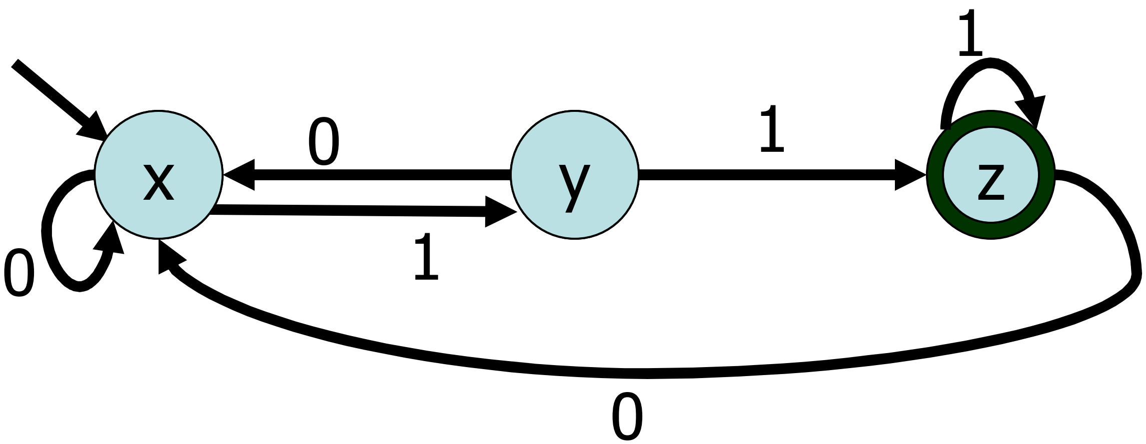
Lema do Bombeamento Linguagens Livres de Contexto

Agenda

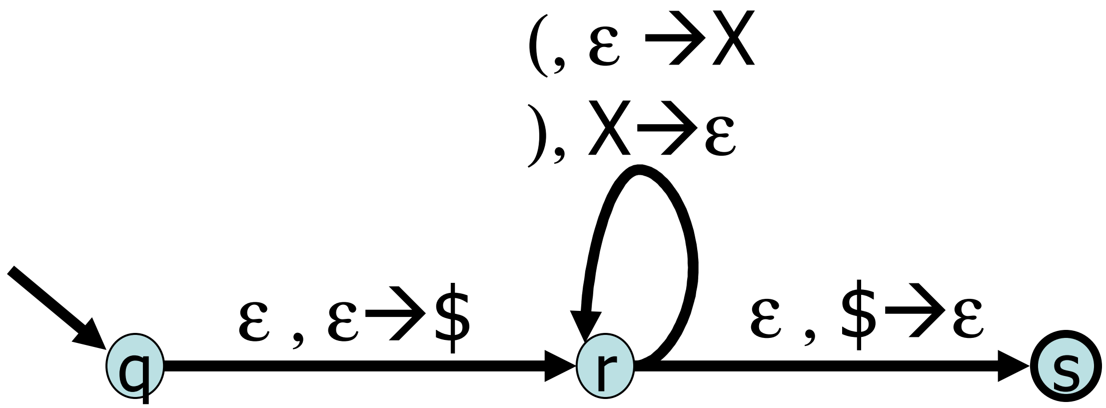
- Lema do Bombeamento para CFL’s
  - Motivação
  - Teorema
  - Prova
- Exemplos de provas usando o lema

Bombeando FA's



Strings de comprimento 3 ou mais no AFD acima podem ser bombeados, pois tais strings correspondem a caminhos de comprimento ≥ 3, e portanto visitam ≥ 4 vértices. O princípio da Casa dos Pombos guarante então que algum vértice é visitado mais de uma vez, resultando em um ciclo de bombeamento.

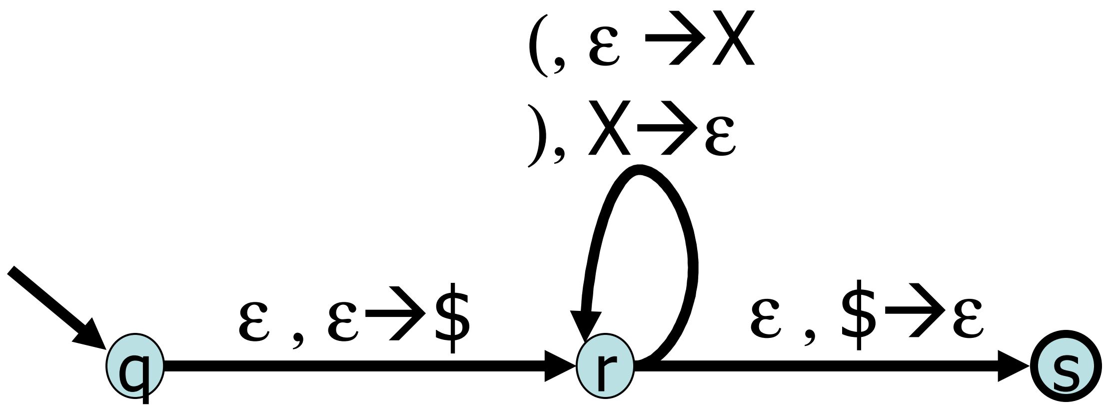
Bombeando PDA's



Entretanto, o lema de bombeamento para linguagem regular falha nesse exemplo.

Q: Dê um exemplo de string que não pode ser bombeado.

Bombeando PDA's



R: $(^n)^n$ não pode ser bombeado na primeira metade.

Entretanto, poderíamos bombear *dois* substrings de uma vez. I.e. tomar *k* parênteses da esquerda e *k da direita*.

Bombeamento Duplo

DEF: Um string s em L é dito ***duplamente bombeável*** se podemos dividir s em

$$s = uvxyz$$

de modo que para todo $i \geq 0$ temos que

$$s = uv^{i}xy^{i}z \in L$$

sendo pelo menos um de v,y não vazio.

Q1: 00111 é duplamente bombeável em 0*111?

Q2: 00100 é duplamente bombeável em $\{0^n10^n\}$?

Q3: 00100100 é duplamente bombeável em $\{0^n10^n10^n\}$?

Bombeamento Duplo

R1: Sim. Todo string bombeável é também duplamente bombeável, fazendo-se $y = \varepsilon$. Neste caso, tomamos $u = \varepsilon$, $v = 00$, $x = y = \varepsilon$, $z = 111$.

$uv^i xy^i z = (00)^i 111$ está de fato em 0^*111.

R2: Sim. Faça $u = \varepsilon$, $v = 00$, $x = 1$, $y = 11$ e $z = \varepsilon$

$uv^i xy^i z = (00)^i 1 (00)^i$ está de fato em $\{0^n 1 0^n\}$

R3: NÃO! Bombeando duplamente 00100100 ou leva a excesso de 1's, ou aumenta 2 das seqüências de 0's, sem aumentar a seqüência de 0's restante.

Bombeamento Duplo

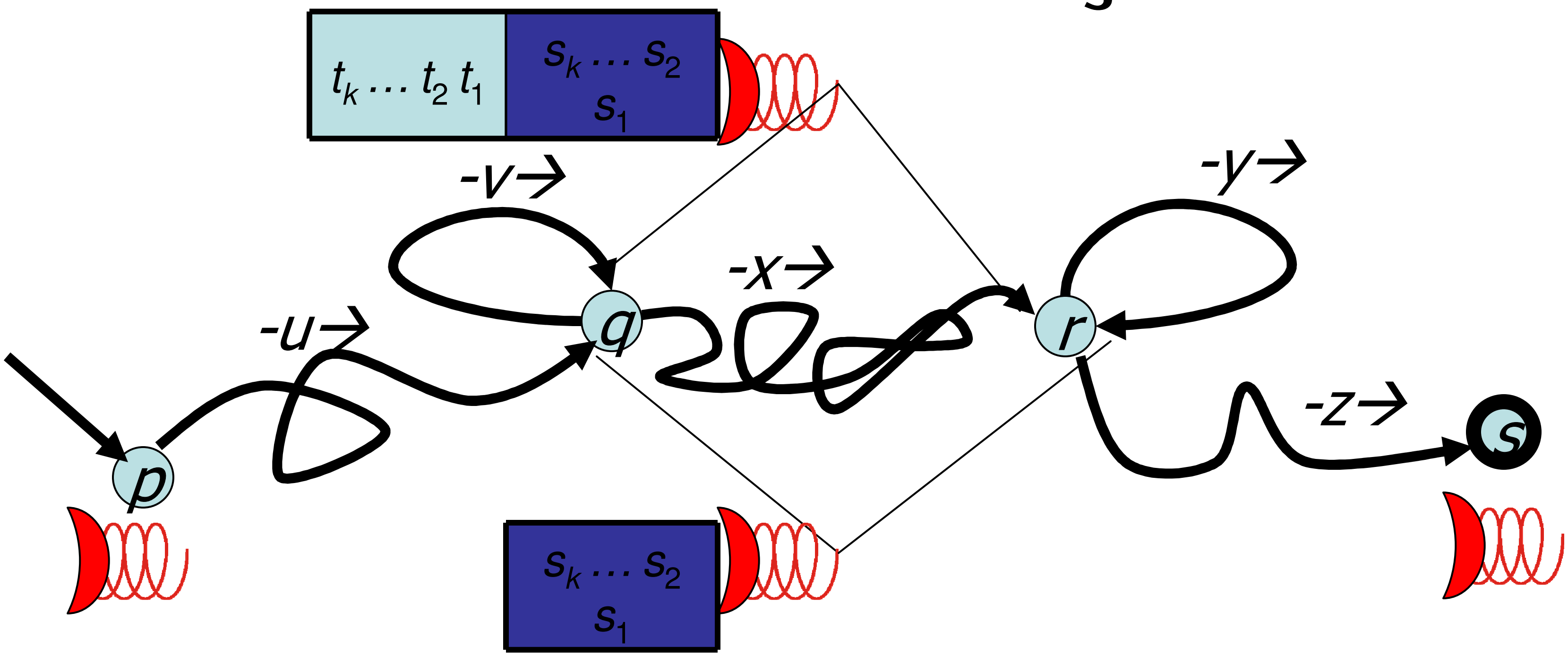
Em geral, como bombeamento implica bombeamento duplo, toda linguagem regular (infinita) é duplamente bombeável. Também é verdade que toda linguagem livre de contexto (infinita) é duplamente bombeável. Mas Q3 pode ser generalizada de modo a mostrar que $\{0^n10^n10^n\}$ não admite bombeamento duplo para strings com comprimento maior do que um determinado. Isso termina provando que $\{0^n10^n10^n\}$ não é livre de contexto:

Lema do Bombeamento Livre de Contexto

THM: Dada uma linguagem livre de contexto L, existe um número p (***no. de bombeamento duplo***) tal que todo string em L de comprimento $\geq p$ é duplamente bombeável dentro de um substring de comprimento p. Em outras palavras, para todo $s \in L$ com $|s| \geq p$ podemos escrever :

- $s = uvxyz$
- $|vy| \geq 1$ (partes bombeávei não vazias)
- $|vxy| \leq p$ (bombeamento dentro da porção p)
- $uv^ixy^iz \in L$ for all $i \geq 0$ (bombea v e y)

CFPL – Intuição



Intuitivamente $s = uvxyz$ é encontrado do seguinte modo: Apenas um número finito de mudanças na pilha podem ocorrer em um ciclo do grafo de comprimento $\leq n$ (o número de estado). Portanto, se s é suficientemente longo, existirão estados q,r tais que o mesmo string empilhado em q é desempilhado em r e tais que o caminho de q a r começa e termina com a mesma configuração de pilha. Com essa hipótese, podemos bombear juntos v e y já que v empilha o que y desempilha

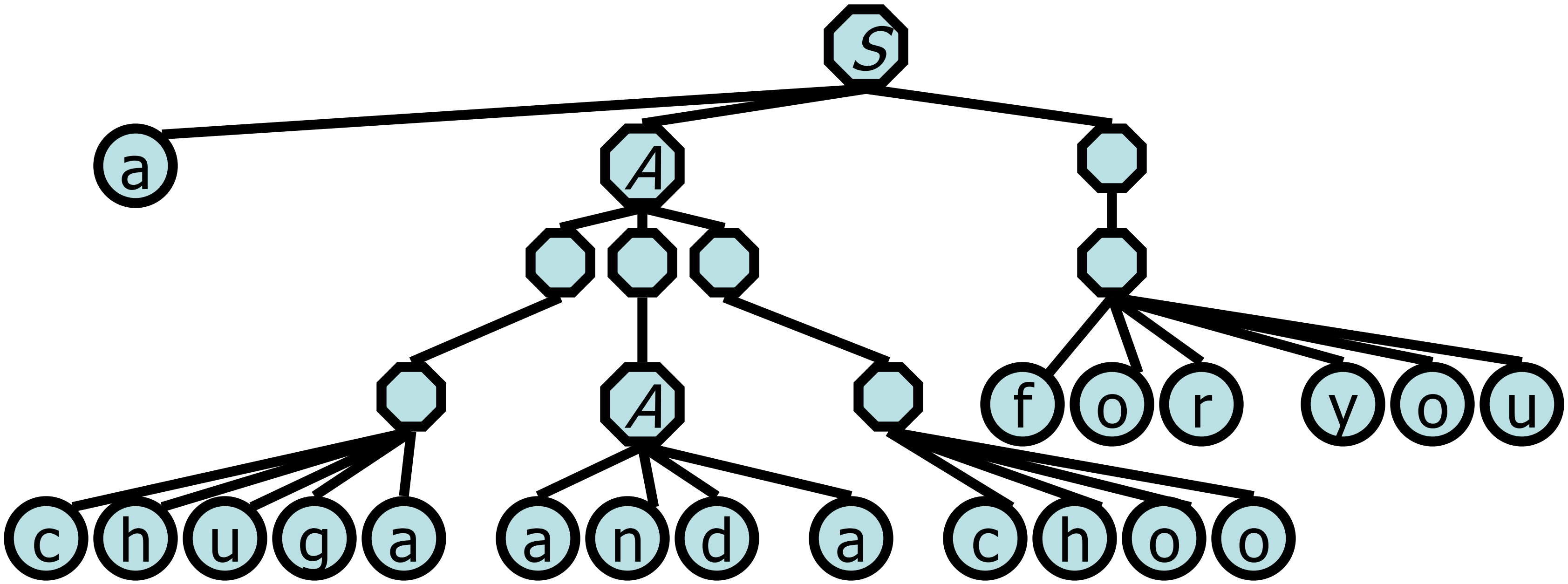
CFPL - Prova

O que foi mostrado anteriormente pode ser formalizado para provar o Lema do Bombeamento para linguagens livres de contexto. Entretanto a prova é muito mais complicada do que a prova formal baseada em gramática:

Prova do CFPL: Seja *L* uma linguagem livre de contexto.

Considere uma árvore de derivação de um string de *L* na qual algum nodo de variável tem ele próprio como ancestral:

CFPL – Prova



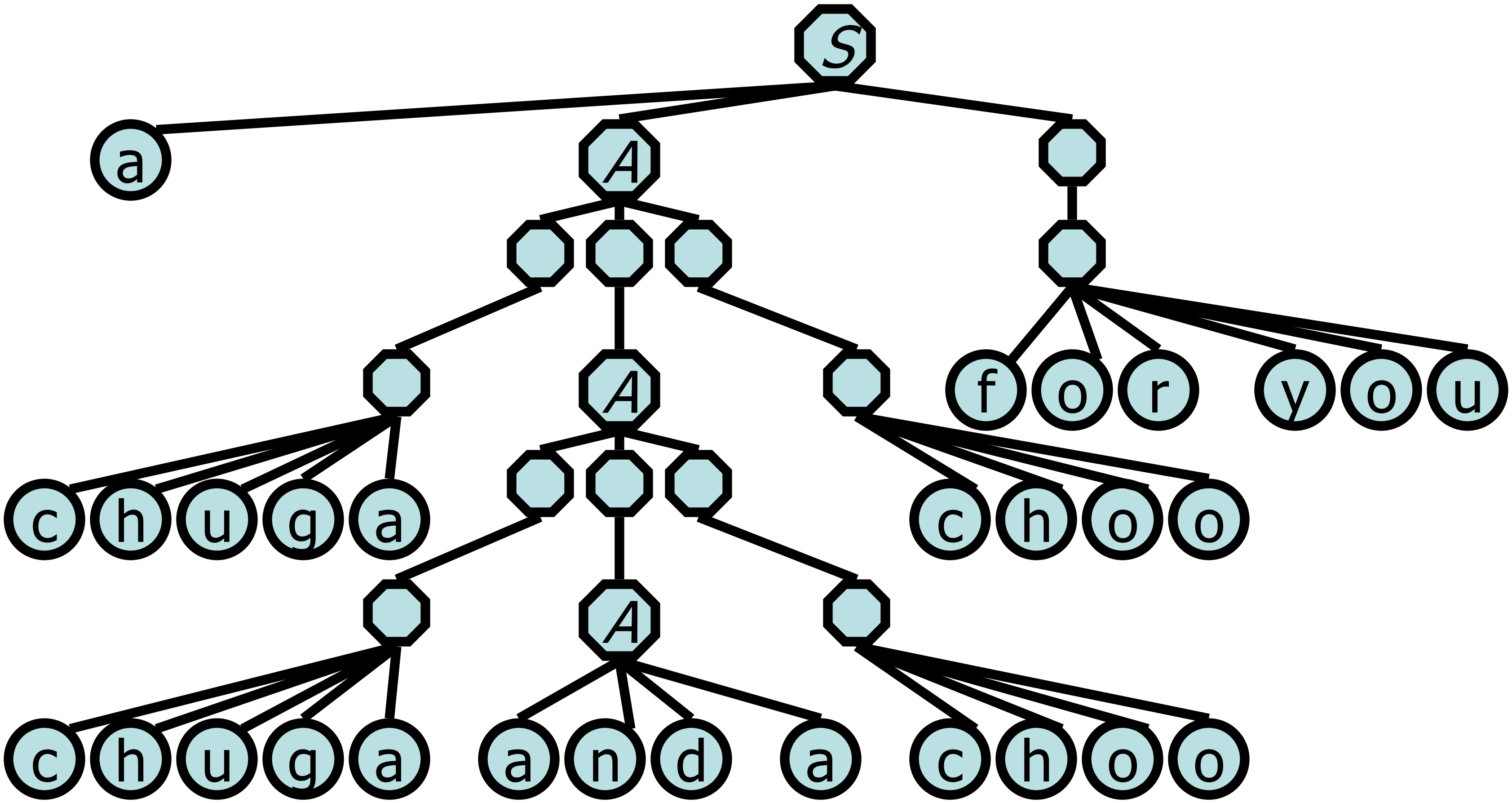
Podemos substituir a última ocorrência de *A* pela primeira. I.e., substituir na árvore

$A \Rightarrow^*$ "and a" por

$A \Rightarrow^*$ "chuga and a choo"

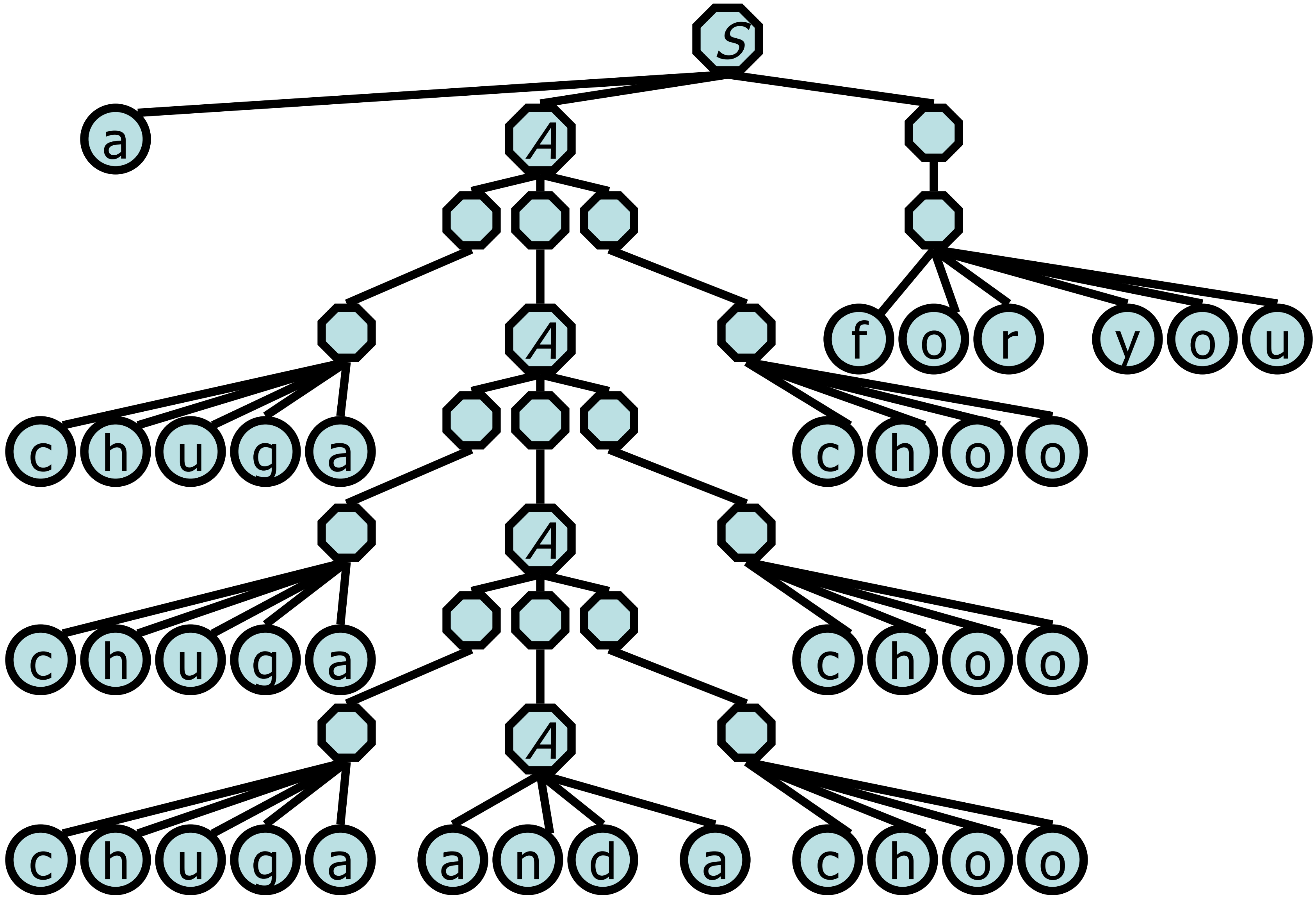
Obtendo o seguinte:

CFPL – Prova

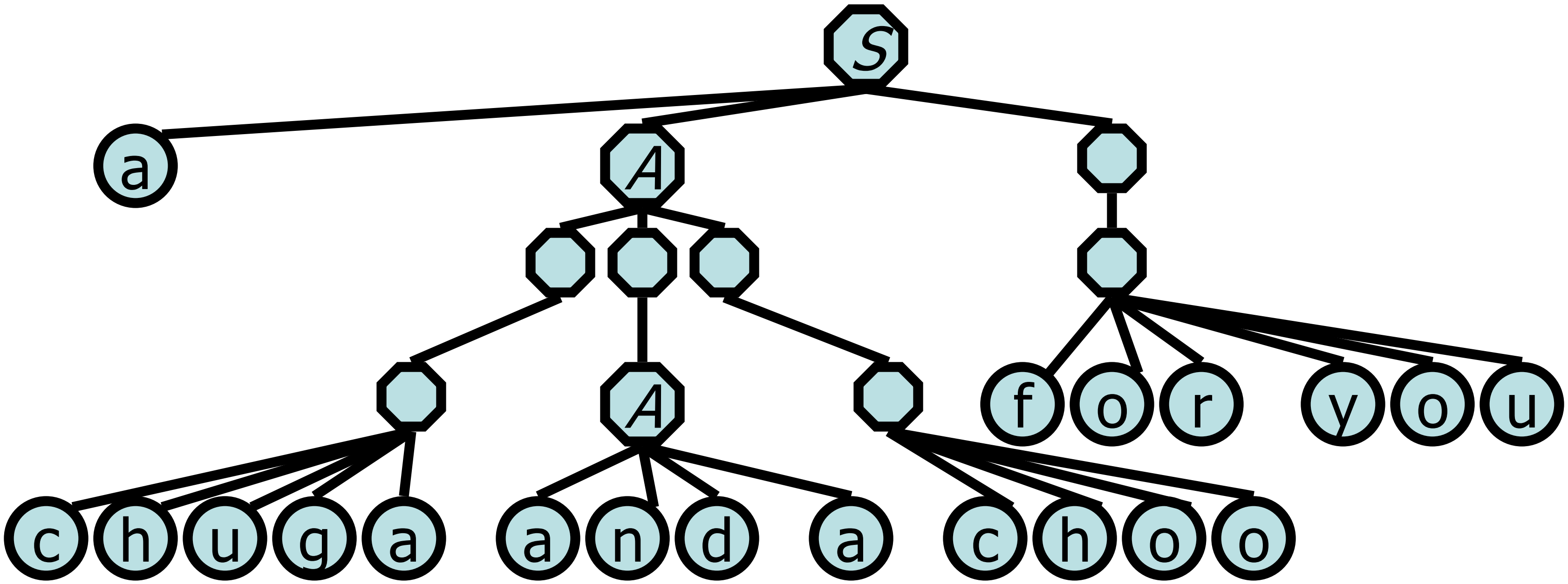


E novamente:

CFPL – Prova



CFPL – Prova



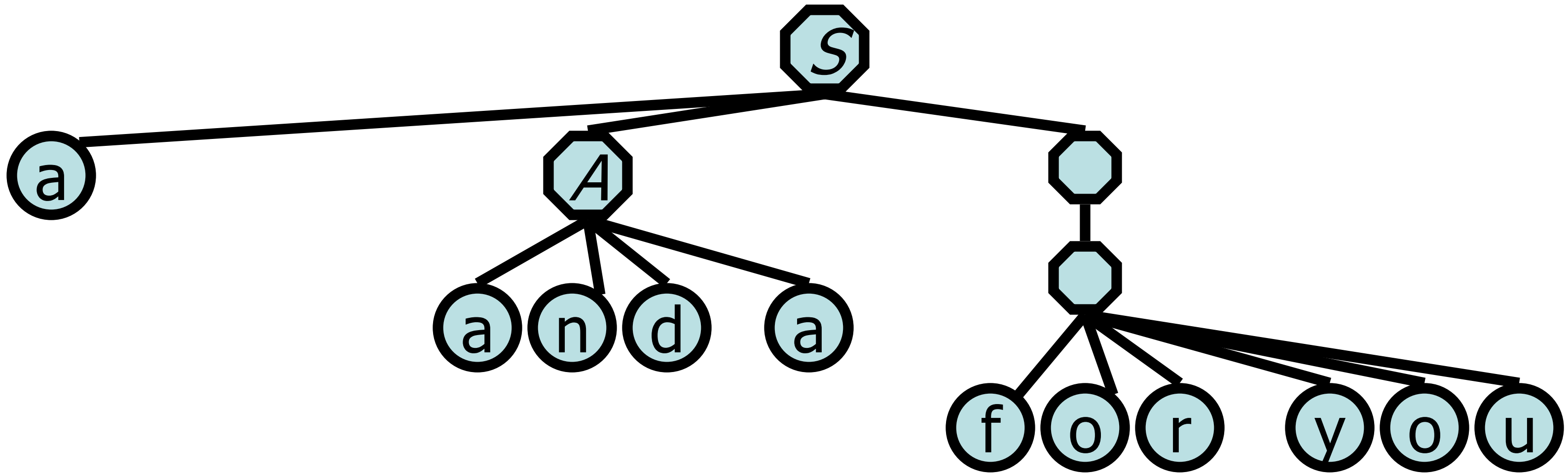
Ou podemos substituir

$A \Rightarrow^*$ "chuga and a choo" por

$A \Rightarrow^*$ "and a"

obtendo o resultado a seguir:

CFPL – Prova



No nosso caso particular, podemos criar qualquer string da forma

a (chuga)^{i} and a (choo)^{i} for you

CFPL – Prova

De modo geral, qualquer caminho na árvore de derivação que tenha uma variável repetida dá origem a strings da forma $uv^{i}xy^{i}z$, todos em L.

O restante da prova é apenas um argumento de contagem que garante a ocorrência de uma variável repetida.

CFPL – Prova

Q: Se *n* é o número de variáveis da gramática, para qual altura da árvore se pode garantir que pelo menos uma variável ocorre repetida?

(Lembre-se: a altura da árvore trivial – apenas a raiz – é 0)

CFPL – Prova

R: Se n é o número de variáveis da gramática, qualquer subárvore de altura $h = n+1$ terá uma variável repetida. Isso porque o nível inferior de uma árvore de derivação é composto de terminais, portanto altura $n+1$ (= $n+2$ níveis) garante $n+1$ níveis de variáveis, em pelo menos um ramo da árvore. O princípio da casa dos pombos garante que alguma variável ocorre 2 vezes!

CFPL – Prova

Q: Se a gramática está na Forma Normal de Chomsky, de que tipo é qualquer árvore de derivação?

CFPL – Prova

R: Uma árvore binária!

Q: Qual é o número máximo de folhas que uma árvore binária de altura *n* pode ter?

CFPL – Prova

A: 2^n

Q: Qual é o número máximo de folhas que pode ter uma árvore de derivação de uma gramática na Forma Normal de Chomsky, se a altura da árvore de derivação é $n+1$?

CFPL – Prova

A: Também 2^n! Isso porque a única maneira de obter um terminal é por uma regra da forma $A \rightarrow a$ e, portanto, não há dois ramos no último nível.

Q: Que comprimento de string garante que sua árvore de derivação tem altura $\geq n+1$?

CFPL – Prova

R: 2^n. Isso porque nenhuma árvore com comprimento $< n+1$ poderia gerar essa quantidade de folhas, ou terminais.

Isso nos leva a definir o número de bombeamento duplo como $p=2^n$.

O resto do teorema segue das considerações feitas previamente. Apenas precisamos verificar que o bombeamento pode ocorrer em um substring de comprimento p. Isso decorre de ocorrer uma variável repetida nos últimos $n+2$ níveis da árvore.

Provando que *L* não CFL

Método padrão p/ aplicar o lema do bombeamento
Apenas no. 3 muda de exemplo p/ exemplo:

1. Suponha que a linguagem é livre de contexto.
2. Então existe um no. de bombeamento *p.*
3. Encontre um string *s* que *não* seja duplamente *bombeável* , em um substring de comprimento *p*
4. 2 e 3 se contradizem, portanto, 1 deve ser falso e a linguagem *não é* livre de contexto.

Provando que *L* não CFL
Exemplo 1

$L = \{ 1^n 0^n 1^n 0^n \mid n \geq 0 \}$

Provando que *L* não CFL Exemplo 1

A parte difícil é a número 3!!! Tente

$$s = 1^p 0^p 1^p 0^p$$

Existem 3 casos onde a "*janela de visão*" *vxy* poderia estar.

I III

1…10…01…10…0

II

Provando que *L* não CFL Exemplo 1

Caso I. Bombear p/ baixo (ou p/ cima) modificaria o no. de 0's e/ou o no. de 1's na primeira metado do string, sem alterar a segunda metade. Isso viola a definição da language.

$$\overbrace{1\ldots1\,0\ldots0}^{\mathrm{I}}\,\overbrace{1\ldots1\,0\ldots0}^{\mathrm{III}}$$

$$1\ldots1\,\underbrace{0\ldots0\,1\ldots1}_{\mathrm{II}}\,0\ldots0$$

Provando que *L* não CFL
Exemplo 1

Casos II e III. Mesmo argumento do Caso I. (Caso III causaria mudança na segunda metade sem alterar a primeira. Caso II causaria mudaça na parte do meio, sem alterar os primeiros 1^p ou os últimos.) Isso completa a prova.

I III
1…10…01…10…0
II

I III
1…10…01…10…0
II

Provando que L não CFL Exemplo 2

ADD = { x=y+z | x, y, e z são bit-strings que satisfazem a equação }

Provando que *L* não CFL
Exemplo 1

A parte difícil é a número 3! Seja *s*:

$$\mathbf{1}^{p+1}\mathbf{=1}^{p}\mathbf{+10}^{p}$$

Existem duas posições onde o substring *vxy* pode ocorrer. (Janela-*p*)

$$\overbrace{1^{p+1}=}^{\mathrm{I}}\underbrace{1^{p}+10^{p}}_{\mathrm{II}}$$

Provando que *L* não CFL Exemplo 1

Caso I. *v* deve ocorrer à esq. de “=” enquanto *y* deve ocorrer à dir. já que, caso contrário, o bombeamento resultaria em excesso de símbolos =, ou afetaria um lado da equação mas não o outro. Seja *k* o comprimento de *v* e *l* o comprimento de *y*. Bombeando p/ baixo obtemos a suposta equação: $1^{p+1-k}=1^{p-l}+10^{p}$. A equação não é válida porque o lado direito é muito maior do que o lado esquerdo.

$$\overbrace{1^{p+1}=}^{\mathrm{I}}\underbrace{1^{p}+10^{p}}_{\mathrm{II}}$$

Provando que *L* não CFL
Exemplo 1

Caso II. O bombeamento deve ocorrer à direita de "=": O lado direito é afetado sem que haja alteração do lado esquerdo. Isso não mantém a propriedade de que o string satisfaz a equação.

Isso conclui a prova de que a linguagem *ADD* não é livre de contexto.

$$\overbrace{1^{p+1}}^{\mathrm{I}} = \underbrace{1^p + 1\,0^p}_{\mathrm{II}}$$

Exercícios

- $\{1^n \mid n$ é primo$\}$
- $\{0^n 1^n 0^n 1^n\}$
- {int x; x = 3; | x é um string alfabético}
  - Portanto, dizer que Java é livre de contexto não é exato. (Se x = 3 ocorre, x deve ser previamente declarado!)