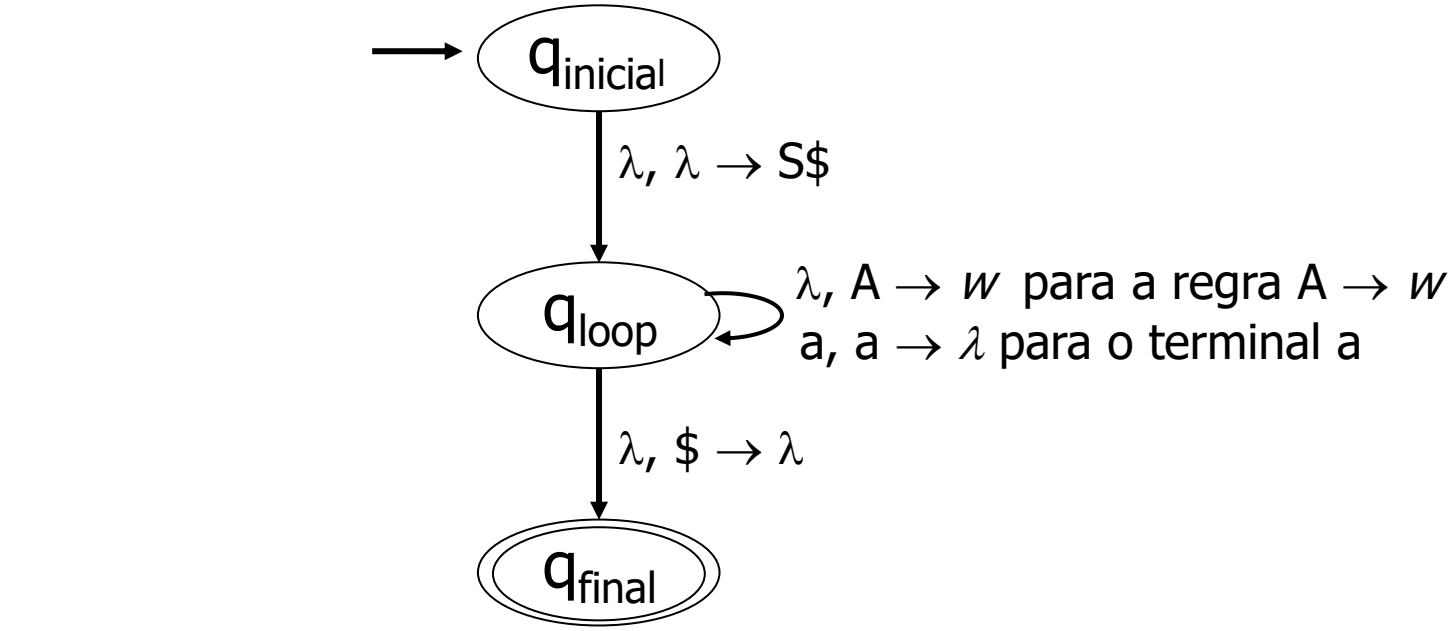
Lema do bombeamento e linguagens além de LC

Universidade Federal de Campina Grande – UFCG
Centro de Engenharia Elétrica e Informática – CEEI
Departamento de Sistemas e Computação – DSC
Professor: Andrey Brito Período: 2023.2

GLC → Autômato de pilha



$\delta(q_{inicio}, \lambda, \lambda) = \{ (q_{loop}, S\$) \}$

$\delta(q_{loop}, \lambda, A) = \{ (q_{loop}, w) \mid A \rightarrow w \in R \}$

$\delta(q_{loop}, a, a) = \{ (q_{loop}, \lambda) \}$

$\delta (q_{loop}, \lambda, \$) = \{ q_{aceita}, \lambda)\}$

Existe também um procedimento para converter APs em GLCs, mas não discutiremos ele aqui.

Outros usos de gramáticas

Os usos mais reais

Exemplo – Gramática para uma linguagem de programação

<*compilation unit*> ::= <*package declaration*>? <*import declarations*>? <*type declarations*>?

<*package declaration*> ::= **package** <*package name*> **;**

<*import declarations*> ::= <*import declaration*> | <*import declarations*> <*import declaration*>

<*import declaration*> ::= <*single type import declaration*> | <*type import on demand declaration*>

<*single type import declaration*> ::= **import** <*type name*> **;**

<*type import on demand declaration*> ::= **import** <*package name*> **. * ;**

<*type declarations*> ::= <*type declaration*> | <*type declarations*> <*type declaration*>

<*type declaration*> ::= <*class declaration*> | <*interface declaration*> | **;**

<*class declaration*> ::= <*class modifiers*>? **class** <*identifier*> <*super*>? <*interfaces*>? <*class body*>

http://www.cs.au.dk/~stm/RegAut/JavaBNF.html

Exemplo – For em Java

BasicForStatement:
for (; ;) Statement
for (; ; ForUpdate) Statement
for (; Expression ;) Statement
for (; Expression ; ForUpdate) Statement
for (ForInit ; ;) Statement
for (ForInit ; ; ForUpdate) Statement
for (ForInit ; Expression ;) Statement
for (ForInit ; Expression ; ForUpdate) Statement

http://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html

Exemplo – Import em Python

```
import_stmt ::=
        "import" module ["as" name]
          ( "," module ["as" name] )*
          | "from" relative_module "import" identifier
             ["as" name]
           ( "," identifier ["as" name] )*
          | "from" relative_module "import" "("
             identifier ["as" name]
           ( "," identifier ["as" name] )* [","] ")"
          | "from" module "import" "*"

module ::=
        (identifier ".")* identifier
```

https://inst.eecs.berkeley.edu/~cs164/fa11/python-grammar.html

Exemplo – Gramática da língua portuguesa

- Pedaço de uma GLC que descreveria a língua portuguesa

<FRASE> → <FRASE NOMINAL><FRASE VERBAL>

<FRASE NOMINAIL> → <SUJEITO COMPOSTO>| <SUJEITO COMPOSTO><FRASE PREPOSICIONAL>

<FRASE VERBAL> → <VERBO COMPOSTO>|<VERBO COMPOSTO><FRASE PREPOSICIONAL>

<FRASE PREPOSICIONAL> → <PREPOSIÇÃO><SUJEITO COMPOSTO>

<SUJEITO COMPOSTO> → <ARTIGO><SUJEITO>

<VERBO COMPOSTO> → <VERBO> | <VERBO><FRASE NOMINAL>

<ARTIGO> → um(a) | o(a)

<SUJEITO> → garoto | garota | flor

<VERBO> → toca | gosta | vê

<PREPOSIÇÃO> → com

- Usadas inicialmente para descrição de linguagens naturais: quais são as partes de uma sentença e como elas são organizadas
  - Houaiss: (1 gramática) conjunto de prescrições e regras que determinam o uso considerado correto da língua escrita e falada

Exemplo

<FRASE> → <FRASE NOMINAL><FRASE VERBAL>

<FRASE NOMINAIL> → <SUJEITO COMPOSTO>| <SUJEITO COMPOSTO><FRASE PREPOSICIONAL>

<FRASE VERBAL> → <VERBO COMPOSTO>|<VERBO COMPOSTO><FRASE PREPOSICIONAL>

<FRASE PREPOSICIONAL> → <PREPOSIÇÃO><SUJEITO COMPOSTO>

<SUJEITO COMPOSTO> → <ARTIGO><SUJEITO>

<VERBO COMPOSTO> → <VERBO> | <VERBO><FRASE NOMINAL>

<ARTIGO> → um(a) | o(a)

<SUJEITO> → garoto | garota | flor

<VERBO> → toca | gosta | vê

<PREPOSIÇÃO> → com

<FRASE> → <FRASE NOMINAL> <FRASE VERBAL>
→ <SUJEITO COMPOSTO> <FRASE VERBAL>
→ <ARTIGO> <SUJEITO> <FRASE VERBAL>
→ um <SUJEITO> <FRASE VERBAL>
→ um garoto <FRASE VERBAL>
→ um garoto <VERBO COMPOSTO>
→ um garoto <VERBO>
→ um garoto vê

Exemplo

- Exemplo de cadeias gerada pela gramática anterior
  - “um garoto vê”
  - “o garoto vê uma flor”
  - “uma garota com uma flor gosta do garoto”

Mas como eu projeto uma gramática?

Projetando GLCs – 4 regras gerais

1. Combinar duas gramáticas mais simples
   - (Frequentemente uma linguagem é uma união de linguagens mais “simples”)
   - Sejam S_1 e S_2 os símbolos iniciais das duas GLCs
   - Adicione $S \rightarrow S_1 \mid S_2$
   - (Forma semelhante pode ser usada para concatenação)

2. Converter AFs para GLCs
   - (Uma LLC pode ser a união de LLCs com LRs, por exemplo, reconhecer os identificadores, números e pedaços de uma expressão que não têm parênteses)
   - Crie uma variável R_i para cada estado q_i
   - Acrescente $R_i \rightarrow aR_j$ se $\delta(q_i, a) = q_j$
   - Se q_i é final adicione $R_i \rightarrow \lambda$
   - Se q_0 é inicial, a variável R_0 é a variável inicial

Projetando GLCs

3. Duas subcadeias que estão ligadas ou podem ser usadas recursivamente
   - (Uma LLC pode precisar de simetria, então você pode usar regras para inserir, por exemplos os parênteses em uma linguagem.)
   - Por exemplo: 1^n0^n ou ((a+a)+a)+a
   - Use regras do tipo: $R \rightarrow xRy$
   - Neste caso: 1R0 ou (R)

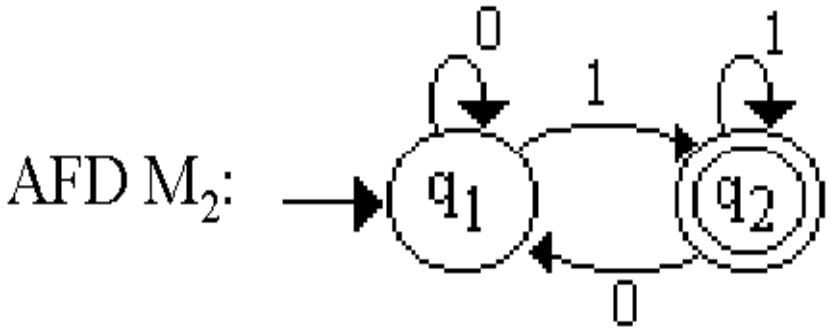
4. Atribua significado para os não terminais
   - Construa a gramática de forma interativa (como fazíamos com os AFs)

Agregando GLCs

- Seja $L = \{0^n1^n : n \geq 0\} \cup \{1^m0^m : m \geq 0\}$
- Construa a gramática para $\{0^n1^n : n \geq 0\}$:
  - $S_1 \rightarrow 0S_11 \mid \lambda$
- Construa a gramática para $\{1^m0^m : m \geq 0\}$:
  - $S_2 \rightarrow 1S_20 \mid \lambda$
- Adicione a regra que faz a união
  - $S \rightarrow S_1 \mid S_2$

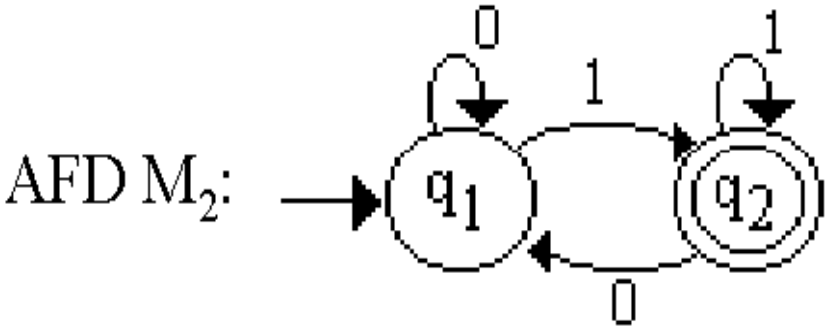
Convertendo AFs

- Seja L = {$w \in \{0, 1\}^*$ | w = x.1, $x \in \{0, 1\}^*$ }



Convertendo AFs

- Seja L = {$w \in \{0, 1\}^* \mid w = x.1, x \in \{0, 1\}^*$ }
- G = <V, Σ, R, S>
  - V = {R_1, R_2}
  - $\sum$ = {0, 1}
  - S = R_1
  - R:
    $R_1 \rightarrow 0R_1$
    $R_1 \rightarrow 1R_2$
    $R_2 \rightarrow 0R_1$
    $R_2 \rightarrow 1R_2$
    $R_2 \rightarrow \lambda$



Atribuindo significado

- Exemplo: uma gramática para construir somente os pares de parênteses, ex.: ()(), (()), (())()

- Atribuindo um sentido aos não terminais
  - A = tem um parênteses aberto, precisa lembrar de fechar um
  - F = está tudo ok, todos os abertos estão fechados, pode abrir ou parar

F → (AF *Quero abrir, então tenho que lembrar de fechar, depois posso começar de novo.*

F → λ *Se não devo nada, posso parar.*

A →) *Pagando a dívida.*

A → (AA *Aumentando a dívida, agora tenho que lembrar que tem dois abertos.*

Atribuindo significado (outro exemplo)

- Para comparação: gramática que gera cadeias com o mesmo número de 0s e 1s recursivamente:
  S → 0S1 | 1S0 | SS | λ

- Novamente, mas agora pensando em como construir a palavra um símbolo por vez (o que é melhor que na gramática anterior)
  - Atribuindo um sentido às variáveis
    - A = a palavra está devendo um 1
    - B = a palavra está devendo um 0
    - S = está balanceada

  S → 0A | 1B | λ
  A → 1S | 0AA
  B → 0S | 1BB

Atribuindo significado (mais um)

- Base: 0, 1 e λ são palíndromos
  - P → 0 | 1 | λ
- Indução: se w é um palíndromo 0w0 e 1w1 são palíndromos
  - P → 0P0 | 1P1
- A gramática será:

  P → 0 | 1 | λ

  P → 0P0 | 1P1

Projetando gramáticas

- Cuidado
  - "*Todas as cadeias geradas têm o mesmo número de 0s e 1s*", exemplo:

    S → 0S1 | 1S0 |λ

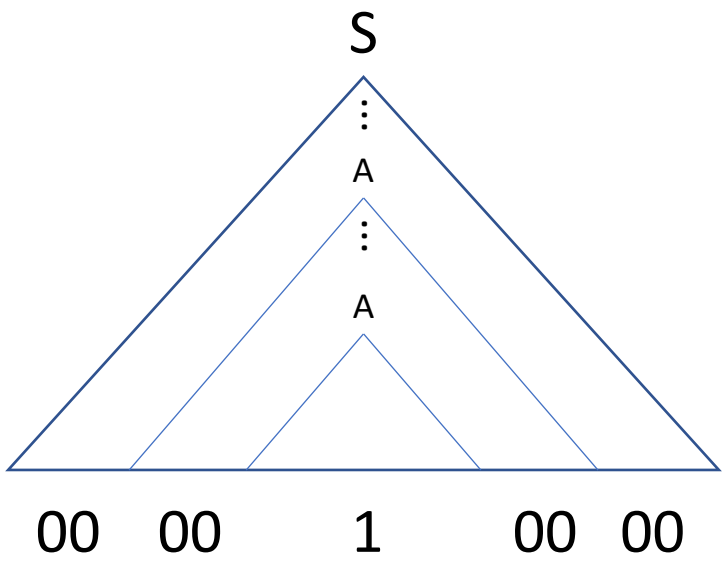
  - Não é a mesma coisa que gerar "*todas as cadeias com mesmo número de 0s e 1s*"

- E como saber se algo está correto?
  - Na realidade provar certas propriedades de gramáticas é um problema "difícil" (*indecidível*)

Lema do bombeamento para GLCs

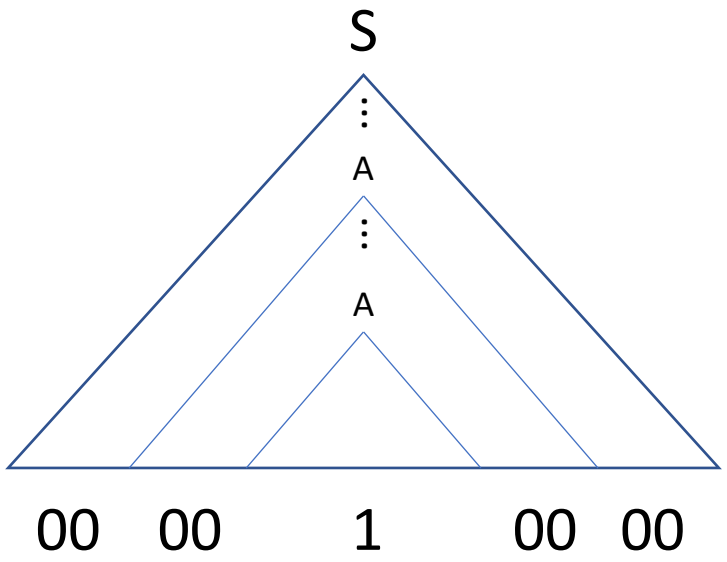
Lema do bombeamento para LLC

- Dada uma árvore de derivação...



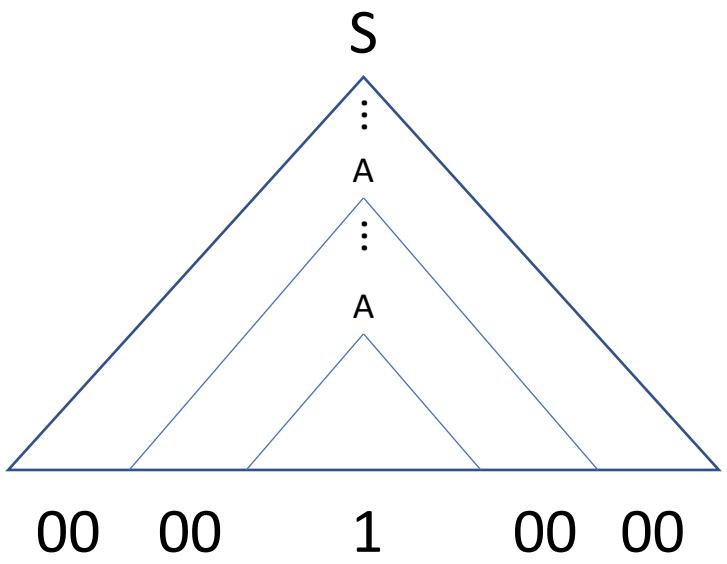
Lema do bombeamento para LLC

- Você conseguiria gerar outras palavras da mesma gramática?



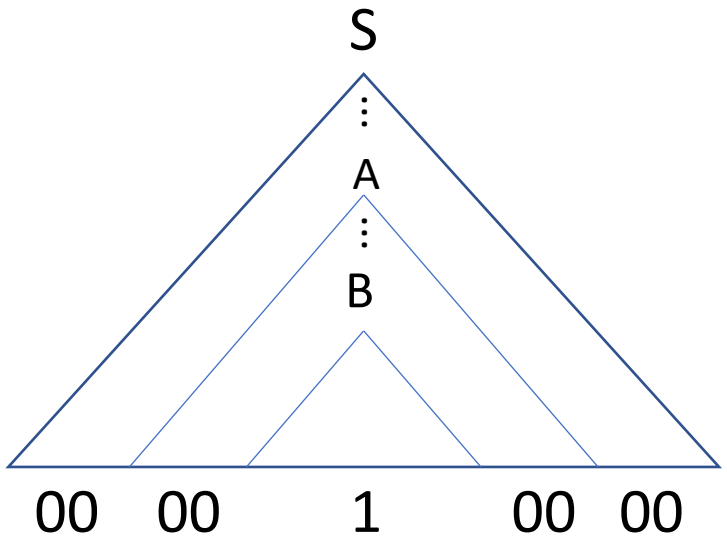
Lema do bombeamento para LLC

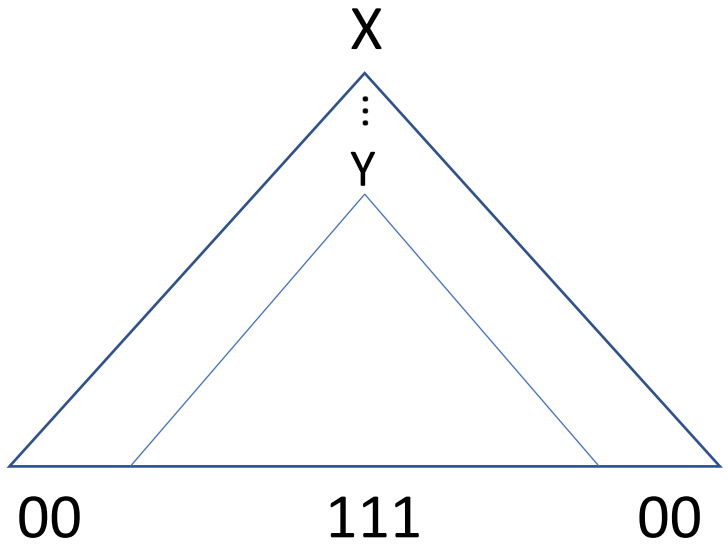
- Você conseguiria gerar outras palavras da mesma gramática? Quantas? Você conseguiria fazer isso para qualquer árvore?



Lema do bombeamento para LLC

- E agora? Dadas outra árvore de derivação que saiu de outra gramática, quantas outras palavras da mesma gramática você conseguiria gerar?

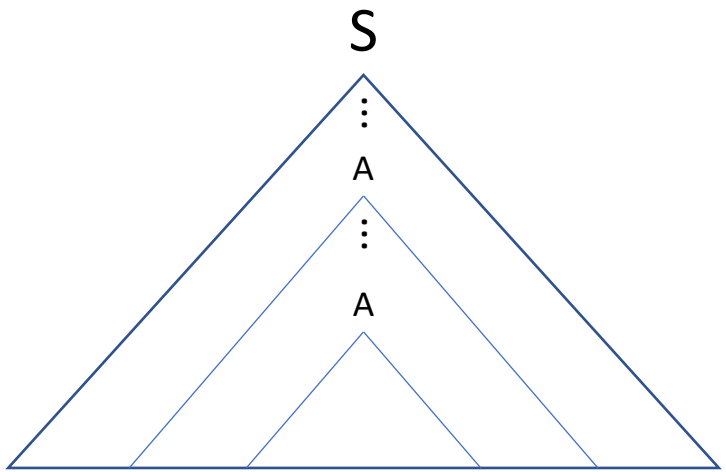




Lema do bombeamento para LLC

- Gerar infinitas palavras só acontece se puder gerar palavras arbitrariamente grandes, e isso só acontece com ciclos

$S \rightarrow 0A1 \mid B$
$A \rightarrow 1B0 \mid C$
$B \rightarrow 2C3 \mid D$
...
$Z \rightarrow \lambda$



$S \rightarrow 0A1 \mid B$
$A \rightarrow 1B0 \mid C$
$B \rightarrow 2C3 \mid D$
$C \rightarrow 3C2 \mid D \mid A$
...
$Z \rightarrow \lambda$

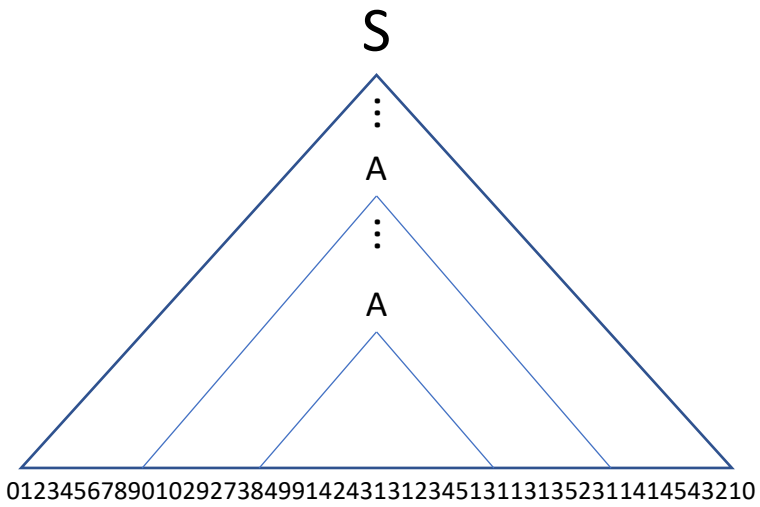
Comprimento máximo limitado a um certo tamanho, pois as variáveis não criam ciclos.

Comprimento máximo ilimitado, pois pode-se usar o ciclo A → B → C → A, para gerar uma palavra bem longa

Lema do bombeamento para LLC

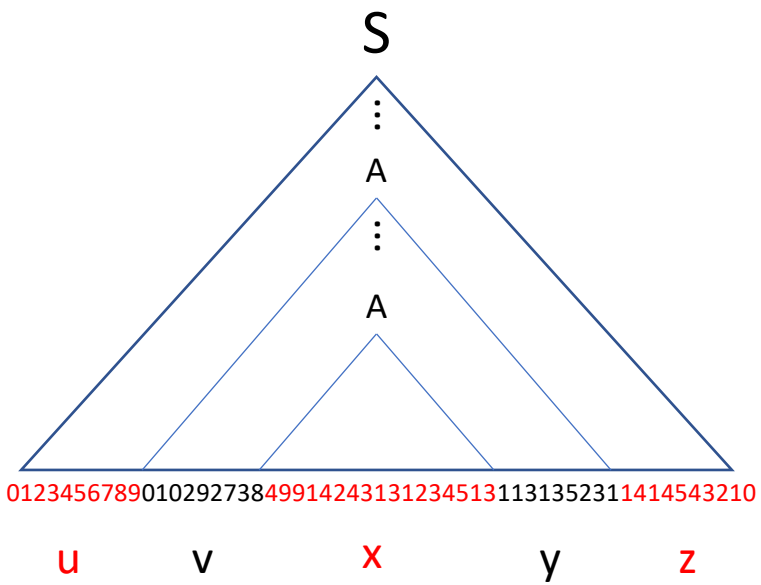
- Invertendo o pensamento: se a gramática tem palavras arbitrariamente longas, alguma variável tem que se repetir. (E se você tem 10 derivações e sua gramática só tem 5 regras?)

$S \rightarrow 0A0$
$A \rightarrow 1B1 \mid \lambda$
$B \rightarrow 2C2$
$C \rightarrow A$



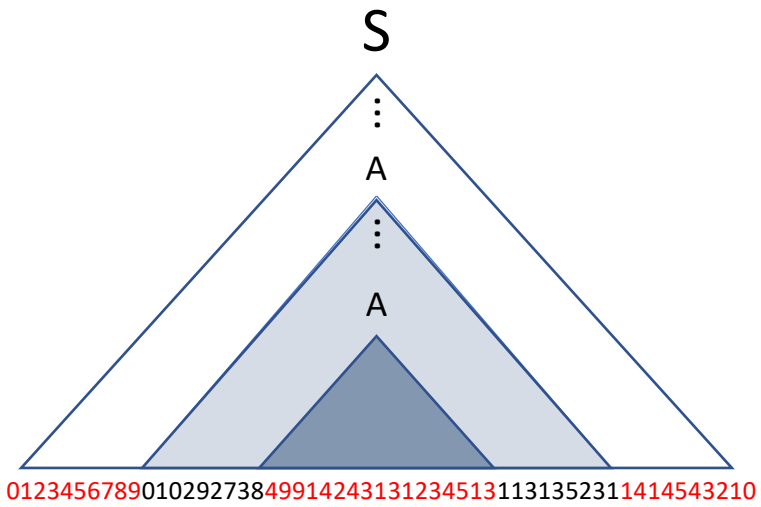
Lema do bombeamento para LLC

- Se a gramática tem palavras arbitrariamente longas, alguma variável tem que se repetir para as palavras longas



Bombeando para menos

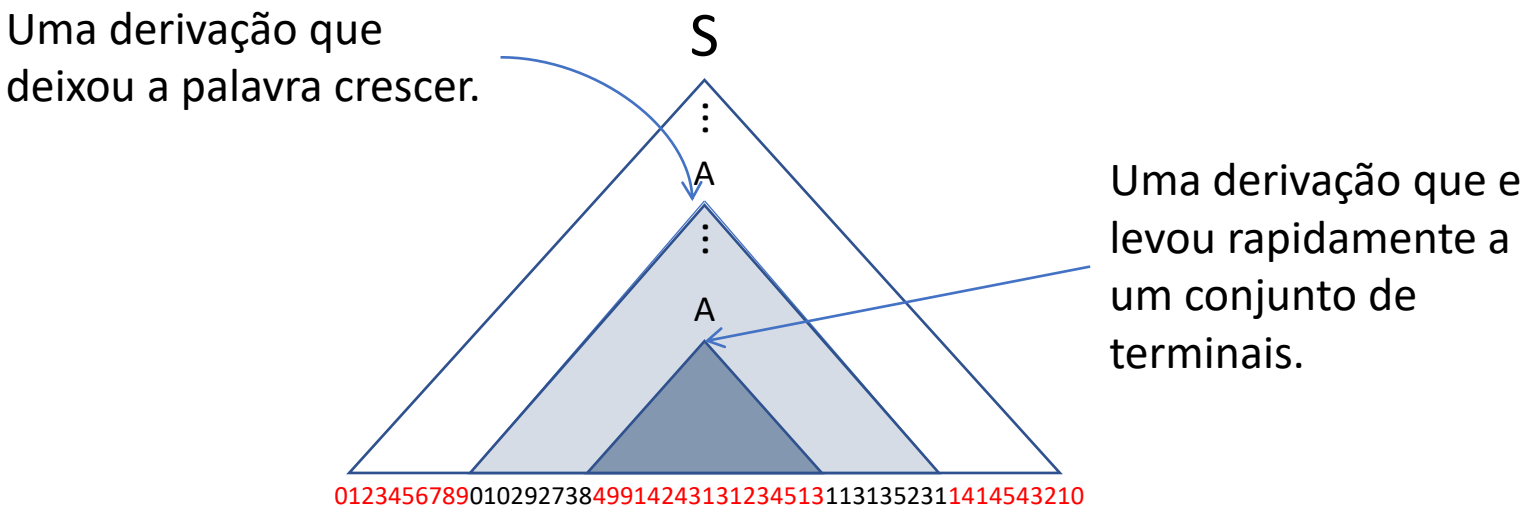
- Se a gramática tem palavras arbitrariamente longas, alguma variável tem que se repetir para as palavras longas



Novamente: Dada uma árvore de derivação como essa, você conseguiria construir outras palavras que certamente estariam na linguagem? Que palavras seriam essas?

Bombeando para menos

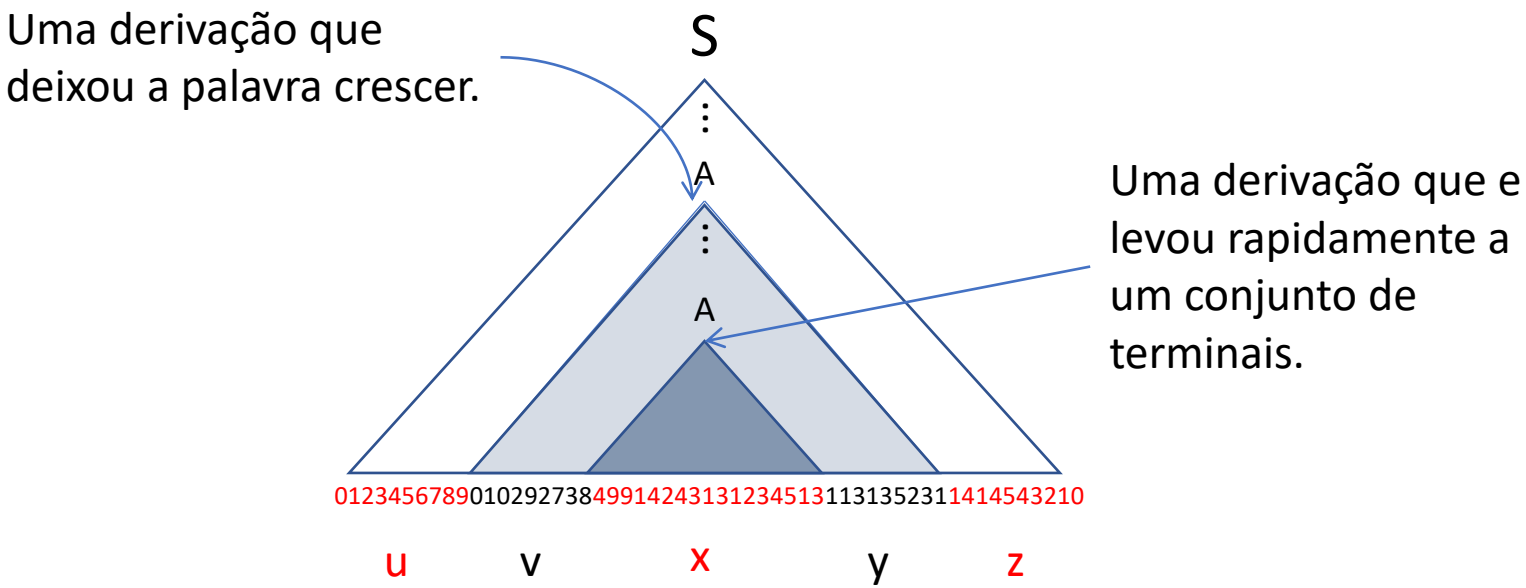
- Se a palavra é longa demais, alguma variável tem que se repetir



Novamente: Dada uma árvore de derivação como essa, você conseguiria construir outras palavras que certamente estariam na linguagem? Que palavras seriam essas?

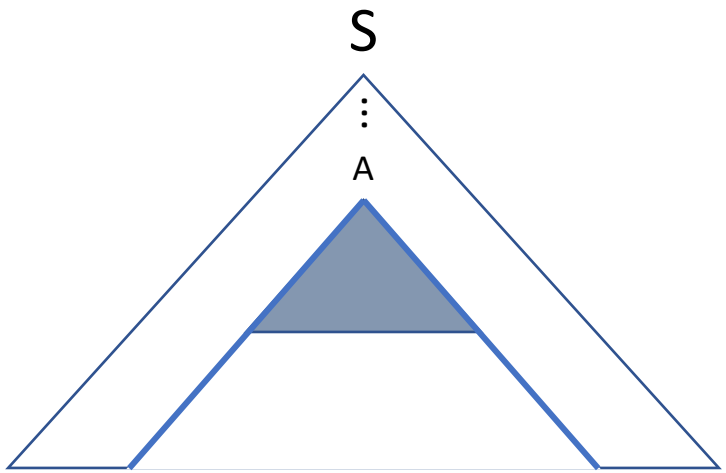
Bombeando para menos

- Se a palavra é longa demais, alguma variável tem que se repetir



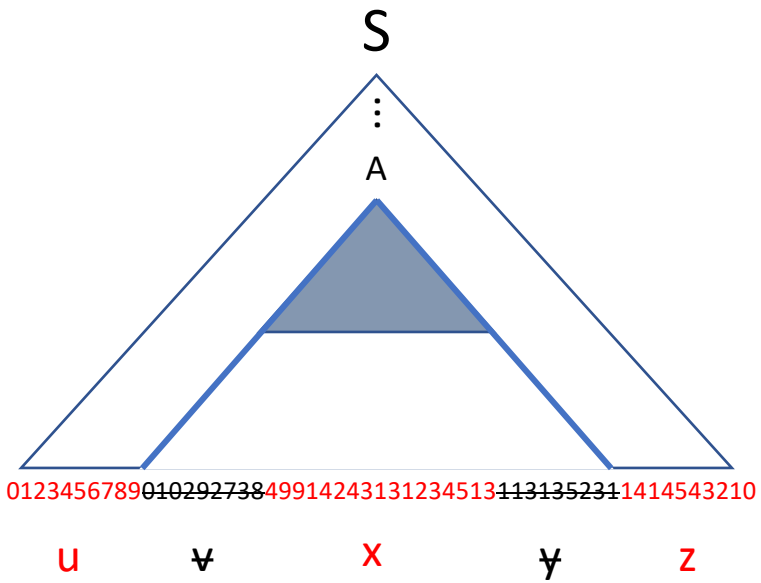
Bombeando para menos

- Então eu poderia ter uma palavra mais curta, fazendo a derivação “terminadora” mais em cima



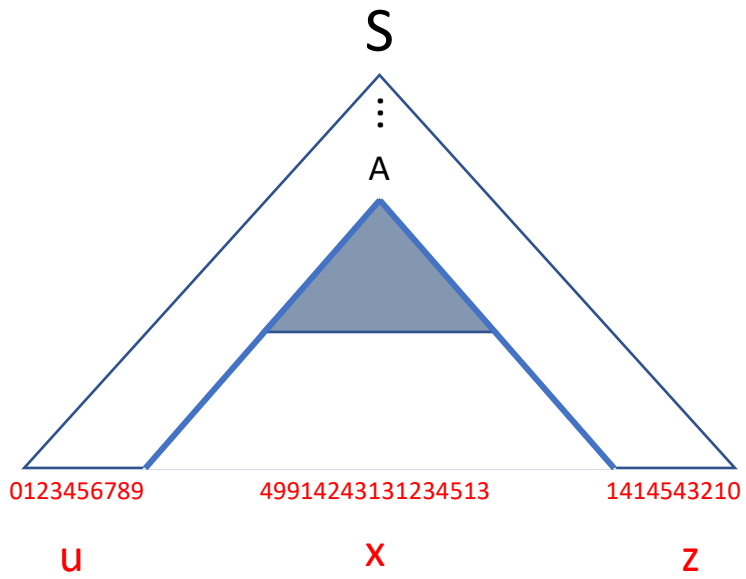
Bombeando para menos

- Então eu poderia ter uma palavra mais curta, fazendo a derivação “terminadora” mais em cima, palavra resultante: **uxz** (onde u,x e z são os pedaços das palavras)



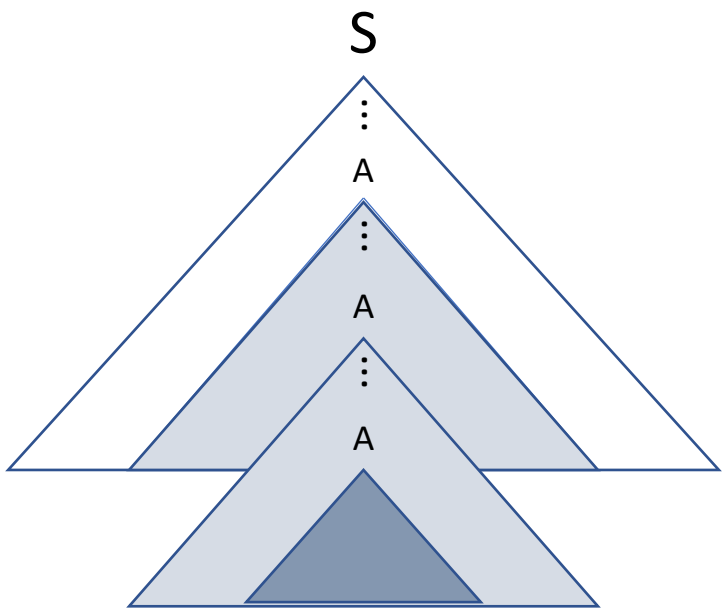
Bombeando para menos

- Então eu poderia ter uma palavra mais curta, fazendo a derivação “terminadora” mais em cima, palavra resultante: **uxz** (onde u,x e z são os pedaços das palavras)



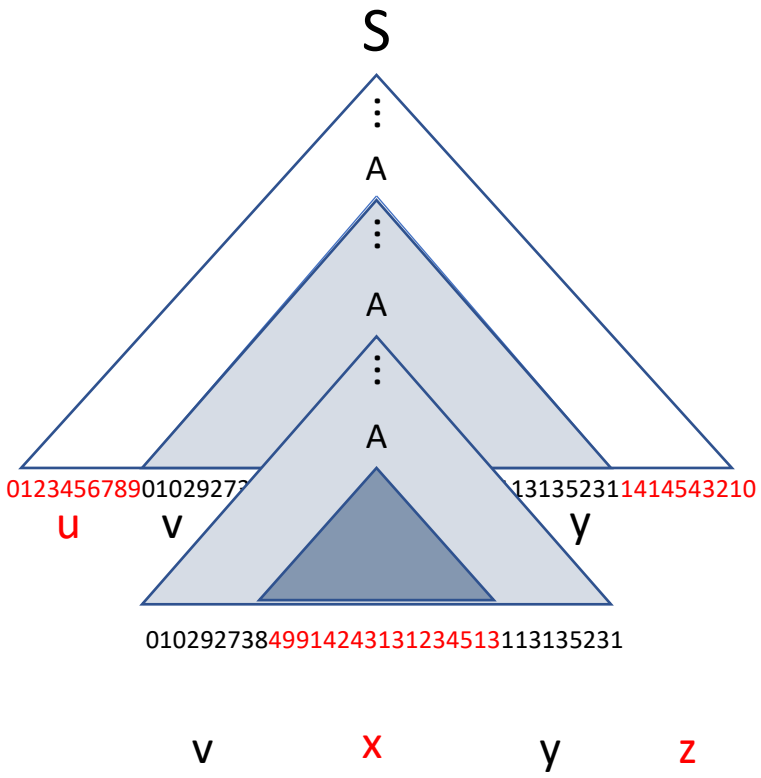
Bombeando para mais

- Ou então eu poderia ter uma palavra mais longa, fazendo a derivação “prolongadora” também da segunda vez



Bombeando para mais

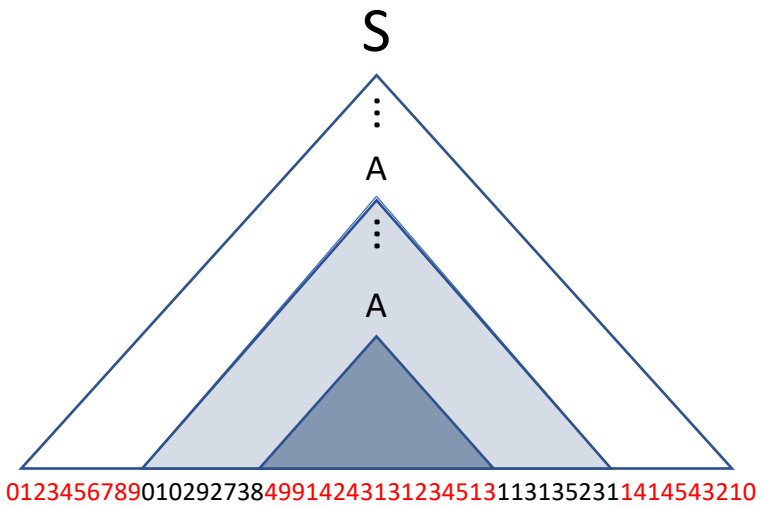
- Ou então eu poderia ter uma palavra mais longa, fazendo a derivação “prolongadora” também da segunda vez; palavra resultante: **uvvxyyz** (onde u, v, x, y e z são os pedaços das palavras)

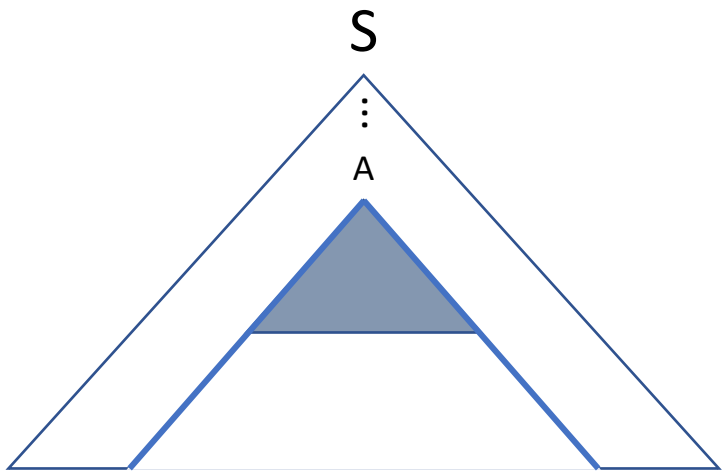


uvvxyyz

Bombeando para menos

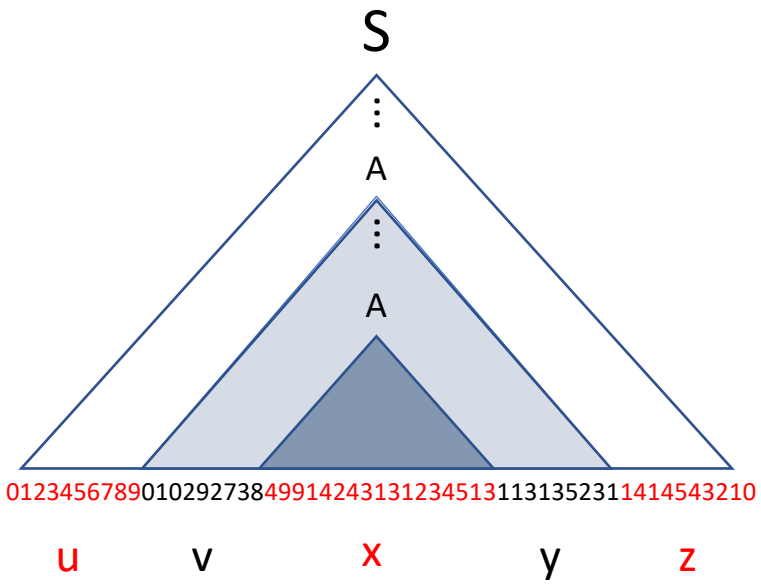
- Como a palavra mais curta se compara com a palavra original? Que pedaços deixaram de existir quando a palavra foi encurtada?

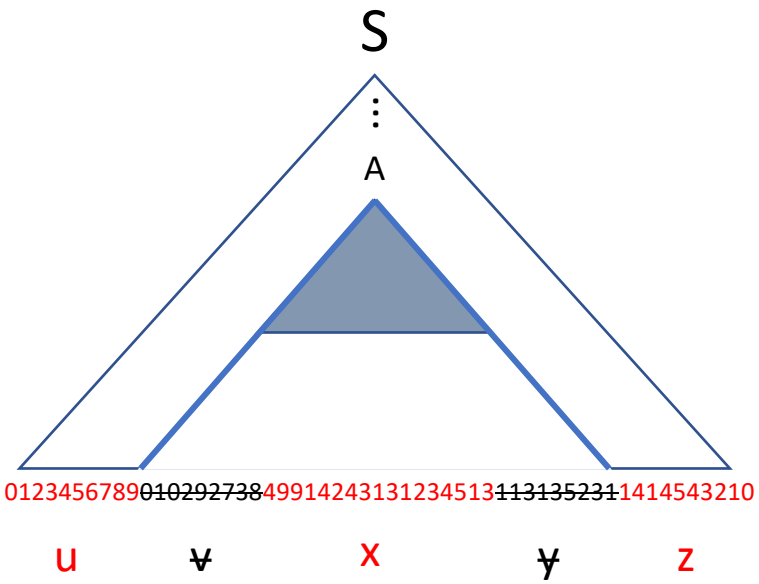




Bombeando para menos

- Como a palavra mais curta se compara com a palavra original? Que pedaços deixaram de existir quando a palavra foi encurtada?

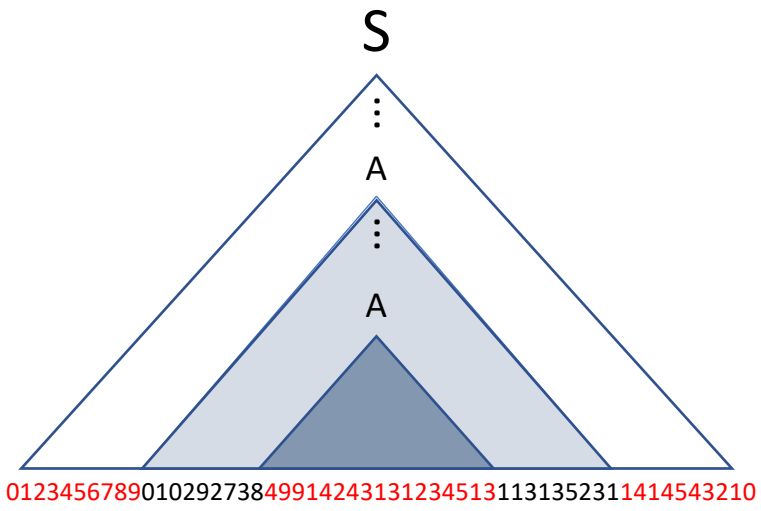


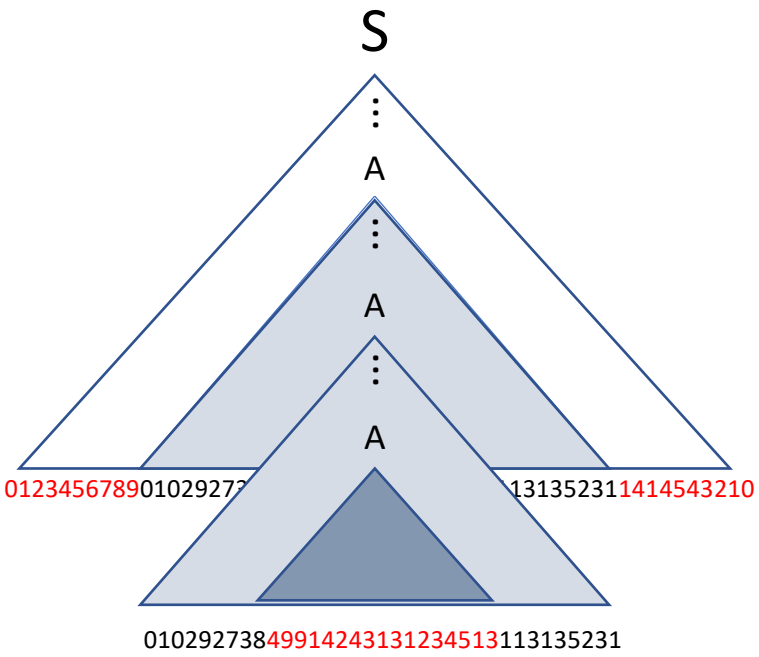


uvxyz → uxz

Bombeando para mais

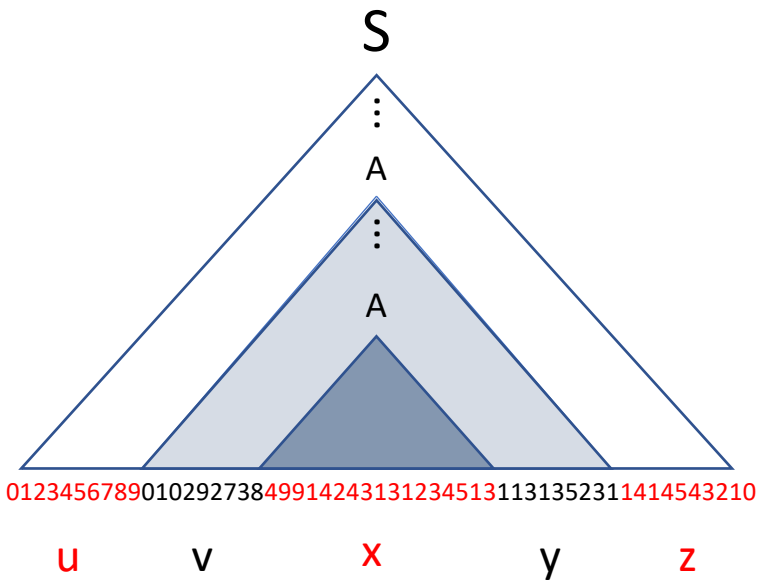
- Como a palavra mais longa se compara com a palavra original? De onde vieram os pedaços que passaram a existir?

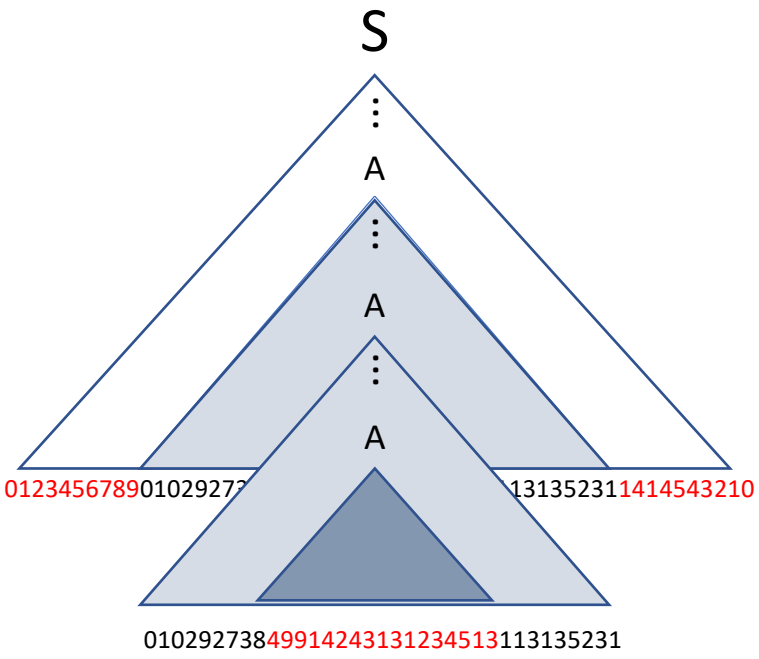




Bombeando para mais

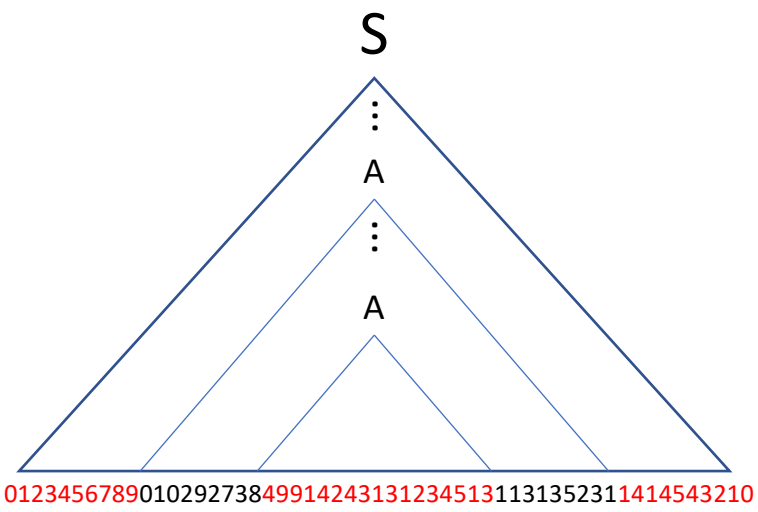
- Como a palavra mais longa se compara com a palavra original? De onde vieram os pedaços que passaram a existir?

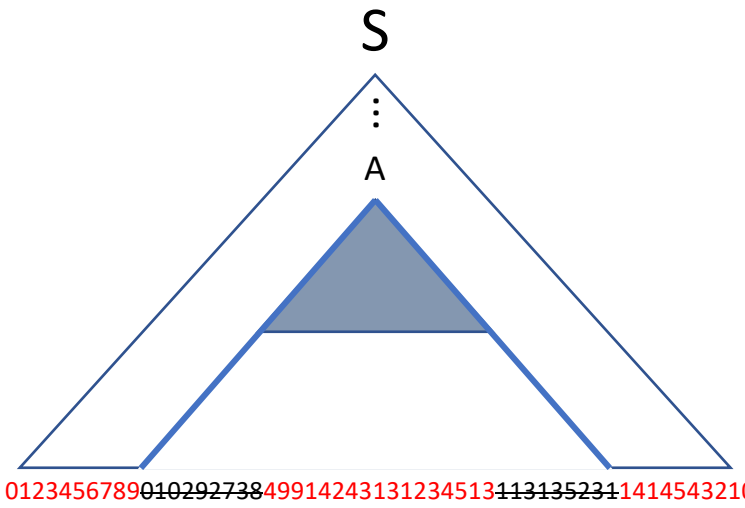


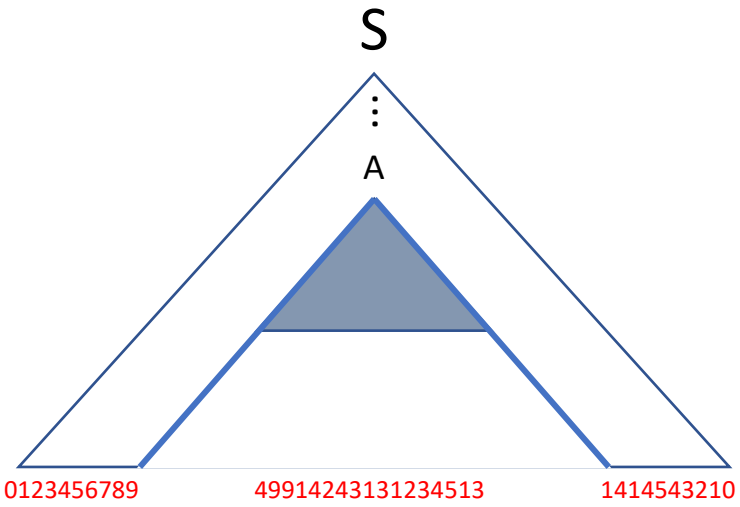


uvxyz → uvvxyyz

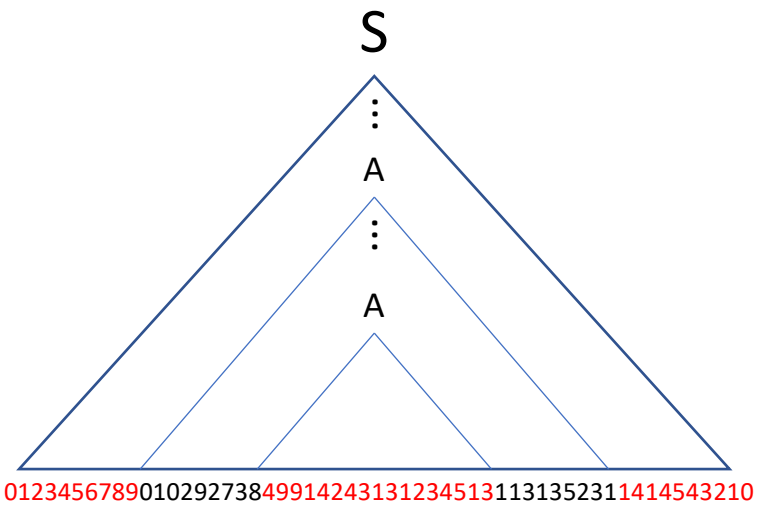
Então note que aqui um pedaço some...

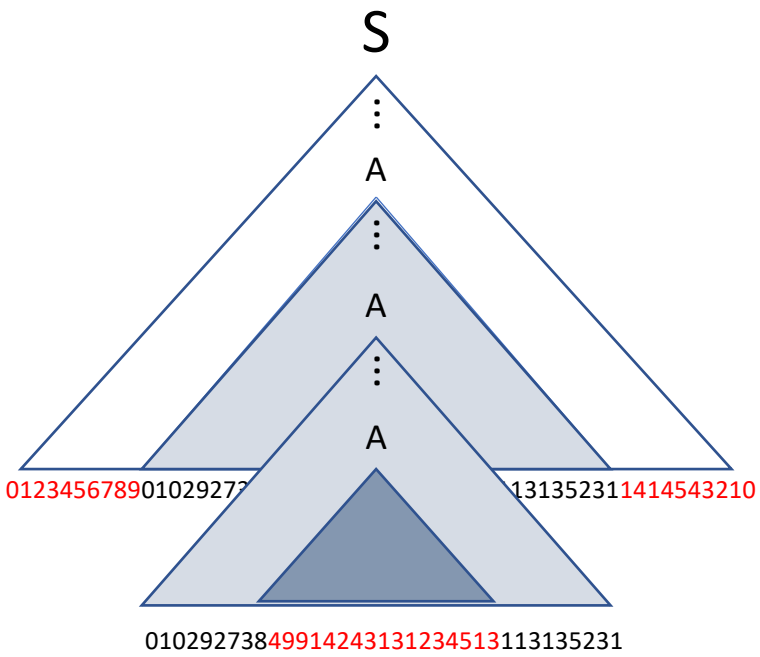






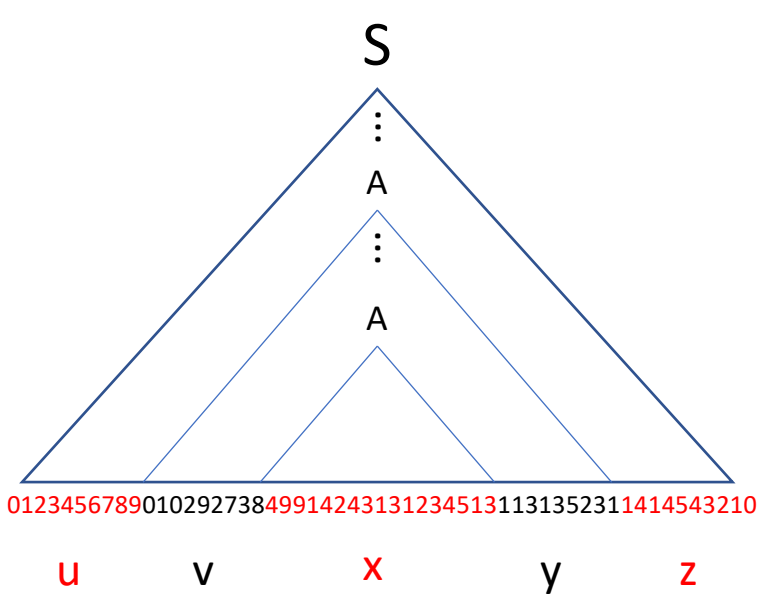
E aqui um pedaço se repete...

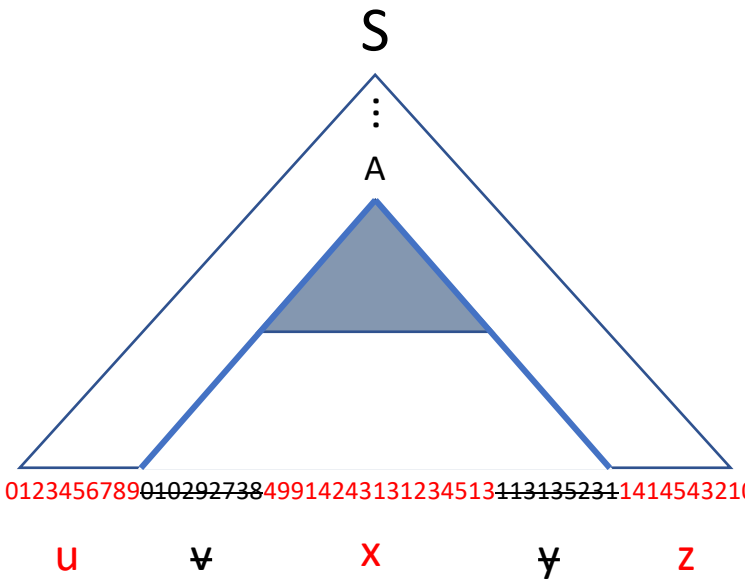


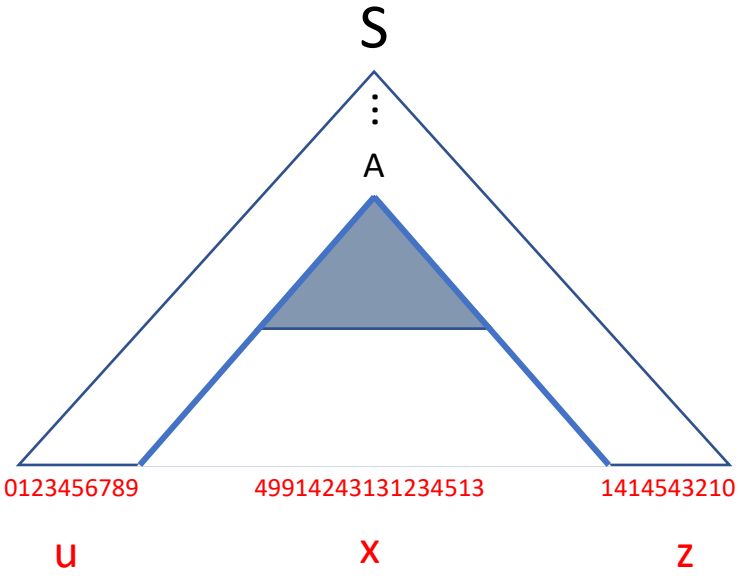


Por quê?

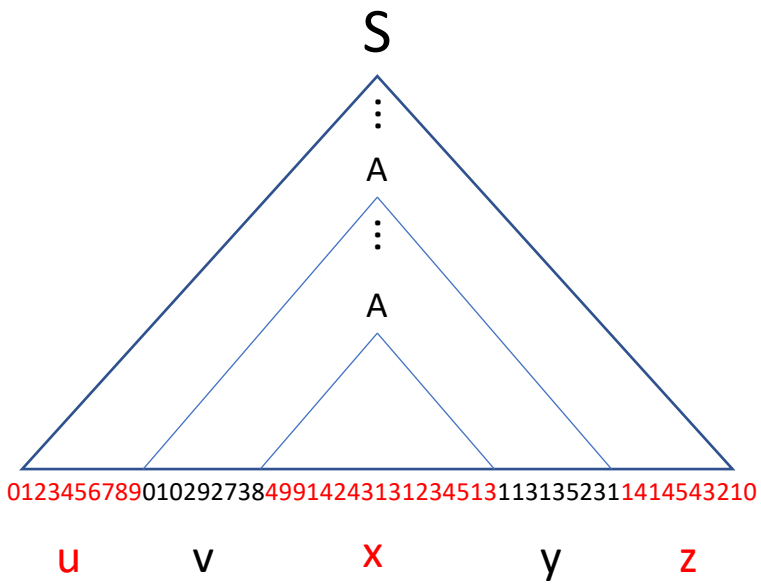
No primeiro caso a parte que some é a gerada pela antiga escolha, que não foi mais usada

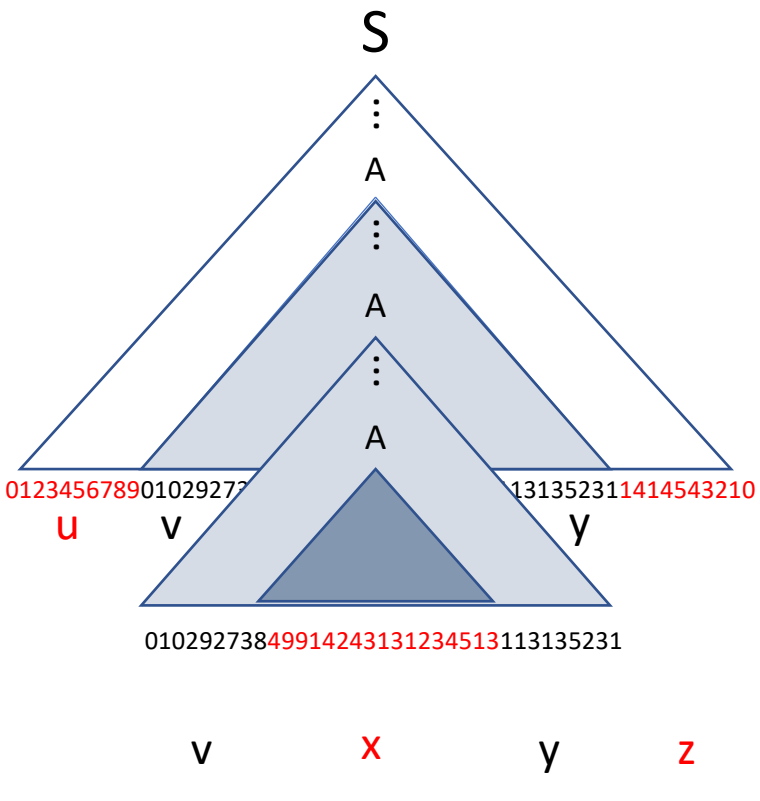






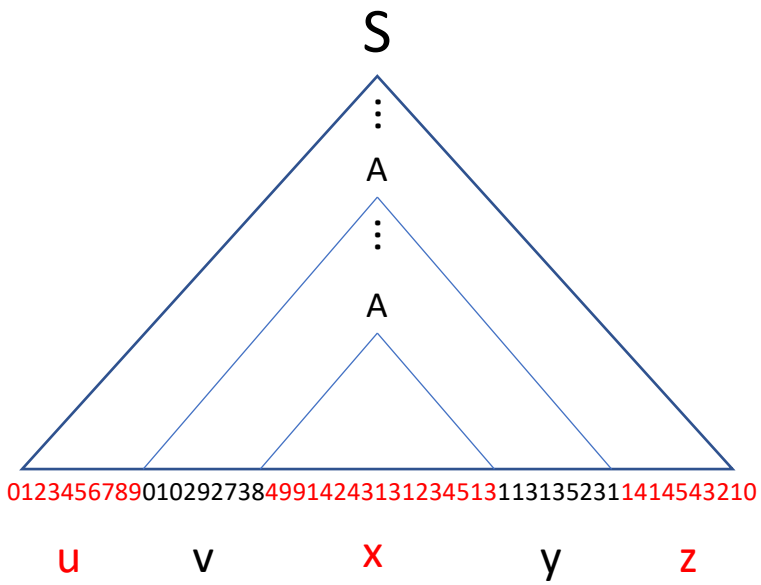
No segundo caso, a antiga escolha foi usada duas vezes, e por isso se repete

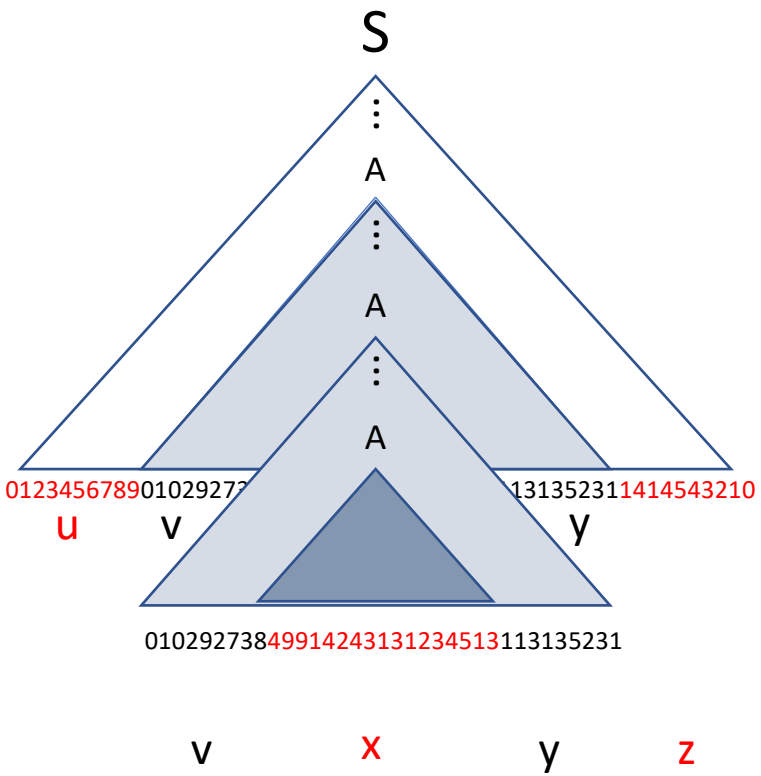




Quantas vezes posso fazer isso?

No segundo caso, a antiga escolha foi usada duas vezes, e por isso se repete





Quantas vezes posso fazer isso? Quantas eu quiser.

Lema do bombeamento para LLC

- Se uma linguagem L é livre de contexto, existe um número *p*, tal que...
- Qualquer palavra em L que seja maior que *p* tem uma árvore de derivação que tem uma repetição, ou seja, daria para gerar variações dessa palavra que precisam pertencer à linguagem
- Então, a palavra pode ser quebrada em cinco pedaços: s = u.v.x.y.z
  - Para $i \geq 0$, $u.v^i.x.y^i.z \in A$

Lema do bombeamento para LLC

- Se uma linguagem L é livre de contexto, existe um número *p*, tal que...
- Qualquer palavra em L que seja maior que *p* tem uma árvore de derivação que tem uma repetição, ou seja, daria para gerar variações dessa palavra que precisam pertencer à linguagem
- Então, a palavra pode ser quebrada em cinco pedaços: s = u.v.x.y.z
  - Para $i \geq 0$, $u.v^i.x.y^i.z \in A$
  - $|v.y| > 0$
  - $|v.x.y| \leq p$