Linguagens Livres de Contexto

Motivação - Exemplo

pal = { $x \in \{0,1\}^* \mid x = x^R$ }

= { ε,

0,1,

00,11,

000,010, 101,111,

0000,0110,1001,1111, … }

Vimos que **pal** não é regular.

Mas existe um padrão p/ construção dos strings.

Q: Se você tem um palíndromo, como pode gerar outro a partir desse?

Motivação - Exemplo

pal = { $x \in \{0,1\}^*$ | $x = x^R$ }

R: Podemos gerar **pal** recursivamente como a seguir.

CASO BASE: ε, 0 e 1 são palíndromos.

RECURSIVO: se *x* é um palíndromo, então 0*x* 0 e 1*x* 1 também são.

Motivação - Exemplo

pal = { $x \in \{0,1\}^*$ | $x = x^R$ }

RESUMO: Qualquer $x \in$ **pal** pode ser usado p/ obter novo string de **pal**

NOTAÇÃO: $x \rightarrow \varepsilon|0|1|0x\,0|1x\,1$

Cada "|" é simplesmente "ou", como em expressões regulares UNIX.

Q: Como você geraria 11011011 a partir da variável x ?

Motivação - Exemplo

A: Gerando um string de fora para dentro:

11011011 = 1(1(0(1()1)0)1)1

de modo que:

$x \Rightarrow 1x1 \Rightarrow 11x11 \Rightarrow 110x\,011$

$\Rightarrow 1101x1011 \Rightarrow 1101\varepsilon 1011 =$

11011011

Gramática Livre de Contexto. Definição

DEF: Uma ***gramática livre de contexto*** consiste de

$G=(V, \Sigma, R, S)$ onde:

- V – conjunto finito de ***variáveis*** (ou ***símbolos não terminais***)
- Σ – conjunto finito de ***terminais*** (ou ***alfabeto***)
- R – conjunto finito de ***regras*** (ou ***produções***) da forma $v \rightarrow w$ onde $v \in V$, e $w \in (\Sigma \cup V)^*$

  (lê-se "*v* ***produz*** *w* ")
- $S \in V$ –***símbolo inicial***.

Q: O que é (V, Σ, R, S) para **pal** ?

Gramática Livre de Contexto. Definição

R: $V = \{x\}$

$\Sigma = \{0,1\}$,

$R = \{x \rightarrow \varepsilon,\ x \rightarrow 0,\ x \rightarrow 1,\ x \rightarrow 0x\ 0,\ x \rightarrow 1x\ 1\}$

$S = x$

Adotando a notação padrão - **pal**:

$V = \{S\ \}$, $S = S$

$R = \{S \rightarrow \varepsilon,\ S \rightarrow 0,\ S \rightarrow 1,\ S \rightarrow 0S\ 0,\ S \rightarrow 1S\ 1\}$

Derivações

DEF: O ***símbolo de derivação*** "$\Rightarrow$", (lê-se "deriva em 1-passo" ou "produz em 1-passo") denota uma relação entre strings em $(\Sigma \cup V)^*$. Escrevemos $x \Rightarrow y$ se x e y podem ser divididos como $x = svt$ e $y = swt$, onde $v \rightarrow w$ é uma regra de R

Q: Que possíveis y satisfazem (em **pal**)

$S\,0000S \Rightarrow y$?

Derivações

R: $S\ 0000S \Rightarrow y$:

Qualquer dos seguintes:

0000*S*, 00000*S*, 10000*S*, 0*S*00000*S*, 1*S*10000*S*, *S*0000, *S*00000, *S*00001, *S*00000*S*0, *S*00001*S*1

Derivações

DEF: O ***símbolo de derivação*** "$\Rightarrow$*"

(lê-se "deriva" ou "produz") denota uma relação entre strings de $(\Sigma \cup V)$*. Escrevemos $x \Rightarrow^* y$ se existe uma sequência de derivações em 1-passo de x para y. I.e., existem strings x_i , i variando de 0 a n, tais que $x = x_0$, $y = x_n$ e

$$x_0 \Rightarrow x_1, x_1 \Rightarrow x_2, x_2 \Rightarrow x_3, \ldots, x_{n-1} \Rightarrow x_n$$

Q: Quais LHS's derivam quais RHS's em **pal**?

01*S*, *SS* →?→ 01, 0, 01*S*, 01110111, 0100111

Derivações

R: Não há apenas uma resposta:

- $01S \Rightarrow^* 01$
- $SS \Rightarrow^* 0$
- $01S \Rightarrow^* 01S$
- $SS \Rightarrow^* 01110111$
- *Nenhum deles deriva* 0100111

Linguagem Gerada por uma CFG

DEF: Seja *G* uma gramática livre de contexto A ***linguagem gerada*** por *G* é o conjunto de todos os strings de símbolos terminais que podem ser derivados a partir do símbolo inicial. Formalmente:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Exemplo: Expressões Aritméticas

Expressões envolvendo {+, ×, *a*, *b*, *c*, (,)}

E representa uma expressão (mais general)

F representa um fator (parte da multiplicação)

T representa um termo (produto de fatores)

V representa uma variável, i.e. *a, b,* ou *c*

Gramática é dada por:

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T \times F$

$F \rightarrow V \mid (E)$

$V \rightarrow a \mid b \mid c$

CONVENÇÃO: Variável inicial - *E*

Exemplo: Expressões Aritméticas

EX: Considere o string u =

$$a \times b + (c + (a + c))$$

Essa é uma expressão aritmética válida. Portanto deve poder ser generada de E.

1. É uma soma de duas expressões, então a primeira derivação deve ser $E \Rightarrow E + T$
2. A sub-expressão $a \times b$ é um produto, portanto um termo generado por $E + T \Rightarrow T + T \Rightarrow T \times F + T \Rightarrow^* a \times b + T$
3. A segunda sub-expressão é um fator, por causa dos parenteses: $a \times b + T \Rightarrow a \times b + F \Rightarrow a \times b + (E) \Rightarrow a \times b + (E + T)$

Exemplo: Expressões Aritméticas

Resumindo:

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow T \times F + T \Rightarrow F \times F + T \Rightarrow V \times F + T$
$\Rightarrow a \times F + T \Rightarrow a \times V + T \Rightarrow a \times b + T \Rightarrow a \times b + F \Rightarrow a \times b + (E) \Rightarrow a \times b + (E + T) \Rightarrow a \times b + (T + T) \Rightarrow a \times b + (F + T) \Rightarrow a \times b + (V + T) \Rightarrow a \times b + (c + T) \Rightarrow a \times b + (c + F) \Rightarrow a \times b + (c + (E)) \Rightarrow a \times b + (c + (E + T)) \Rightarrow a \times b + (c + (T + T)) \Rightarrow a \times b + (c + (F + T)) \Rightarrow a \times b + (c + (a + T)) \Rightarrow a \times b + (c + (a + F)) \Rightarrow a \times b + (c + (a + V)) \Rightarrow a \times b + (c + (a + c))$

Esta é dita uma ***derivação mais à esquerda*.**

Derivação mais à direita

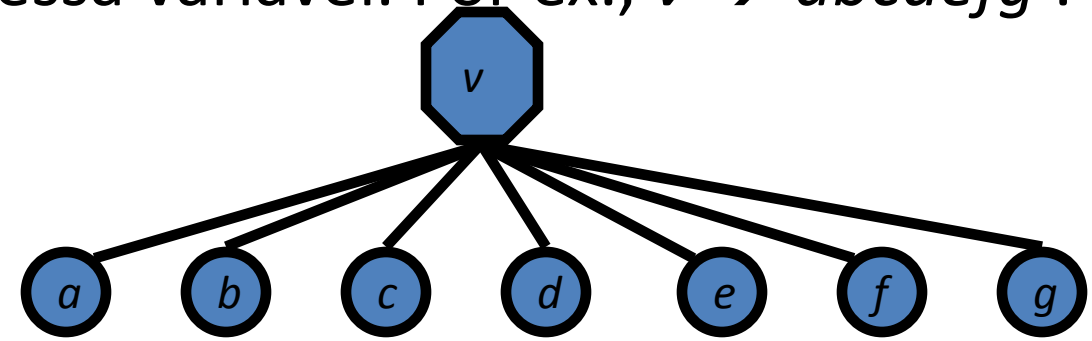
Em uma derivação mais à direita, sempre substituimos a variável mais à direita.

$E \Rightarrow E + T \Rightarrow E + F \Rightarrow E + (E) \Rightarrow E + (E + T) \Rightarrow E + (E + F) \Rightarrow$ etc.

Para abstrair dessa ambiguidade na ordem de aplicação das regras, usualmente representamos um derivação na forma de uma *árvore de derivação*.

Árvore de Derivação

Em uma ***árvore de derivação*** (ou ***parse tree***) cada nodo é um símbolo. Cada nodo-pai é uma variável cujos filhos correspondem ao lado direito de uma regra dessa variável. Por ex., $v \rightarrow abcdefg$:



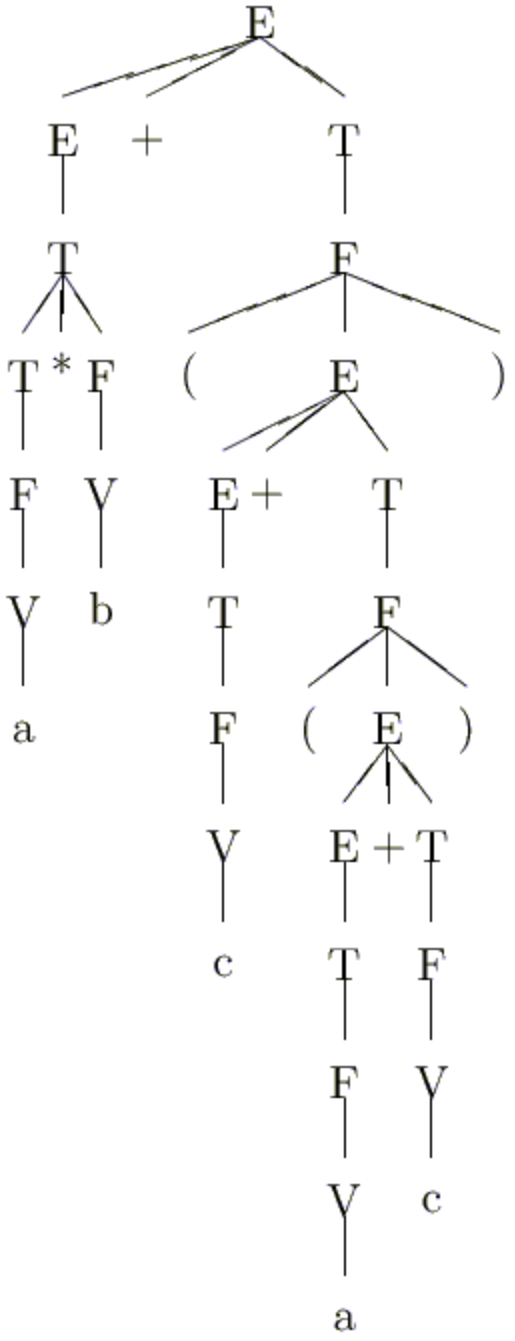
A raiz da árvore é o símbolo inicial. As folhas representam o string derivado (lido da esquerda para a direita).

Árvore de Derivação

Uma árvore de derivação para

$$a \times b + (c + (a + c))$$

Vantagem: Árvores de derivação ajudam também a entender a semântica! Podemos ver, a partir da árvore, como a expressão deve ser avaliada.



Ambiguidade

<*sentence*> → <*action*>|<*action*>com<*subject*>

<*action*> → <*subject*><*activity*>

<*subject*> → <*noun*>| <*noun*>and<*subject*>

<*activity*> → <*verb*>| <*verb*><*object*>

<*noun*> → Hannibal | Clair | arroz | cebolas

<*verb*> → comeu | jogou

<*prep*> → com | e | ou

<*object*>→ <*noun*>|<*noun*><*prep*><*object*>

- Clair jogou com Hannibal
- Clair comeu arroz com cebolas
- Hannibal comeu arroz com Clair

Q: Alguma dessas sentenças é ambígua?

Ambiguidade

A: Considere “Hannibal comeu arroz com Clair”.

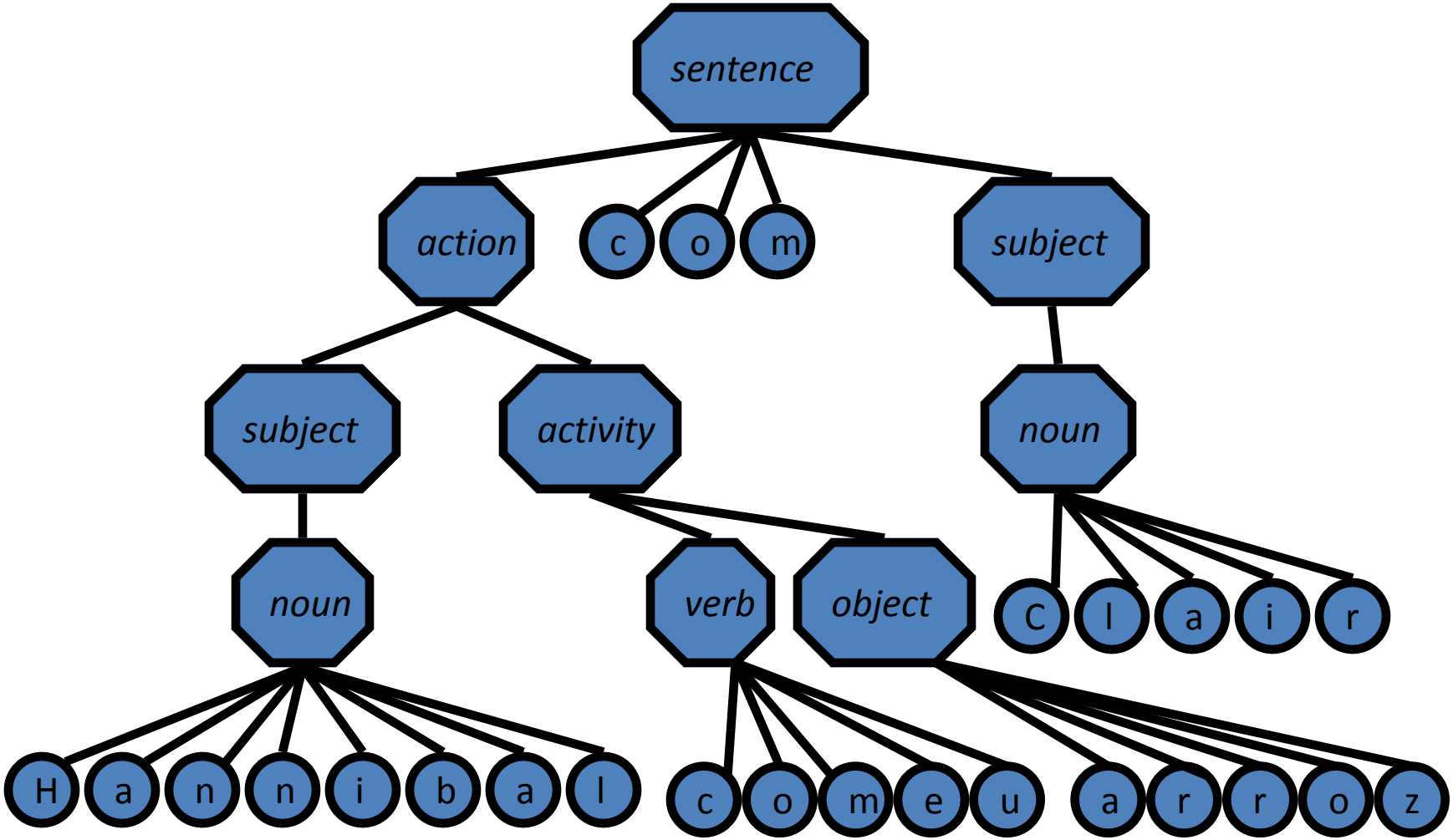
Poderia significar

- Hannibal e Clair comeram arroz juntos.
- Hannibal comeu arroz e *comeu* Clair.

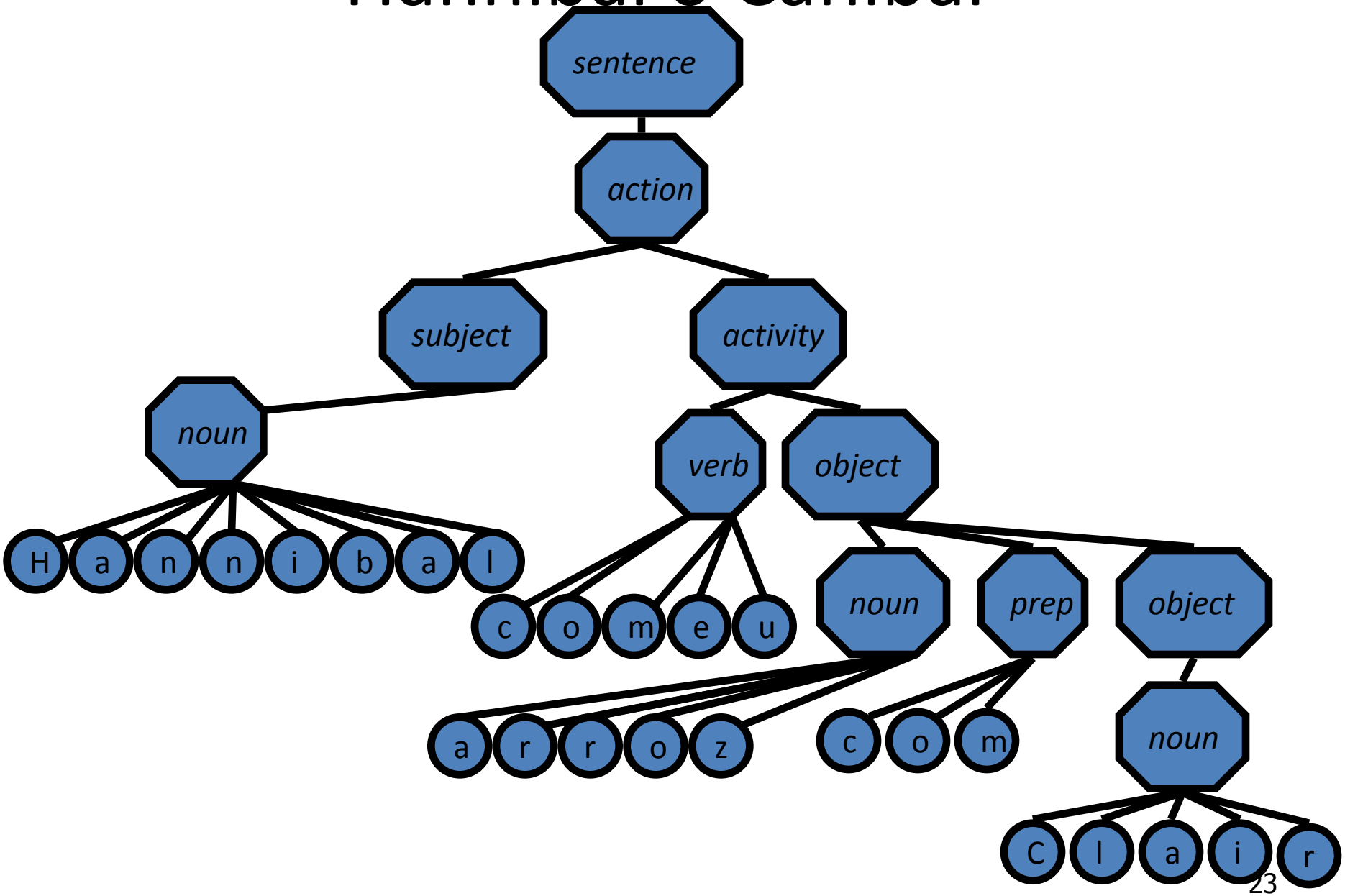
Isso não é absurdo, se sabemos quem é Hannibal!

Essa ambiguidade decorre do fato de que essa sentença tem duas árvores de derivação distintas e, portanto, duas diferentes interpretações:

Hannibal e Clair comem



Hannibal o Canibal



Ambiguidade. Definição

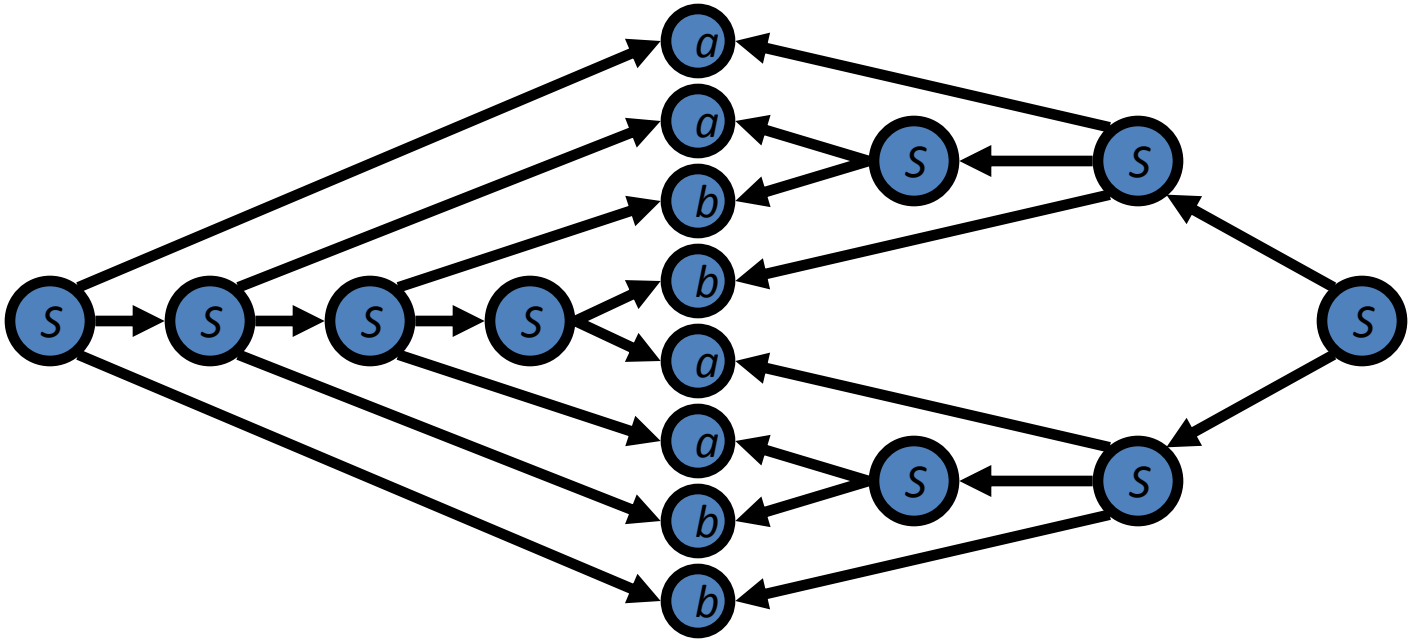
DEF: Um string *x* é dito ambíguo em relação a uma gramática *G* se existem duas maneiras ***essencialmente diferentes*** de derivar *x* em *G*. I.e. *x* admite duas (ou mais) árvores de derição distintas (ou equivalentemente, *x* admite diferentes derivações mais à esq.[dir]) Uma gramática *G* é dita ambígua se existe algum string $x \in L(G)$ que seja ambíguo.

Q: A gramática $S \rightarrow ab \mid ba \mid aSb \mid bSa \mid SS$ é ambígua? Qual é a linguagem gerada?

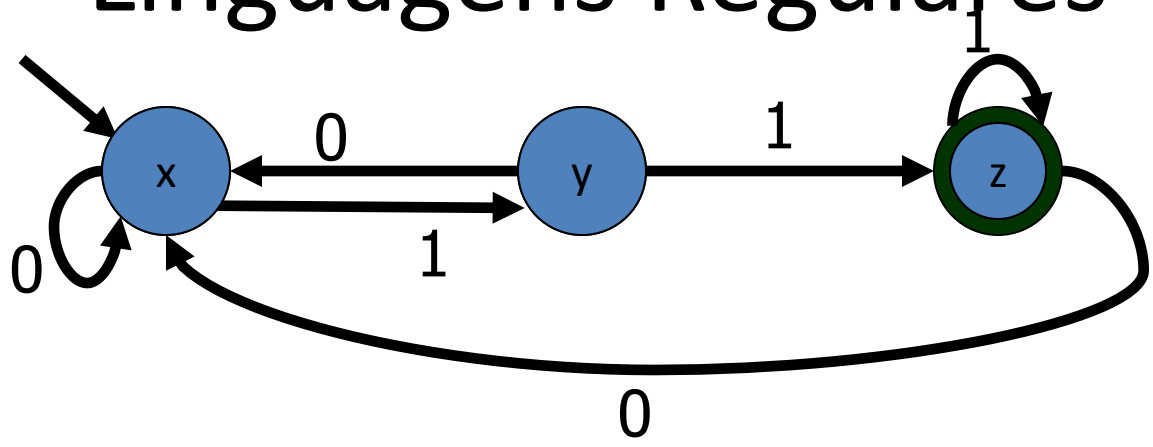
Ambiguidade

R: $L(G)$ = linguagem dos strings com igual no. de *a'* s e *b'* s

Sim, a gramática é ambígua:



Gramáticas Lineares à Direita e Linguagens Regulares



O DFA acima pode ser simulado pela gramática

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

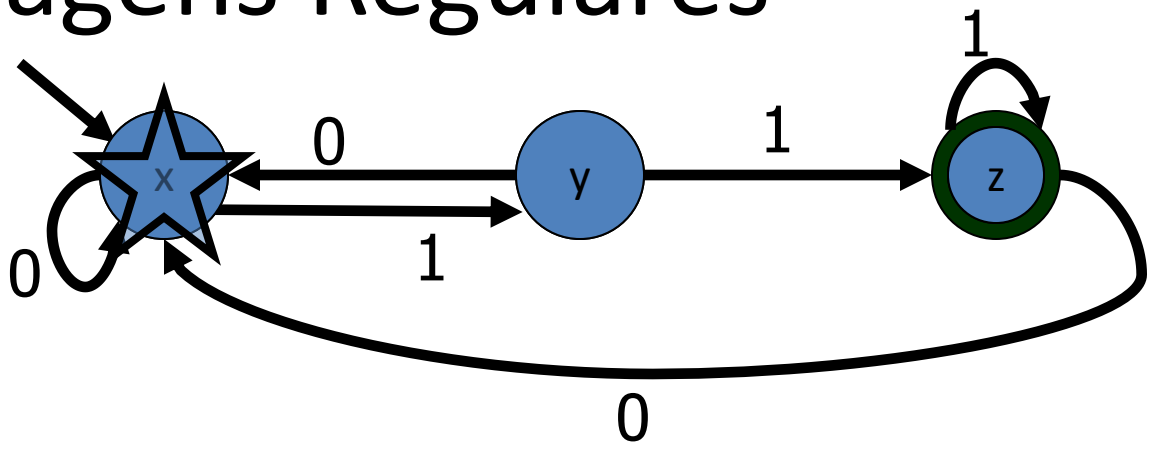
$z \rightarrow 0x \mid 1z \mid \varepsilon$

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



x

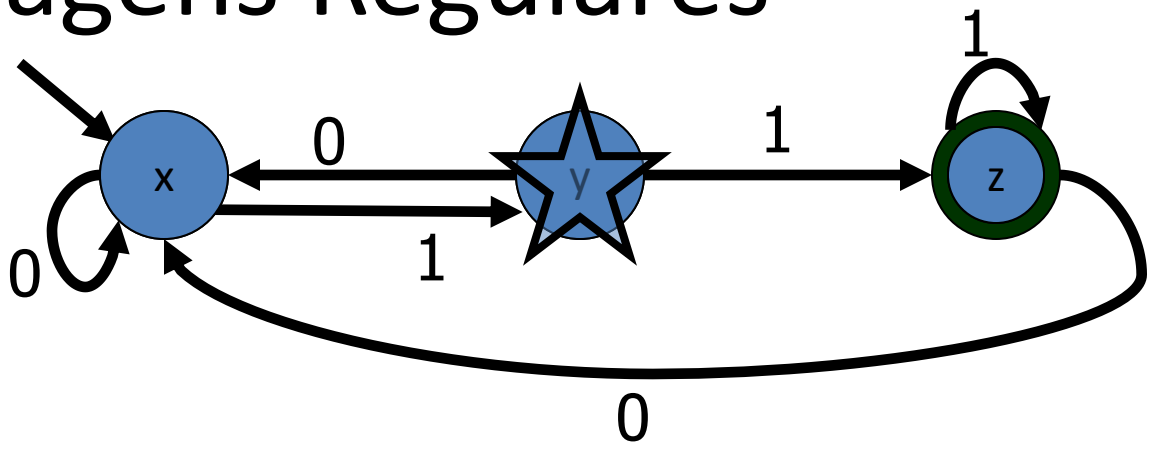
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y$

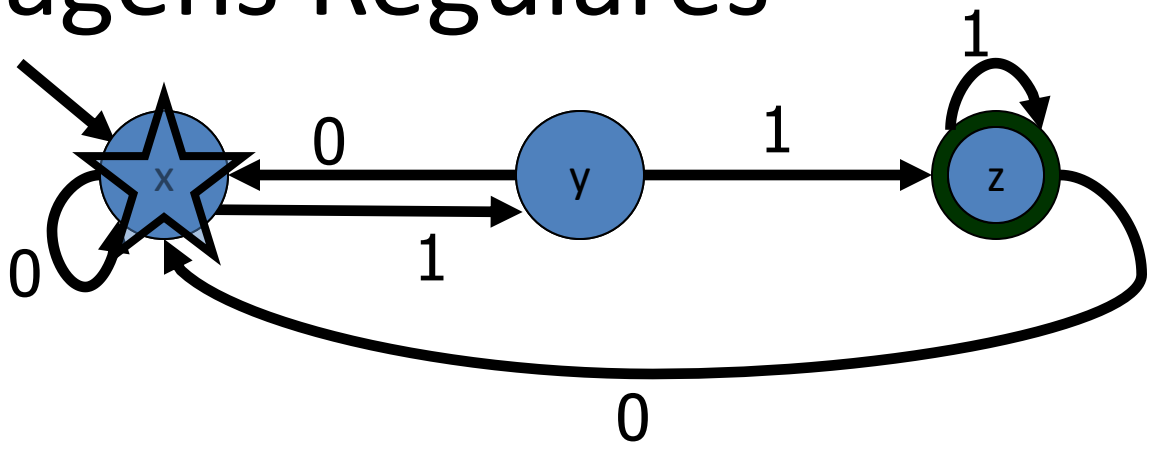
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y \Rightarrow 10x$

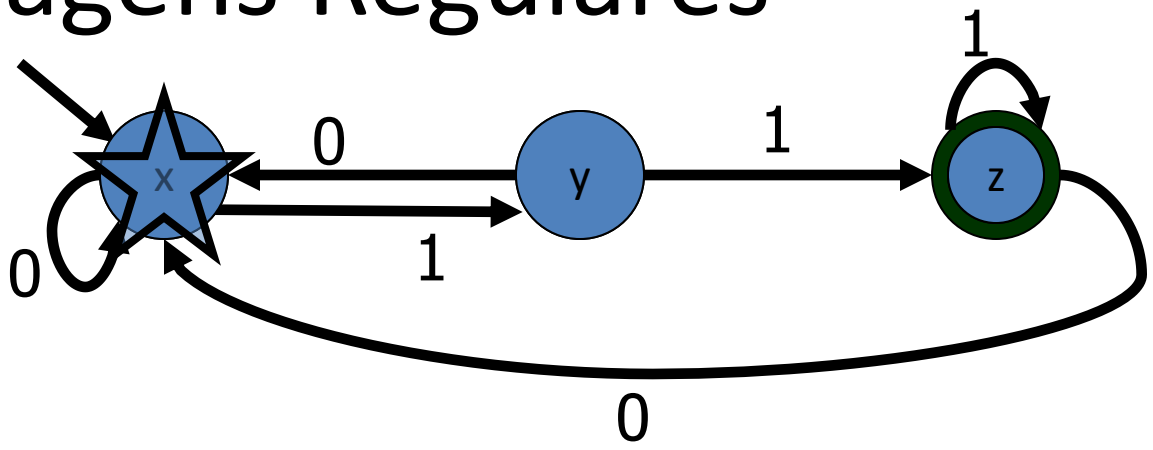
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y \Rightarrow 10x \Rightarrow 100x$

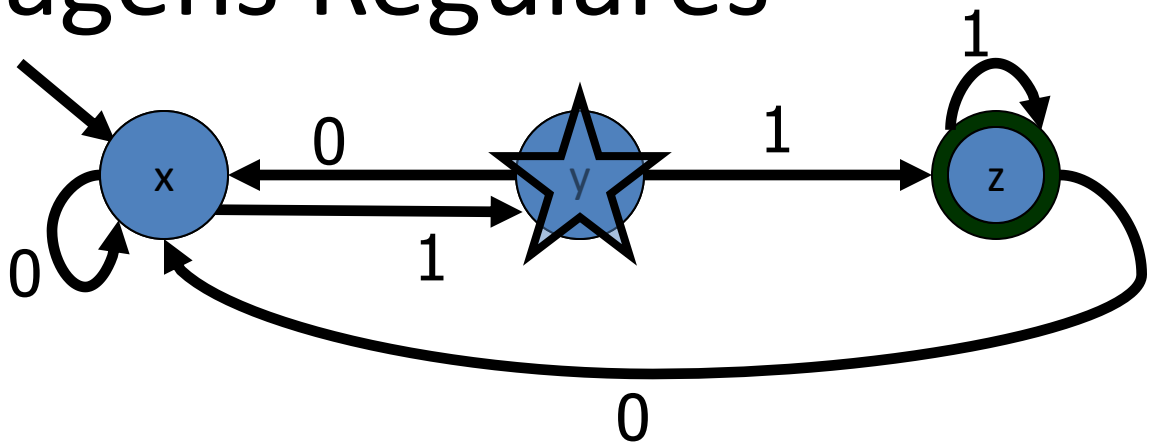
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y \Rightarrow 10x \Rightarrow 100x \Rightarrow 1001y$

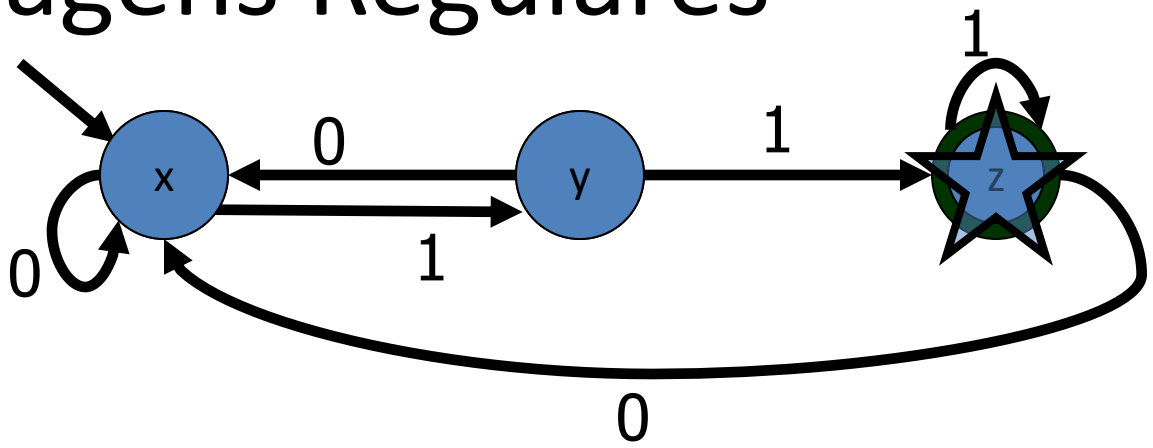
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y \Rightarrow 10x \Rightarrow 100x \Rightarrow 1001y$

$\Rightarrow 10011z$

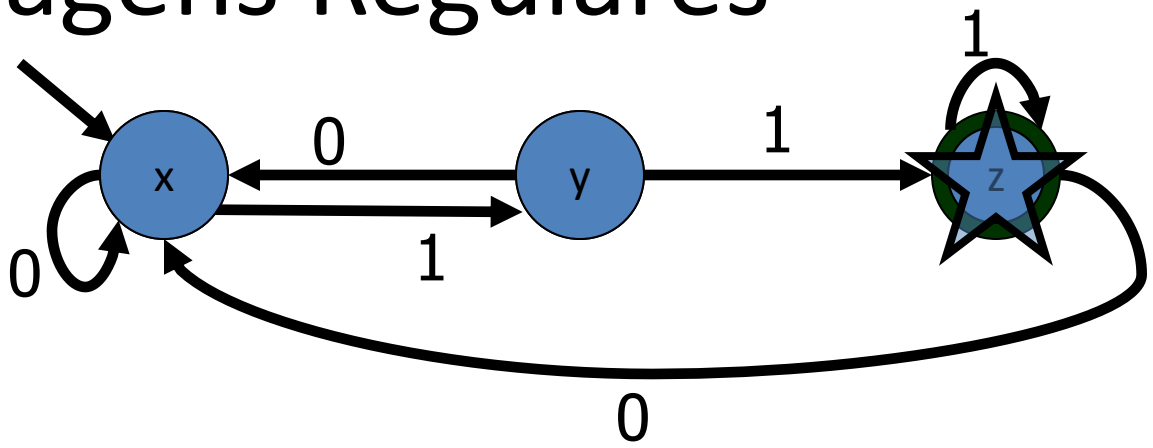
10011

Gramáticas Lineares à Direita e Linguagens Regulares

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$



$x \Rightarrow 1y \Rightarrow 10x \Rightarrow 100x \Rightarrow 1001y$

$\Rightarrow 10011z \Rightarrow 10011$

10011

ACEITA!

Gramáticas Lineares à Direita e Linguagens Regulares

A gramática

$x \rightarrow 0x \mid 1y$

$y \rightarrow 0x \mid 1z$

$z \rightarrow 0x \mid 1z \mid \varepsilon$

é um exemplo de gramática linear à direita.

DEF: Uma ***gramática linear à direita*** é uma CFG em que toda regra é da forma $A \rightarrow uB$, ou $A \rightarrow u$ onde u é um string de terminais, e A,B são variáveis.

Gramáticas Lineares à Direita e Linguagens Regulares

THM: Se $N = (Q, \Sigma, \delta, q_0, F)$ é um AFN então existe uma gramática linear à direita G (N) que gera a linguagem $L(N)$.

Prova.

- Variáveis são os estados: $V = Q$
- O símbolo inicial é o estado incial: $S = q_0$
- Mesmo alfabeto de terminais: Σ
- Uma transição $(q_1, a) \rightarrow q_2$ torna-se uma regra $q_1 \rightarrow aq_2$
- Estados de aceitação $q \in F$ definem ε-regras $q \rightarrow \varepsilon$

Caminhos de aceitação correspondem a derivações finais e vice-versa. □

Gramáticas Lineares à Direita e Linguagens Regulares

Q: O que você pode dizer se usamos a mesma idéia para converter um AFD? Que propriedades a gramática terá?

Gramáticas Lineares à Direita e Linguagens Regulares

R: Como um AFD define caminhos de aceitação unívocos , cada string aceito terá uma única derivação mais à esquerda. Portanto, a gramática correspondente é não ambígua:

THM: A classe das linguagens regulares é igual à classe das linguagens geradas por Gramáticas Lineares à Direita não ambíguas.

Prova. Mostramos que toda linguagem regular é linear à direita não ambígua.

EXERCICIO P/ CASA: Mostre o inverso, ou seja, dada uma gramática linear à direta, como construir um GNFA correspondente. □

Gramáticas Lineares à Direita e Linguagens Regulares

Q: Será que toda CFG pode ser convertida em uma gramática linear à direita?

Gramáticas Lineares à Direita e Linguagens Regulares

R: Não! Isso significaria que toda linguagem livre de contexto é regular, o que já vimos que não é verdade.

EX:

$$S \rightarrow \varepsilon \mid aSb$$

Não pode ser convertida, porque a linguagem $\{a^n b^n\}$ não é regular.

Forma Normal de Chomsky

Embora nem toda CFG possa ser convertida em uma gramática linear à direita, ou mesmo, *em geral*, ser transformada em uma gramática não ambígua, existe uma forma simples para a qual toda CFG pode ser convertida:

Forma Normal de Chomsky

Noam Chomsky definiu um tipo de gramática livre de contexto especialmente simple e capaz de expressar todas as linguagens livres de contexto.

A forma gramatical de Chomsky é particularmente útil quando queremos provar certos fatos sobre linguagens livres de contexto. Isso porque considerar uma forma mais restritiva de gramática muitas vezes torna provas mais fáceis e simples.

Forma Normal de Chomsky Definição

DEF: Uma CFG é dita na ***Forma Normal de Chomsky*** se cada regra da gramática tem uma das seguintes formas:

- $S \rightarrow \varepsilon$ (se a lingugem contém ε)
- $A \rightarrow BC$ (regra de variável diádica)
- $A \rightarrow a$ (regra de terminal)

Onde *S* é a variável inicial, *A,B,C* são variáveis e *a* é um terminal. Portanto, ε pode ocorrer apenas do lado direito de uma regra de *S;* o lado direito das demais regras ou tem 2 variáveis ou tem um único terminal.

CFG → CNF

A conversão de uma CFG para a Forma Normal de Chomsky é feita em 4 passos:

1. Garanta que a variável inicial não ocorre do lado direito de nenhuma regra.
2. Remova todas as ε–regras, exceto a da variável inicial (caso ocorra).
3. Remova regras cujo lado direito é uma única variável (da forma $A \rightarrow B$).
4. Adicione variáveis novas e regras diádicas para substituir regras cujo lado direito tem mais de 2 variáveis ou mais de 1 terminal.

CFG → CNF
Exemplo

Vamos ver como isso funciona, aplicando o método à seguinte gramática para pal:

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

CFG → CNF

1. Variável inicial

Garanta que a variável inicial não ocorre do lado direito de nenhuma regra.

$$S' \rightarrow S$$

$$S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$$

CFG → CNF

2. Remova ε–regras

Remova todas as ε–regras, exceto da variável inicial (se ocorrer).

$$S' \rightarrow S \mid \varepsilon$$

$$S \rightarrow \cancel{\varepsilon} \mid a \mid b \mid aSa \mid bSb \mid aa \mid bb$$

CFG → CNF

3. Remova Renomeação

Remova produções com uma única variável do lado direito (da forma $A \rightarrow B$).

$$S' \rightarrow \cancel{S} \mid \varepsilon \mid a \mid b \mid aSa \mid bSb \mid aa \mid bb$$

$$S \rightarrow a \mid b \mid aSa \mid bSb \mid aa \mid bb$$

CFG → CNF

4. Produções mais longas

Adicione variáveis e regras diádicas para substituir regras muito longas.

$$S' \to \varepsilon \mid a \mid b \mid \sout{aSa} \mid \sout{bSb} \mid \sout{aa} \mid \sout{bb} \mid AB \mid CD \mid AA \mid CC$$

$$S \to a \mid b \mid \sout{aSa} \mid \sout{bSb} \mid \sout{aa} \mid \sout{bb} \mid AB \mid CD \mid AA \mid CC$$

$$A \to a$$

$$B \to SA$$

$$C \to b$$

$$D \to SC$$

CFG → CNF
Resultado

$$S' \to \varepsilon \mid a \mid b \mid AB \mid CD \mid AA \mid CC$$

$$S \to a \mid b \mid AB \mid CD \mid AA \mid CC$$

$$A \to a$$

$$B \to SA$$

$$C \to b$$

$$D \to SC$$

Exercício

- Passe a CFG para CNF

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

P → APCB | C

A → AaaA | ε

B → BBb | C

C → c C | ε

b) G=({E, T, F}, {+, *, a, b, (,)}, R , E)

E → E + T | T

T → T * F | F

F → (E) | a | b

CFG
Provando Correção

A natureza recursiva de CFGs indica que é natural provar sua correção indutivamente.

Por exemplo considere a gramática

$$G = (S \rightarrow \varepsilon \mid ab \mid ba \mid aSb \mid bSa \mid SS)$$

Afirmamos que a linguagem gerada por G é

$$L(G) = \{ x \in \{a,b\}^* \mid n_a(x) = n_b(x) \}.$$

$n_a(x)$ é o número de *a'*s em *x,* e

$n_b(x)$ é o número de *b'*s.

CFG
Provando Correção

Prova. Devemos provar duas partes. Seja L a linguagem supostamente gerada por G. Queremos mostrar que $L = L(G)$. Então devemos provar as duas seguintes inclusões:

I. $L \subseteq L(G)$. Todo string da suposta linguagem deve poder ser gerado por G.

II. $L \supseteq L(G)$. G apenas gera strings da suposta linguagem.

Provando Correção

$L \subseteq L(G)$

I. $L \subseteq L(G)$: Provar que todo string x com igual número de a's e b's é gerado por G. A prova é por indução sobre $n = |x|$.

Caso base ($n = 0$): Então $x = \varepsilon$ e pode ser derivado pela regra $S \rightarrow \varepsilon$.

Caso indutivo: Suponha $n > 0$. Seja u o menor prefixo não vazio de x que é também um string de L. Existem dois casos:

1) $u = x$
2) u é um prefixo próprio de x

Provando Correção

$L \subseteq L(G)$

I.1) $u = x$: Note que u não pode começar e terminar com a mesma letra. Ex., se começar e terminar com a, $u = ava$, para algum v, o que significa que *v deve ter* 2 *b's* a mais que *a's*. Então em algum ponto de v os *b's* de u "casam" com os *a's*, o que significa que existe em x um prefixo menor que esteja em L, contradizendo a definição de u como o *menor* prefixo de x em L. Portanto, para algum v em L temos que

$u = avb$ OU $u = bva$

O comprimento de v é menor do que o de u e, portanto, pela hipótese de indução, existe uma derivação $S \Rightarrow^* v$. Portanto, u pode ser derivado por:

$S \Rightarrow aSb \Rightarrow^* avb = x$ OU $S \Rightarrow aSb \Rightarrow^* avb = x$

Provando Correção

$L \subseteq L(G)$

I.2) *u* é um prefixo próprio de *x*. Então podemos escrever

$x = uv$ sendo *u e v* ambos em *L*

Como *u* e *v* têm ambos comprimento menor que *x*, podemos aplicar a hipótese de indução, e supor que existem derivações $S \Rightarrow^* u$ e $S \Rightarrow^* v$. Então *x* pode ser derivado como:

$$S \Rightarrow SS \Rightarrow^* uS \Rightarrow^* uv = x$$

Provando Correção

$L \supseteq L(G)$

II) Devemos agora provar que todo string derivado por *G* pertence a *L*. Vamos provar algo mais geral: todo string em {*S,a,*b}* derivável de *S* contém igual número de *a'*s e *b'*s. Novamente usamos indução, mas agora sobre o número de passos *n* da derivação

Caso base (*n* = 0): O único string derivável em 0 passos é *S* , que tem 0 *a*'s e 0 *b*'s. OK!

Caso indutivo: Seja *u* um string derivável de *S* em *n* passos.

I.e., $S \Rightarrow^n u$. Qualquer passo de derivação adicional deve utilizar uma das regras $S \rightarrow \varepsilon \mid ab \mid ba \mid aSb \mid bSa \mid SS$. Mas cada uma delas preserva a igualdade de número de *a'*s e *b'*s. Portanto, qq string derivável de *S* em *n*+1 passos também tem igual número de *a*'s e *b*'s. //**QED**