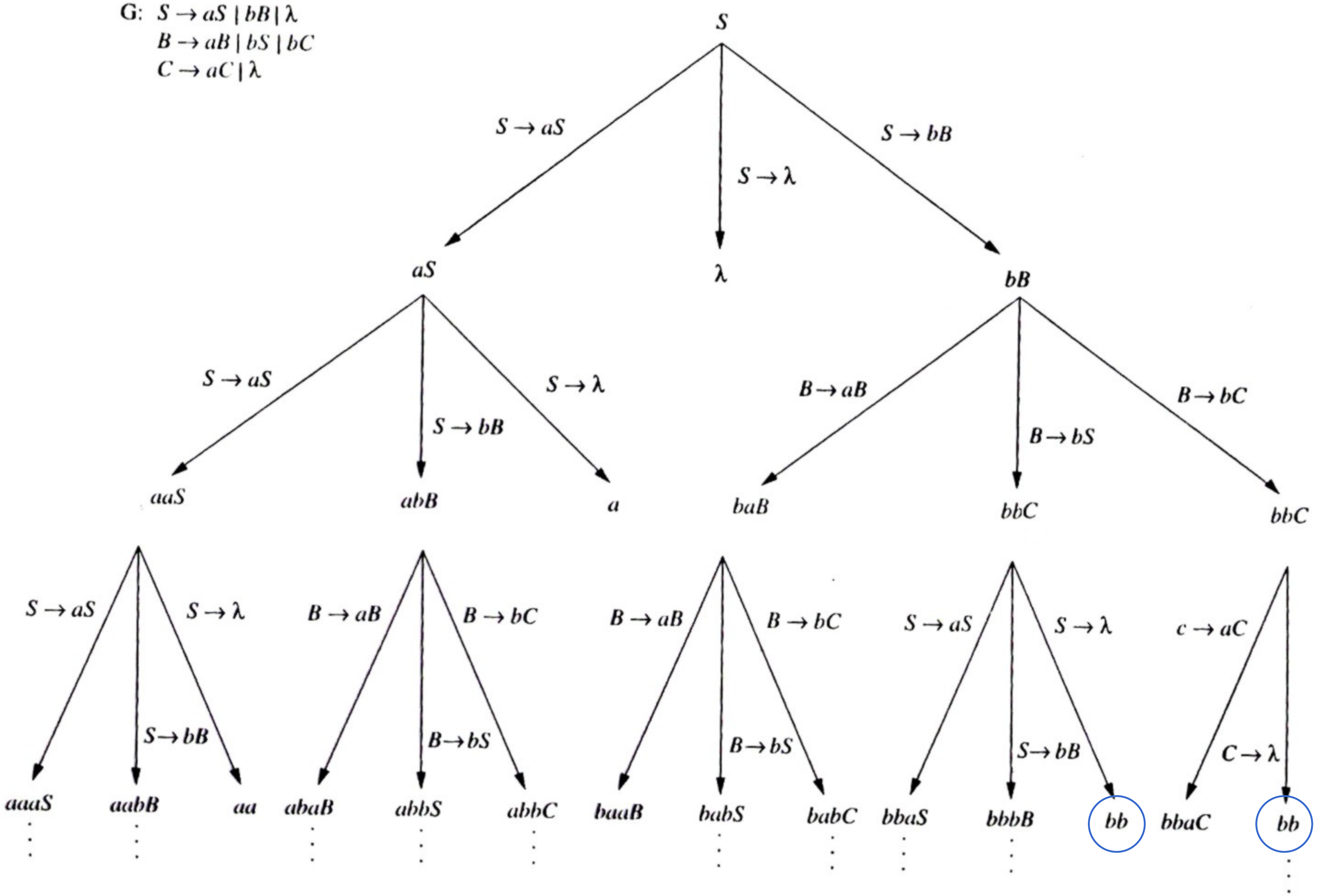
- **N = $\{w \in (V \cup \Sigma)^* \mid S \overset{*}{\underset{L}{\Rightarrow}} w\}$**
- **A = $\{[v,w, r] \in N \times N \times P \mid v \underset{L}{\Rightarrow} w$ por aplicação da regra r$\}$**

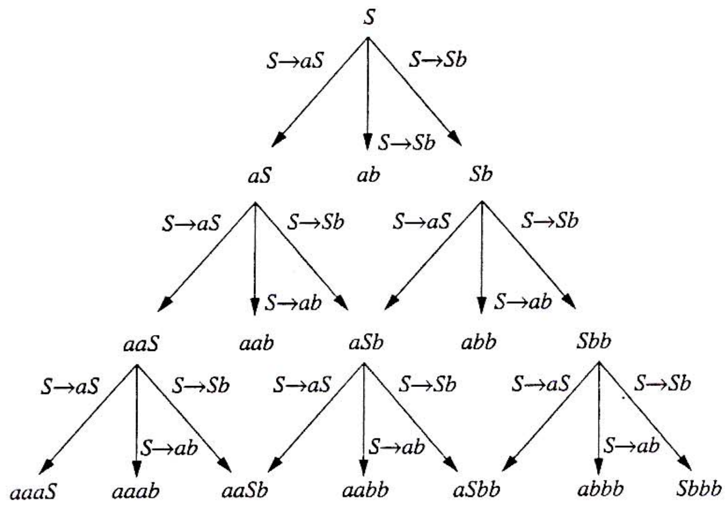
Cada caminho de S a *w* em g(G) representa uma derivação à esquerda para *w.*

O rótulo no arco de v a *w* indica a regra utilizada para obter *w* a partir de v.

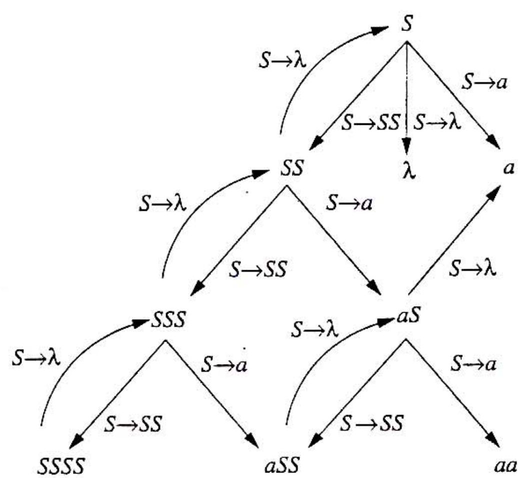
Decidir se uma dada cadeia pode ser gerada por G (ou seja, se está de acordo com a sintaxe da linguagem de G) equivale a <u>achar um caminho de S a *w*</u> no grafo g (G).

G: $S \to aS \mid bB \mid \lambda$
$B \to aB \mid bS \mid bC$
$C \to aC \mid \lambda$





(a)



(b)

FIGURE 4.2 Graphs (a) $S \to aS \mid Sb \mid ab$. (b) $S \to SS \mid a \mid \lambda$.

Análise Sintática

Idéia para a análise Sintática: Expansão do grafo da GLC.

Estratégias:

- ***Top-down:*** começo com nó S e tento achar caminho para cadeia w.
- ***Bottom-up:*** começo com cadeia w e tento chegar no nó inicial S.

Processo não-determinista: para construir uma derivação, em geral existem várias possíveis regras a serem aplicadas à forma sentencial.

Um Analisador Sintático *Breadth-First, Top-Down*

Top-down: derivações à esquerda a partir de S.

Prefixo da foram sentencial *uVw:u.*

*Breadth-first:*busca completa (garante *parsing* caso a cadeia seja da gramática).

Algoritmo

- Inicializo uma fila com S.
- Loop: **Removo nó do início da fila. Tem prefixo consistente com cadeia a ser analisada?**
    - Sim: Expando o nó do inicio da fila. Cada filho é uma derivação à esquerda possível, rotulando pela cadeia resultante da derivação. Coloco os nós correspondentes no **final da fila.**
    - Não: Apenas removo o nó do início da fila.

  Até que o nó do início da fila seja igual a cadeia analisada ou fila fique vazia.

Exemplo: parsing de (b+b) para gramática

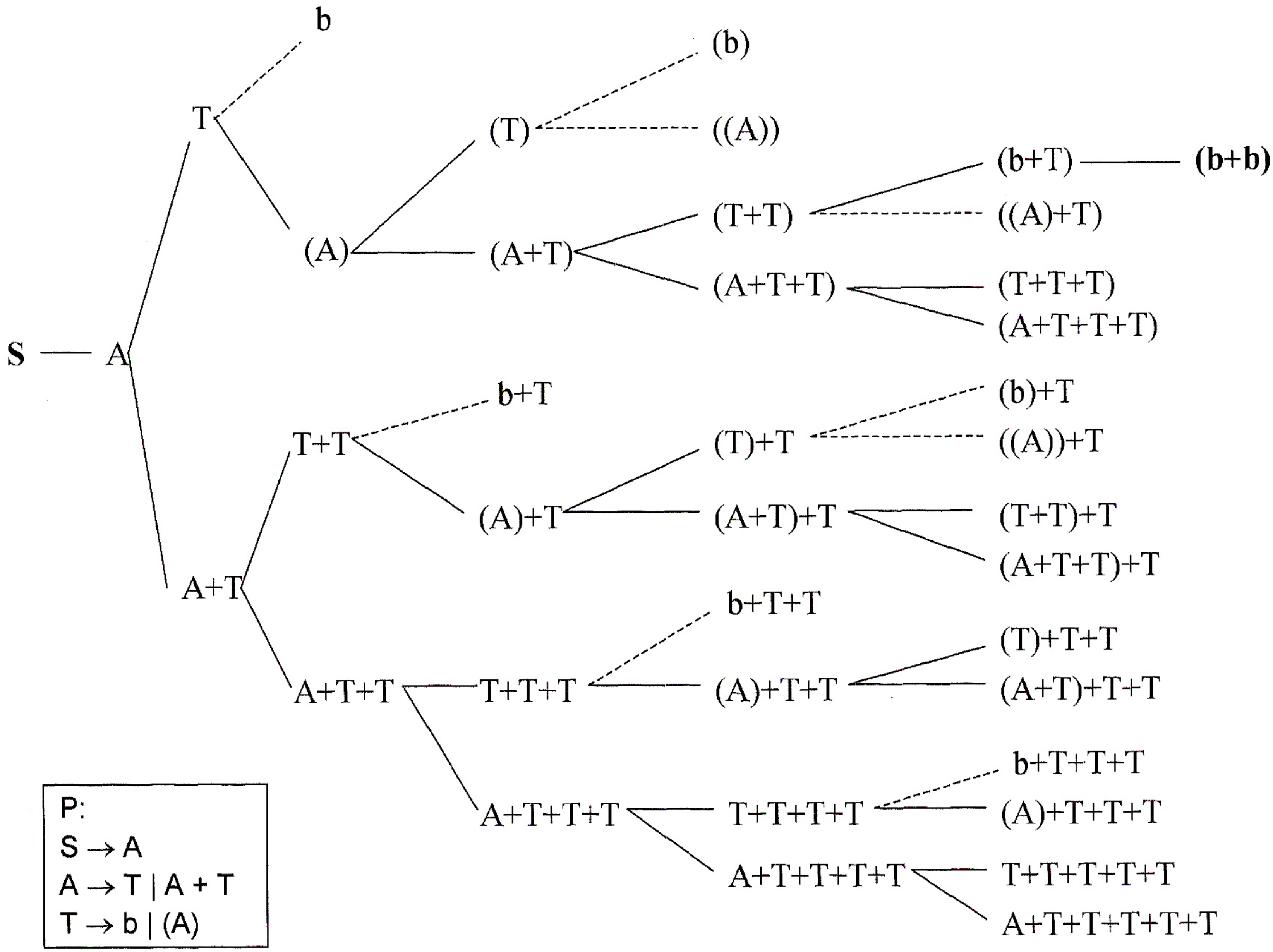
G: V= {S, A, T}

Σ={ b,+, (,)}

P: S→A

A→T | A + T

T →b | (A)



P:

$S \rightarrow A$

$A \rightarrow T \mid A + T$

$T \rightarrow b \mid (A)$

Um Analisador Sintático *Depth-first, Top-down*

Top-down: derivações à esquerda a partir de S.

Algoritmo

- Inicializo uma pilha com S.
- Loop:

  - Nó do topo da pilha tem prefixo consistente com cadeia a ser analisada?
    - Sim: Removo o espaço o nó do topo da pilha. Cada filho é uma derivação à esquerda possível, rotulado pela cadeia resultante da derivação. Coloco os nós correspondentes no **topo da pilha**.
    - Não: Removo o nó do topo da pilha.

  Até que nó do topo da pilha seja igual a cadeia analisada ou pilha fique vazia.

Exemplo: parsing de (b+b) para a gramática

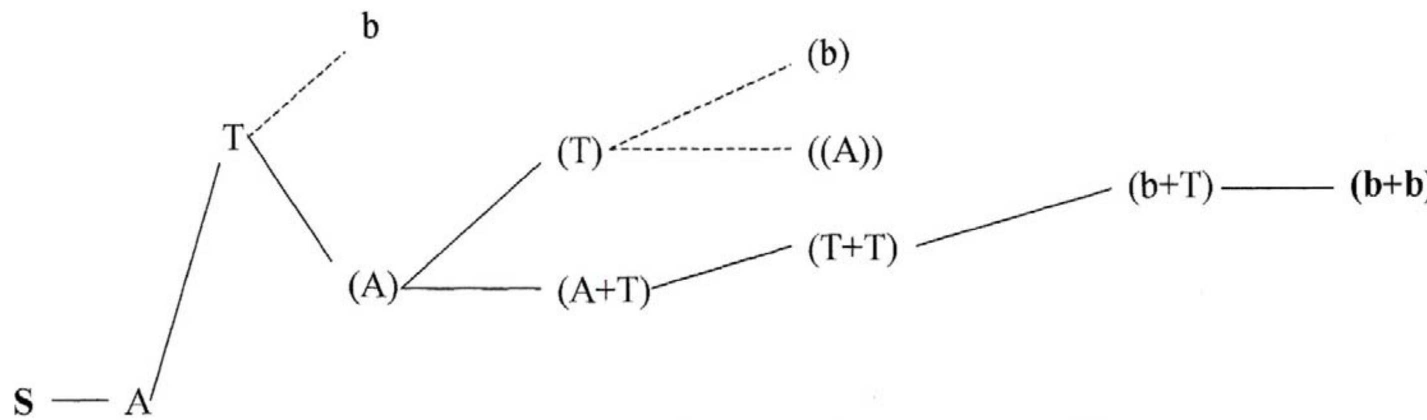
G: V = {S, A, T}

Σ = {b, +, (,)}

P: S → A

A → T | A + T

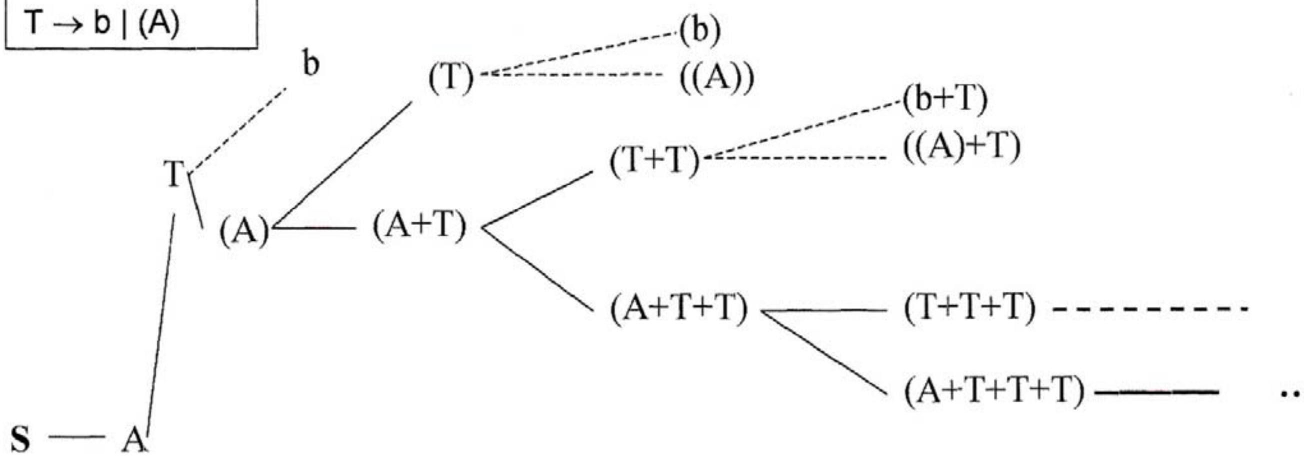
T → b | (A)



Vantagem: muito mais econômico do que *breadth-first* → mantém menos nós na memória.

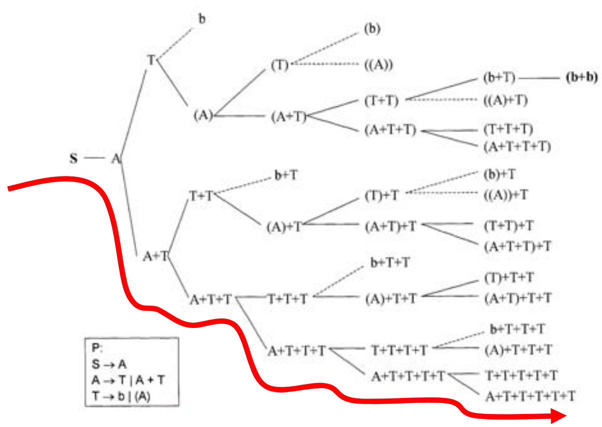
P:
S → A
A → T | A + T
T → b | (A)

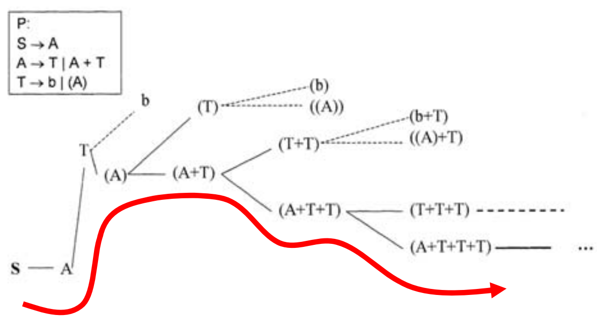
Desvantagem: risco de loops infinitos. Exemplo: *parsing* de (b) + b



Eliminação de recursão direta à esquerda

- Observe um problema comum na análise top-down:





$$A \rightarrow A+T$$

Recursão direta à esquerda: possibilidade de não-terminação da análise sintática!

Eliminação de recursão direta à esquerda: Exemplos

G: $A \rightarrow Aa \mid b$	$L(G)=ba^*$
G': $A \rightarrow bZ \mid b$	*(geração do símbolo não-recursivo)*
$Z \rightarrow aZ \mid a$	*(recursão)*

G: $A \rightarrow Aa \mid Ab \mid b \mid c$	$L(G)=(b \cup c)(a \cup b)^*$
G': $A \rightarrow bZ \mid cZ \mid b \mid c$	*(geração de símbolos não-recursivos)*
$Z \rightarrow aZ \mid bZ \mid a \mid b$	(recursão)

Eliminação de recursão direta à esquerda:
Um Teorema

Seja $G=(V,\Sigma,P,S)$ uma GLC e seja $A \in V$ uma variável recursiva diretamente à esquerda em G. Existe uma GLC G' (e algoritmo correspondente para produzí-la) que satisfaz:

i) $L(G')=L(G)$.
ii) A em G' não é variável recursiva diretamente à esquerda

Eliminação de recursão direta à esquerda: Algoritmo

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

$A \in V$ uma variável recursiva diretamente à esquerda.

1. Divida o conjunto de regras de A em:

$$A \rightarrow Au_1 \mid Au_2 \mid \ldots \mid Au_j$$
$$A \rightarrow v_1 \mid v_2 \mid \ldots \mid v_k$$

2. Defina novas regras A sem recursão sobre A para geração inicial dos v_i , $1 \leq i \leq k$:

$$A \rightarrow v_1 \mid v_2 \mid \ldots \mid v_k \mid v_1Z \mid \ldots \mid v_kZ$$

3. Use recursão sobre variável auxiliar Z para gerar os símbolos u_i, $1 \leq i \leq j$:

$$Z \rightarrow u_1Z \mid u_2Z \mid \ldots \mid u_jZ \mid u_1 \mid \ldots \mid u_j$$

4. Remova as regras de A originais.

Eliminação de recursão direta à esquerda: Algoritmo (Exemplo)

G: $A \rightarrow Aa \mid Aab \mid bb \mid b$

Variáveis com recursão direta à esquerda: A

Regras com recursão direta à esquerda: $A \rightarrow Aa \mid Aab$
Regras sem recursão: $A \rightarrow bb \mid b$

Eliminação da recursão sobre A: $A \rightarrow bbZ \mid bZ \mid bb \mid b$
Recursão sobre variável Z: $Z \rightarrow aZ \mid abZ \mid a \mid ab$

G': $A \rightarrow bbZ \mid bZ \mid bb \mid b$
$Z \rightarrow aZ \mid abZ \mid a \mid ab$

Infelizmente, também pode haver recursão indireta à esquerda:

$A \rightarrow Bu$
$B \rightarrow Av$

$A \Rightarrow Bu \Rightarrow Avu \Rightarrow Buvu \Rightarrow Avuvu \Rightarrow \ldots$

Observe que o problema é a existência permanente de um prefixo não-terminal: enquanto existir, não posso – na análise sintática *top-down* – identificar prefixos incompatíveis com o começo da sentença analisada…

SOLUÇÃO: FORMA NORMAL DE GREIBACH !!!

Forma Normal de Greibach (revisão)

- *Def.:* Uma GLC G está na forma normal de Greibach se cada regra estiver em uma das seguintes formas: $A \to aA_1A_2\ldots A_n$, $A \to a$, $A \to \varepsilon$

Seja $G=(V,\Sigma,P,S)$ uma GLC na FNC. Existe GLC G' (e algoritmo correspondente para produzí-la) que satisfaz:

i) L(G')=L(G).
ii) G' está na forma normal de Greibach

Sheila Greibach

Observe que $A \to aA_1A_2\ldots A_n$, $A \to a$, $A \to \varepsilon$

Sempre garante que a aplicação de qualquer regra (a menos de $A \to \varepsilon$) aumenta o tamanho do prefixo de terminais da forma sentencial!

Geração de Gramática na Forma Normal de Greibach: Algoritmo

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

1. Associe ao símbolo inicial S o índice 1 e indexe (i=2,3,…) arbitrariamente os demais símbolos não-terminais.

2. Construa uma gramática intermediária G_i equivalente a G, em que cada produção está em uma das formas $S \to \varepsilon$, $A \to aw$ ou $A \to Bw$, com $w \in V^*$ e índice de B maior do que o índice de A. Gere esta gramática usando alternadamente o algoritmo de eliminação de recursão direta à esquerda e o Lema de Substituição do quadro abaixo.

3. Use o Lema de Substituição para gerar a gramática na Forma Normal de Greibach.

Lema de Substituição

Seja $G = (V, \Sigma, P, S)$. Seja $A \to uBv$ uma produção em P e $B \to w_1 \mid w_2 \mid \ldots \mid w_n$ as produções a partir de B em P.
A gramática $G' = (V, \Sigma, P', S)$ em que $P' = (P - \{A \to uBv\}) \cup \{A \to uw_1v \mid uw_2v \mid \ldots \mid uw_nv\}$ é equivalente a G.

Geração de Gramática na Forma Normal de Greibach: Exemplo

Sudkamp, págs. 141-144

Análise Sintática *Bottom-Up*

Idéia: realizar busca no grafo a partir da cadeia a ser analisada (geração de caminhos na direção da raiz). Como as únicas derivações consideradas são as que podem gerar a cadeia, o tamanho da árvore expandida tende a ser menor.

O processo de geração das formas sentenciais em níveis cada vez mais altos da árvore é conhecido como **redução**.

Exemplo: Redução de (b) + b para gramática

G: V = { S, A, T}

Σ = {b, +, (,)}

P: S → A

A → T | A + T

T → b | (A)

Redução	Regra
(b) + b	
(T)+ b	T→b
(A)+ b	A→T
T+ b	T→(A)
A+ b	A→T
A+ T	T→b
A	A→A+T
S	S→A

Análise Sintática *Bottom-Up:* Geração Automática das Reduções

- **Algoritmo:**

1. Escrevo $w=uv$ (na primeira interação, $u=\varepsilon$, $v=w$).
   Para cada regra:
2. Comparo lado direito das regras com sufixos de u:
   $u=u_1 q$ e $A \rightarrow q \in P$? Reduzo w a $u_1 A v$
3. Volto a 1, fazendo w=u'v' tal que u'é u concatenado ao primeiro elemento de v, e v'é v sem o seu primeiro elemento (processo de ***shift)***

Exemplo:

Redução de (A+T) para gramática

G: V = {S,A,T}

Σ = {b,+, (,)}

P: S → A

A → T | A + T

T → b | (A)

	u	**v**	**regra**	**redução**
	ε	(A+T)		
Shift	(	A+T)		
Shift	(A	+T)	S→A	(S+T)
Shift	(A+	T)		
Shift	(A+T	)	A→A+T	(A)
			A→T	(A+A)
shift	A+T)	ε		

Um Analisador Sintático *Breadth-First, Bottom-Up*

Bottom-up: derivações à **direita** a partir de S, pois as reduções são feitas **à esquerda** a partir da cadeia terminal..

*Breadth- first:*busca completa (garante *parsing* caso a cadeia seja da gramática).

Algoritmo

- Inicializo uma fila com cadeia terminal.
- Loop:
  - **Removo nó do início da fila. Existe redução de uwv deste nó para uAv com v formado só por símbolos terminais?**
    - Sim: Expando o nó do inicio da fila. Cada filho é uma redução à esquerda possível, rotulando pela cadeia resultante da redução. Coloco os nós correspondentes no **final da fila.**
    - Não: Apenas removo o nó.

  Até que o nó do início da fila seja igual a S ou fila fique fazia.

Exemplo: parsing de (b+b) para gramática

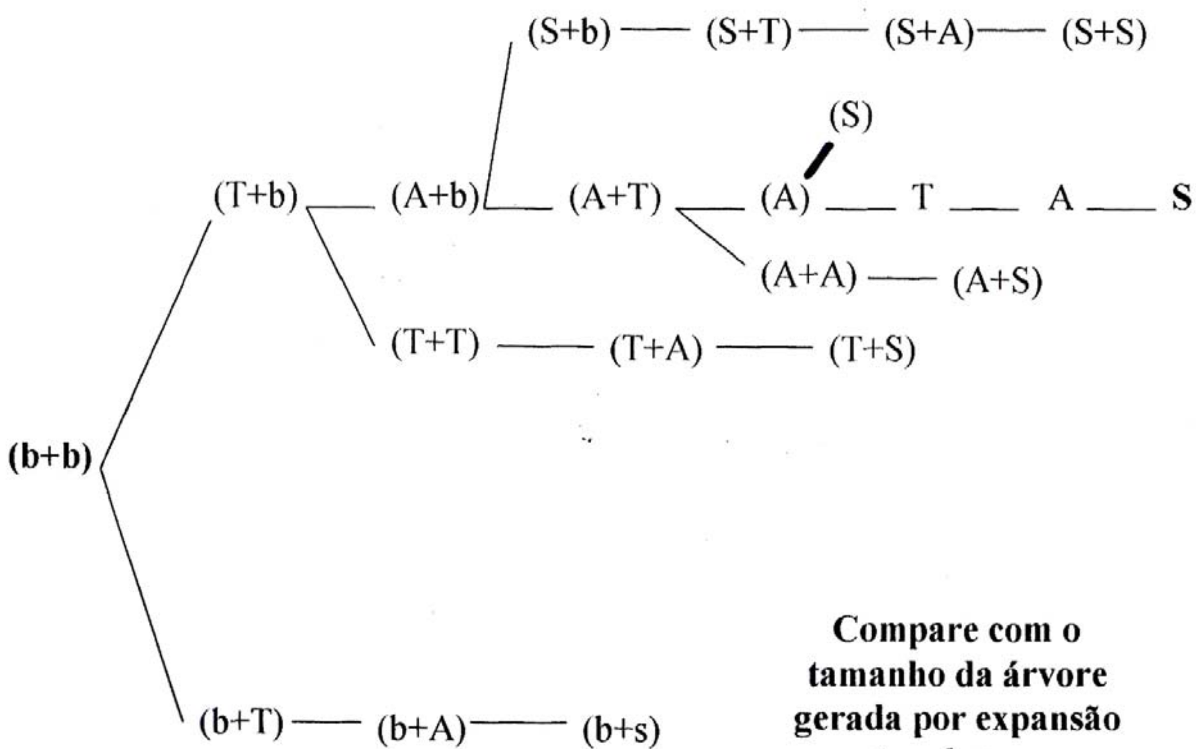
G: V= {S, A, T}

Σ={ b,+, (,)}

P: S→A

A→T | A + T

T →b | (A)



Compare com o tamanho da árvore gerada por expansão *top-down*...

P:
$S \to A$
$A \to T \mid A + T$
$T \to b \mid (A)$

Variações

- Técnicas de busca:Aumento da eficiência computacional
  - aprofundamento iterativo (depth-first com aumento gradual do nível de profundidade).
  - busca bidirecional (top-down e bottom-up simultâneos)
  - etc,etc,...
- Heurísticas: Diminuem o espaço de busca.
  - Análise a priori da cadeia (uso de contadores, etc.): por exemplo, formas sentenciais com mais símbolos terminais do que a sentença analisada certamente não levam a *parsing.*
- Normalização de GLCs:Normalização Greibach garante completeza da análise sintática *depth-first top-down*..
- Particularização do processo de *parsing* para classes particulares de GLCs: por exemplo, a classe LL (k) permite *parsing top-down* **determinista**, desde que se utilize o conceito de *look-ahead.*

  Exemplo: *uAv* obtido durante o *parsing de p=uaw.* Ao invés de considerar apenas o prefixo *u,* olho mais adiante (*look-ahead)* e analizo regras *A.* Aquelas cuja produção não começa com a podem ser consideradas.

  O determinismo surge naturalmente: a classe LL (1) obriga uma única regra aplicável se for usado look-ahead de 1 símbolo, a classe LL(2) obriga uma única regra se for usado look-ahead de 2 símbolos, etc.