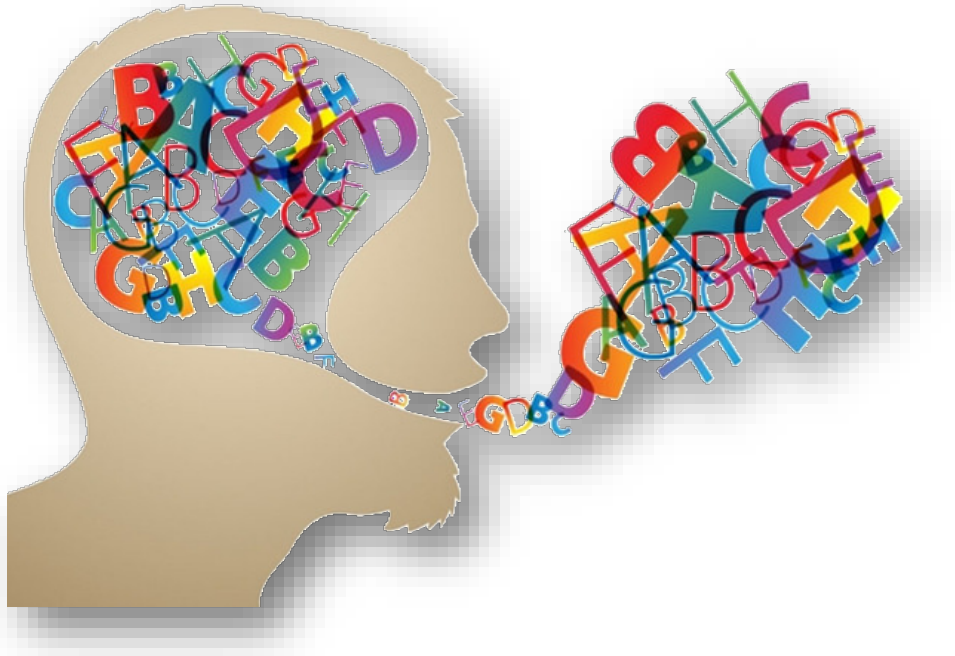
# Gramática Livre de Contexto (GLC)



Material extraído do livro e slides do Prof. Newton Vieira (http://dcc.ufmg.br/~nvieira)

# GRAMÁTICAS LIVRES DE CONTEXTO (GLC'S)

# BNF

- Exemplo de uma gramática livre de contexto em notação BNF (*Backus-Naur Form*) para definir parte da sintaxe de uma LP

⟨programa⟩ → ⟨declarações⟩ ; ⟨lista-de-cmds⟩ .
⟨lista-de-cmds⟩ → ⟨comando⟩ ; ⟨lista-de-cmds⟩
| λ
⟨comando⟩ → ⟨cmd-enquanto⟩
| ⟨cmd-se⟩
| ⟨cmd-atribuição⟩
| ...
⟨cmd-enquanto⟩ → **enquanto** ⟨exp-lógica⟩
**faça** ⟨lista-de-cmds⟩ **fimenquanto**
⟨cmd-se⟩ → **se** ⟨exp-lógica⟩ **então**
⟨lista-de-cmds⟩ ⟨senaoses⟩ ⟨senao⟩ **fimse**
⟨senaoses⟩ → **senão se** ⟨exp-lógica⟩ **então**
⟨lista-de-cmds⟩ ⟨senaoses⟩
| λ

Do lado esquerdo da regra possui apenas uma variável, do lado direito qualquer combinação de variáveis e terminais

# Gramática Livre de Contexto

- **Definição:** Uma gramática livre de contexto (GLC) é uma gramática $(V,\Sigma,R,P)$, em que cada regra tem a forma $X \rightarrow w$, onde $X \in V$ e $w \in (V \cup \Sigma)^*$

Repare que uma **Gramática Regular** é um caso especial de **Gramática Livre de Contexto**

- Exemplo: a linguagem não regular $\{0^n1^n \mid n \in N\}$ é gerada pela GLC $G = (\{P\}, \{0,1\}, R, P)$, onde R consta das duas regras

$$P \rightarrow 0P1 \mid \lambda$$

  - As palavras são geradas por *n* ($n \geq 0$) aplicações da regra $\mathbf{P \rightarrow 0P1}$, seguida de uma aplicação de $\mathbf{P \rightarrow \lambda}$

$$P \overset{n}{\Rightarrow} 0^nP1^n \Rightarrow 0^n1^n$$

# Mais Exemplos

- $L = \{ w \in \{0,1\}^* \mid w = w^R \}$

  $G = (\{P\}, \{0,1\}, R, P)$, onde R consta das 5 regras

  $$P \rightarrow 0P0 \mid 1P1 \mid 0 \mid 1 \mid \lambda$$

- $L = \{ w \in \{0,1\}^* \mid$ o número de 0s é igual ao número de 1s em w $\}$

  $G = (\{P\}, \{0,1\}, R, P)$, onde R consta das 3 regras

  $$P \rightarrow 0P1P \mid 1P0P \mid \lambda$$

# Ainda Outro Exemplo

- Seja a GLC ({E, T, F}, {t, +, *, (, )}, R, E), para expressões aritméticas, onde R consta das regras

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow ( E ) \mid t$$

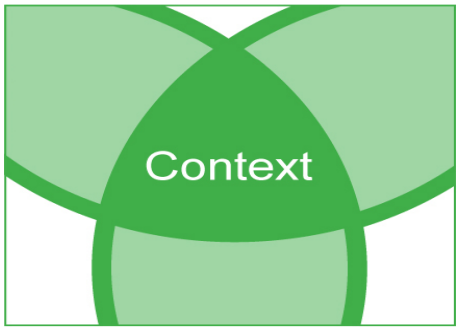
  - **E → E + T | T:** uma expressão artimética (E) é formada por um ou mais termos T's somados
  - **T → T * F | F:** um termo (T) é formado por um ou mais fatores F's multiplicados
  - **F → ( E ) | t:** um fator é um terminal t ou, recursivamente, uma expressão artimética entre parênteses

# Linguagem Livre de Contexto

- **Definição:** Uma linguagem é dita ser uma linguagem livre do contexto se existe uma gramática livre do contexto que a gera


Linguagem

Livre




de Contexto

Por que a linguagem é chamada de livre de contexto?

## Linguagem Livre de Contexto

Por que essas gramáticas são chamadas de **‘livres de contexto’** ?

*Porque todas as regras contêm apenas um símbolo no lado esquerdo -- e sempre que virmos esse símbolo enquanto fazemos uma derivação, estamos livres para substituí-lo com o que está no lado direito. Ou seja, o ‘contexto’ no qual ocorre um símbolo no lado esquerdo de uma regra não é importante -- podemos sempre usar a regra para reescrever enquanto fazemos uma derivação.*

# DERIVAÇÕES E AMBIGUIDADE

# Árvore de Derivações

- Uma árvore de derivação (AD) captura a essência de uma derivação, a história da obtenção de uma forma sentencial (que não depende da ordem de aplicação das regras)

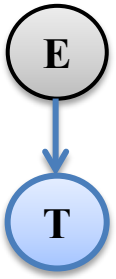
- **Definição:** Seja uma GLC G = (V, Σ, R, P ). Uma árvore de derivação (AD) é construída recursivamente como se segue
  a) Uma árvore com apenas o vértice de rótulo P é uma AD
  b) Se $X \in V$ é rótulo de uma folha f de uma AD, então
     i) se $X \rightarrow \lambda \in R$, então a árvore obtida acrescentando-se mais um vértice v com rótulo λ e uma aresta {f, v} é uma AD
     ii) se $X \rightarrow x_1x_2\ldots x_n \in R$, onde $x_1, x_2, \ldots, x_n \in (V \cup \Sigma)$, então a árvore obtida acrescentando-se mais *n* vértices $v_1, v_2, \ldots, v_n$ com rótulos $x_1, x_2, \ldots, x_n$, nessa ordem, e *n* arestas $\{f, v_1\}, \{f, v_2\}, \ldots, \{f, v_n\}$, é uma AD

# Exemplo

- Considere a gramática a seguir

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow ( E ) \mid t$$

- A AD para **t * (t + t)** pode ser construída passo a passo




Derivações:

$\mathbf{E \Rightarrow T}$ (regra $E \rightarrow T$)

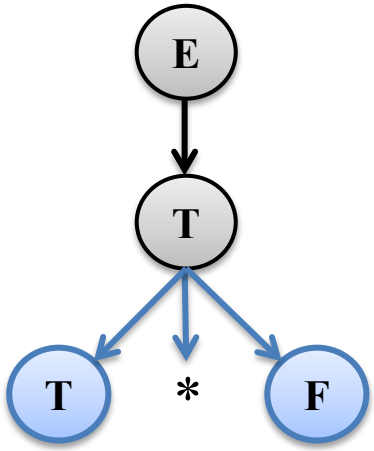
Qual é a próxima regra aplicável?

# Exemplo

- Considere a gramática a seguir

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow ( E ) \mid t$$

- A AD para **t * (t + t)** pode ser construída passo a passo




Derivações:

$E \Rightarrow \mathbf{T}$ (regra $E \rightarrow T$)

$\Rightarrow \mathbf{T * F}$ (regra $T \rightarrow T * F$)

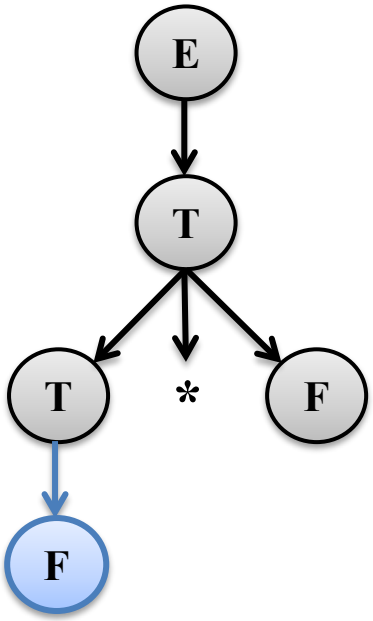
Qual regra se deve derivar agora?

# Exemplo

- Considere a gramática a seguir

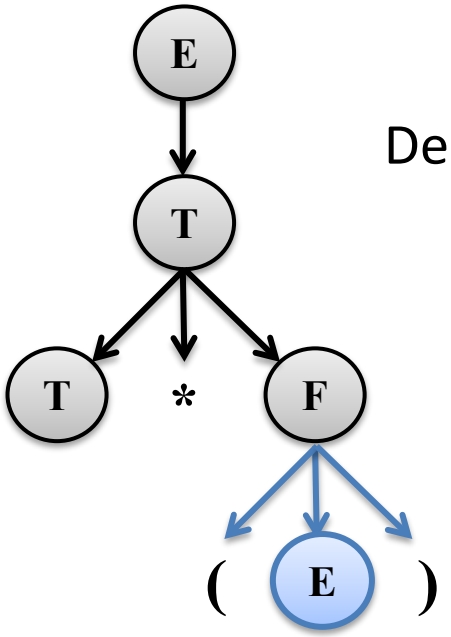
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow ( E ) \mid t$$

- A AD para **t * (t + t)** pode ser construída passo a passo




Derivações:

$E \Rightarrow T$
$\Rightarrow \mathbf{T} * F$
$\Rightarrow \mathbf{F} * \mathbf{F}$



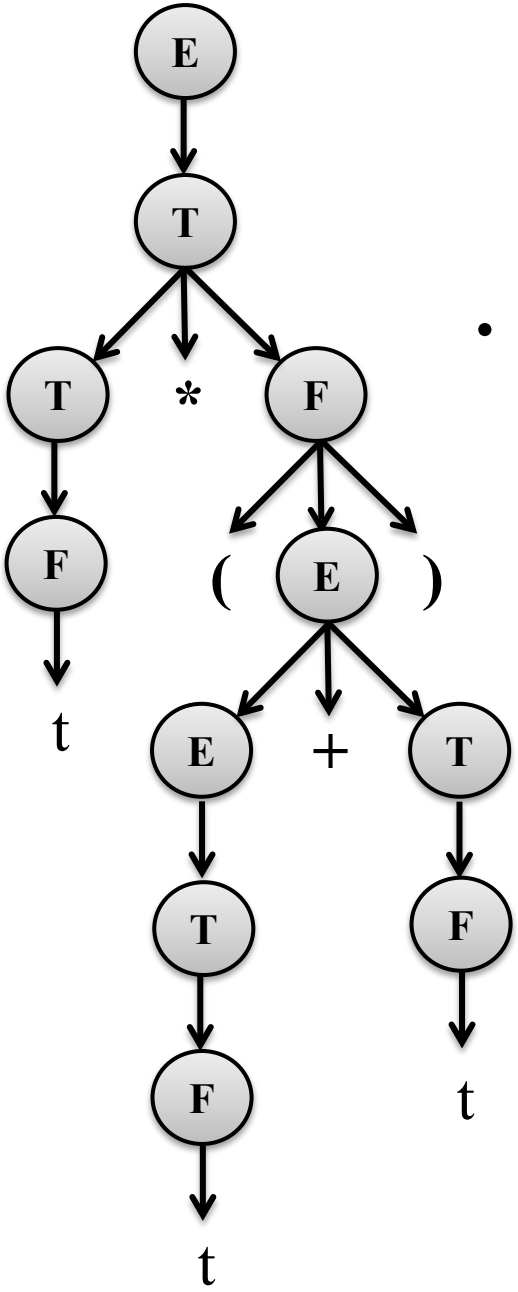

Derivações:

$E \Rightarrow T$
$\Rightarrow T * \mathbf{F}$
$\Rightarrow \mathbf{T} * \mathbf{(} \mathbf{E} \mathbf{)}$

Algumas regras podem ser alternadas na derivação e ainda sim gerar a mesma AD

# Exemplo

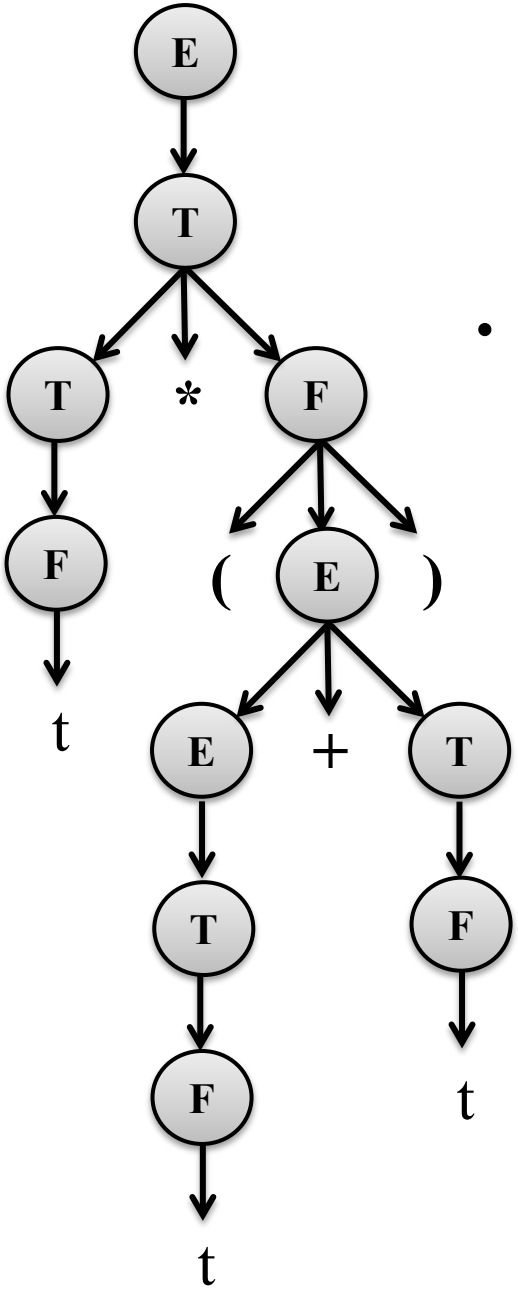
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow ( E ) \mid t$$




- A árvore de derivação completa para **t * (t + t)**

Derivações:

E ⇒ T (regra E → T)
⇒ T * F (regra T → T * F)
⇒ F * F (regra T → F)
⇒ t * F (regra F → t)
⇒ t * ( E ) (regra F → ( E ))
⇒ t * ( E + T ) (regra E → E + T)
⇒ t * ( T + T ) (regra E → T)
⇒ t * ( F + T ) (regra T → F)
⇒ t * ( t + T ) (regra F → t)
⇒ t * ( t + F ) (regra T → F)
⇒ t * ( t + t ) (regra F → t)




# Exemplo

- A árvore de derivação completa para **t * (t + t)**

Observações

- Número de passos da derivação é o número de vértices internos
- A estrutura da AD é normalmente utilizada para associar significado
- Mais de uma AD para w ⇒ mais de um significado para w

Pode existir mais de uma derivação para uma palavra **w** de uma gramática **G**?

# Ambiguidade

- **Definição:** Uma GLC é dita ambígua quando existe mais de uma AD para alguma sentença que ela gera

- Exemplo: uma GLC G = ({E}, {t, +, ∗, (, )}, R, E) de expressões aritméticas **ambíguas**, onde R consta das regras

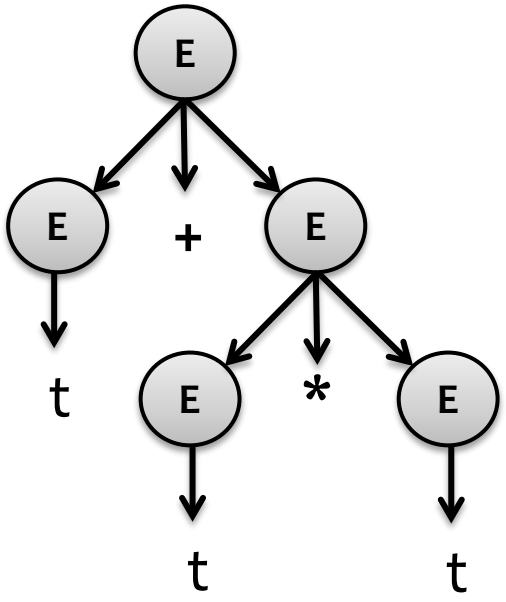
$$
\begin{aligned}
E \rightarrow\ & E + E \mid \\
& E * E \mid \\
& ( E ) \mid \\
& t
\end{aligned}
$$

Existe mais de uma derivação para a sentença **t+t*t**?

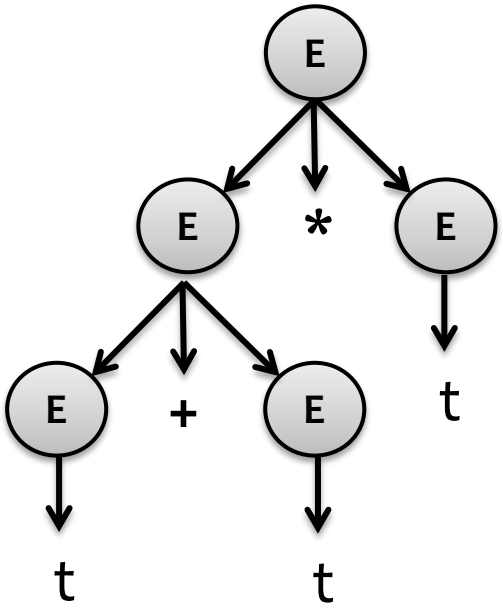
# Demonstrando a Ambiguidade

- A ambiguidade pode ser demostrada gerando duas árvores de derivação para uma sentença da linguagem para a gramática

- Exemplo: sentença **t+t*t** para a gramática do exemplo anterior




Significado: **t + (t * t)**




Significado: **(t + t) * t**

**Cuidado:** a ambiguidade é da gramática e não da linguagem que ela gera

# Dois Tipos de Derivações

- **Derivação mais à esquerda:** Uma derivação é dita mais à esquerda (DME) se em cada passo é expandida a variável mais à esquerda (pode-se usar o símbolo $\Rightarrow_E$)

- **Derivação mais à direita:** Uma derivação é dita mais à direta (DMD) se em cada passo é expandida a variável mais à direita (pode-se usar o símbolo $\Rightarrow_D$)

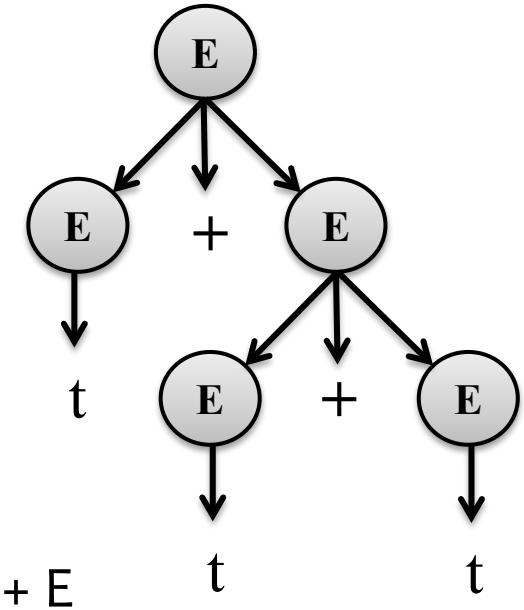
Existe uma única DME e uma única DMD correspondentes a uma AD

# Ambiguidades DME/DMD

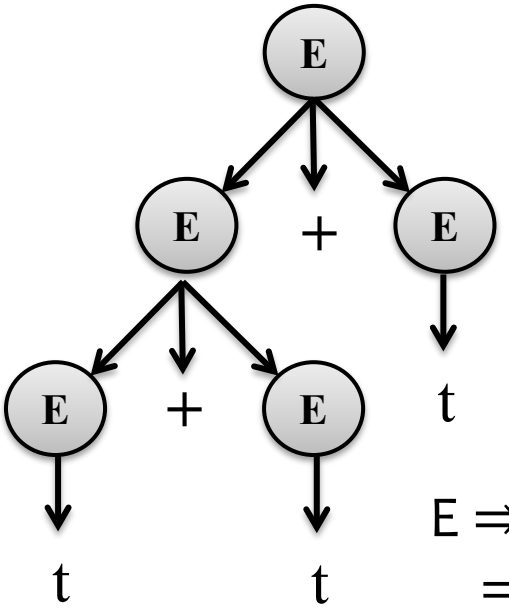
- Como existe uma única DME e uma única DMD correspondentes a uma AD, pode-se dizer que
  - uma GLC é ambígua se, e somente se, existe mais de uma DME para alguma sentença que ela gera
  - uma GLC é ambígua se, e somente se, existe mais de uma DMD para alguma sentença que ela gera

# Exemplo

- Para a mesma gramática de expressões aritméticas, pode-se mostrar usando somente DMD que a gramática é ambígua




$E \Rightarrow_D E + E$
$\Rightarrow_D E + E + E$
$\Rightarrow_D E + E + t$
$\Rightarrow_D E + t + t$
$\Rightarrow_D t + t + t$




$E \Rightarrow_D E + E$
$\Rightarrow_D E + t$
$\Rightarrow_D E + E + t$
$\Rightarrow_D E + t + t$
$\Rightarrow_D t + t + t$

# Linguagens Inerentemente Ambíguas

- Existem linguagens livres do contexto (LLC's) para as quais existem apenas gramáticas ambíguas, essas são chamadas de **linguagens inerentemente ambíguas**

- Exemplo: $L = \{ a^m b^n c^k \mid m = n \text{ ou } n = k \}$

Pode-se mostrar que qualquer GLC que gere tal linguagem terá mais de uma AD para $\mathbf{a^n b^n c^n}$

# Conclusão

- A detecção e remoção de ambiguidade é muito importante
  - Ex.: gramática para geração de um compilador para uma LP

**Cuidado:** o problema de determinar se uma GLC é ambígua é **indecidível**

- Uma linguagem pode ser gerada por inúmeras gramáticas, mas algumas gramáticas podem ser mais adequadas que outras dependendo do contexto

Existem técnicas para manipulação de GLC's que não alteram a linguagem gerada

# Manipulação de Gramáticas

- Uma gramática pode ser manipulada, sem alterar a linguagem que ela gere

  - Eliminando variáveis inúteis
  - Eliminando regras $\lambda$
  - Eliminando regras unitárias

# Variáveis Inúteis

- **Definição:** Seja uma GLC G = (V, Σ, R, P). Uma variável X ∈ V é dita ser uma variável **útil** se, e somente se, existem u,v ∈ $(V \cup \Sigma)^*$ e w ∈ $\Sigma^*$ tais que

$$P \overset{*}{\Rightarrow} uXv \overset{*}{\Rightarrow} w$$

- Exemplo: GLC G = (V, Σ, R, P), com as seguintes regras R

$$\begin{aligned} P &\rightarrow AB \mid a \\ B &\rightarrow B \mid b \\ C &\rightarrow c \end{aligned}$$

Existem variáveis inúteis nessa gramática?

# Exemplo

- Seja a GLC G = (V, Σ, R, P), onde R é formado pelas regras

$$P \rightarrow AB \mid a$$
$$B \rightarrow B \mid b$$
$$C \rightarrow c$$

  - As seguintes variáveis são inúteis
    - **C:** não existem u e v tais que $P \overset{*}{\Rightarrow} uCv$
    - **A:** não existe $w \in \Sigma^*$ tal que $A \overset{*}{\Rightarrow} w$
    - **B:** $P \overset{*}{\Rightarrow} uBv$ apenas para u = A e v = λ, e não existe $w \in \Sigma^*$ tal que $AB \overset{*}{\Rightarrow} w$

- Gramática equivalente sem símbolos inúteis

$$P \rightarrow a$$

# Eliminação de Variáveis Inúteis

- **Definição:** Seja uma GLC G tal que $L(G) \neq \emptyset$. Existe uma GLC, equivalente a G, sem variáveis inúteis

- Seja $G = (V, \Sigma, R, P)$ tal que $L(G) \neq \emptyset$. Uma GLC G'' equivalente a G, sem variáveis inúteis, pode ser construída em dois passos

  a) Obter $G' = (V', \Sigma, R', P)$, em que
     - $V' = \{ X \in V \mid X \Rightarrow_G^* w \text{ para algum } w \in \Sigma^* \}$
     - $R' = \{ r \in R \mid r \text{ não contém símbolos de } V - V' \}$

  b) Obter $G'' = (V'', \Sigma, R'', P)$, em que
     - $V'' = \{ X \in V' \mid P \Rightarrow_{G'}^* uXv \text{ para algum } u, v \in (V' \cup \Sigma)^* \}$
     - $R'' = \{ r \in R' \mid r \text{ não contém símbolos de } V' - V'' \}$

# Algoritmo: parte (a)

- Algoritmo para determinar **varíaveis que produzem sentenças:** $\{ X \in V \mid X \overset{*}{\Rightarrow} w \text{ para algum } w \in \Sigma^* \}$

**Entrada:** GLC $G = (V, \Sigma, R, P)$
**Saída:** $\mathbf{I_1} = \{ X \in V \mid X \Rightarrow w \text{ para algum } w \in \Sigma^* \}$

$I_1 \leftarrow \emptyset$
**repita**
    $N \leftarrow \{ X \notin I_1 \mid X \rightarrow z \in R \text{ e } z \in (I_1 \cup \Sigma)^* \}$
    $I_1 \leftarrow I_1 \cup N$
**até** $N = \emptyset$
**retorne** $I_1$

# Algoritmo: parte (b)

- Algoritmo para determinar **variáveis alcançáveis a partir de P**:
  $\{ X \in V \mid P \Rightarrow^{*} uXv \text{ para algum } u, v \in (V \cup \Sigma)^{*} \}$

**Entrada:** GLC $G = (V, \Sigma, R, P)$
**Saída:** $\mathbf{I_2} = \{ X \in V \mid P \Rightarrow^{*} uXv \text{ para algum } u, v \in (V \cup \Sigma)^{*} \}$

$I_2 \leftarrow \emptyset$
$N \leftarrow \{ P \}$
**repita**
    $I_2 \leftarrow I_2 \cup N$
    $N \leftarrow \{ Y \notin I_2 \mid X \rightarrow uYv \text{ para algum } X \in N \text{ e } u,v \in (V \cup \Sigma)^{*} \}$
**até** $N = \emptyset$
**retorne** $I_2$

# Exemplo

- Seja a gramática G = ({A, B, C, D, E, F}, {0, 1}, R, A}), onde R contém as regras:

$$A \rightarrow ABC \mid AEF \mid BD$$
$$B \rightarrow B0 \mid 0$$
$$C \rightarrow 0C \mid EB$$
$$D \rightarrow 1D \mid 1$$
$$E \rightarrow BE$$
$$F \rightarrow 1F1 \mid 1$$

Aplicando o algoritmo (a)

**V' = { B, D, F, A }**

$$A \rightarrow BD$$
$$B \rightarrow B0 \mid 0$$
$$D \rightarrow 1D \mid 1$$
$$F \rightarrow 1F1 \mid 1$$

Aplicando o algoritmo (b)

**V'' = { A, B, D }**

$$A \rightarrow BD$$
$$B \rightarrow B0 \mid 0$$
$$D \rightarrow 1D \mid 1$$

# Eliminação de uma Regra

- Uma regra da forma $X \rightarrow w$, onde X não é a variável de partida, pode ser eliminada simulando sua aplicação em todos os contextos possíveis: para cada ocorrência de X do lado direito de uma regra, substitui-se por w

- **Teorema:** Seja uma GLC $G = (V, \Sigma, R, P)$. Seja $X \rightarrow w \in R$, $X \neq P$. Seja a GLC $G' = (V, \Sigma, R', P)$, onde R' é obtido assim
  a) para cada regra de R em que X não ocorre do lado direito, exceto $X \rightarrow w$, coloque-a em R'
  b) para cada regra de R da forma $Y \rightarrow x_1X_1x_2X_2...X_nx_{n+1}$, com pelo menos uma ocorrência de X do lado direito, com $n \geq 1$ e $x_i \in [(V - \{X\}) \cup \Sigma]^*$, coloque em R' todas as regras da forma $Y \rightarrow x_1Y_1x_2Y_2...Y_nx_{n+1}$, sendo que cada $Y_j$ pode ser X ou w, com exceção da regra $X \rightarrow w$

G' é equivalente a G

# Algoritmo

- Algoritmo para eliminação de uma regra

Entrada: (1) uma GLC $G = (V, \Sigma, R, P)$, e
(2) uma regra $X \to w \in R$, $X \neq P$.
Saída: uma GLC $G'$ equivalente a $G$, sem a regra $X \to w$.
$R' \leftarrow \emptyset$;
**para** cada regra $Y \to z \in R$ **faça**
    **para** cada forma de escrever $z$ como $x_1 X x_2 \ldots X x_{n+1}$ **faça**
        $R' \leftarrow R' \cup \{Y \to x_1 w x_2 \ldots w x_{n+1}\}$
    **fimpara**;
**fimpara**;
retorne $G' = (V, \Sigma, R' - \{X \to w\}, P)$.

# Exemplo

- Seja a gramática G = ({P, A, B}, {a, b, c}, R, P}), onde R contém as regras

$$P \rightarrow ABA$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bBc \mid \lambda$$

Como é a derivação de **aa** em G?

- A GLC G' = ({P, A, B}, {a, b, c}, R, P}), equivalente a G, pode ser obtida eliminando-se a regra A → a

$$P \rightarrow ABA \mid ABa \mid aBA \mid aBa$$
$$A \rightarrow aA \mid aa$$
$$B \rightarrow bBc \mid \lambda$$

Como é a derivação de **aa** em G'?

# Variável Anulável

- **Definição:** uma variável X é **anulável** em uma GLC G se, e somente se, $X \Rightarrow_G \lambda$

- Algoritmo para determinar variáveis anuláveis

**Entrada:** GLC G = (V, Σ, R, P)
**Saída:** $A = \{ X \in V \mid X \Rightarrow \lambda^* \}$

$A \leftarrow \emptyset$
**repita**
    $N \leftarrow \{ Y \notin A \mid Y \rightarrow z \in R \text{ e } z \in A^* \}$
    $A \leftarrow A \cup N$
**até** $N = \emptyset$
**retorne** A

# Eliminação de Regras λ

- Seja G = (V, Σ, R, P). Seja G' = (V, Σ, R', P) em que R' é obtido assim

a) para cada regra de R cujo lado direito não contém variável anulável, exceto regra λ, coloque-a em R'

b) para cada regra de R da forma $Y \rightarrow x_1X_1x_2X_2...X_nx_{n+1}$, sendo cada $X_i$ uma variável anulável, com $n \geq 1$ e cada $x_i$ sem variáveis anuláveis, coloque em R' todas as regras da forma $Y \rightarrow x_1Y_1x_2Y_2...Y_nx_{n+1}$, em que cada $Y_j$ pode ser $X_j$ ou λ, com exceção da regra λ

c) Se P for anulável, coloque $P \rightarrow \lambda$ em R'

L(G) = L(G') e sua única regra λ é $P \rightarrow \lambda$ (se houver)

# Algoritmo

- Algoritmo para eliminação de regras λ

Entrada: uma GLC $G = (V, \Sigma, R, P)$;
Saída: uma GLC $G'$ equivalente a $G$, sem regras $\lambda$, exceto $P \rightarrow \lambda$.
$\mathcal{A} \leftarrow$ variáveis anuláveis de $G$;
$R' \leftarrow \emptyset$;
**para** cada regra $X \rightarrow w \in R$ **faça**
    **para** cada forma de escrever $w$ como $x_1 Y_1 x_2 \ldots Y_n x_{n+1}$, com $Y_1 \ldots Y_n \in \mathcal{A}$ **faça**
        $R' \leftarrow R' \cup \{X \rightarrow x_1 x_2 \ldots x_{n+1}\}$
    **fimpara**;
**fimpara**;
retorne $G' = (V, \Sigma, R' - \{X \rightarrow \lambda \mid X \neq P\}, P)$.

# Exemplo

- Seja a gramática G = ({P, A, B, C}, {a, b, c}, R, P}), onde R contém as regras

$$P \rightarrow APB \mid C$$
$$A \rightarrow AaaA \mid \lambda$$
$$B \rightarrow BBb \mid C$$
$$C \rightarrow cC \mid \lambda$$

  - Obtendo o conjunto de variáveis anuláveis

$$\mathcal{A} = \{ A, C, P, B \}$$

  - Eliminando as variáveis anuláveis

$$P \rightarrow APB \mid AP \mid AB \mid PB \mid A \mid B \mid C \mid \lambda$$
$$A \rightarrow AaaA \mid aaA \mid Aaa \mid aa$$
$$B \rightarrow BBb \mid Bb \mid b \mid C$$
$$C \rightarrow cC \mid c$$

A regra $P \rightarrow P$ foi descartada por motivos óbvios

# Variáveis Encadeadas

- **Definição:** Seja uma gramática G = (V, Σ, R, P). Diz-se que uma variável Z ∈ V é encadeada a uma variável X ∈ V se Z = X ou se existe uma sequência de regras $X \rightarrow Y_1$, $Y_1 \rightarrow Y_2$, ..., $Y_n \rightarrow Z$ em R, n ≥ 0; quando n = 0, tem-se apenas a regra $X \rightarrow Z$. Ao conjunto de todas as variáveis encadeadas a X é dado o nome de enc(X).

- Uma GLC equivalente a G = (V, Σ, R, P), sem regras unitárias, é G' = (V, Σ, R', P), em que

$$R' = \{ X \rightarrow w \mid \text{existe } Y \in enc(X) \text{ tal que } Y \rightarrow w \in R \text{ e } w \notin V \}$$

# Algoritmo

- Algoritmo para obter as variáveis encadeadas de uma variável

**Entrada:** (1) GLC $G = (V, \Sigma, R, P)$
(2) variável $X \in V$
**Saída:** enc(X)

$U \leftarrow \emptyset$
$N \leftarrow \{ X \}$
**repita**
$U \leftarrow U \cup N$
$N \leftarrow \{ Y \notin U \mid Z \rightarrow Y \in R \text{ para algum } Z \in N \}$
**até** $N = \emptyset$
**retorne** U

# Exemplo

- Relembrando a GLC para expressões aritméticas

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid t$$

  - Os conjuntos enc(X) para cada variável $X \in V$

$$enc(E) = \{ E, T, F \}$$
$$enc(T) = \{ T, F \}$$
$$enc(F) = \{ F \}$$

  - Eliminando as regras unitárias

$$E \rightarrow E + T \mid T * F \mid (E) \mid t$$
$$T \rightarrow T * F \mid (E) \mid t$$
$$F \rightarrow (E) \mid t$$

# Interação entre os Métodos de Eliminação

- Ao se aplicar vários tipos de eliminações em sequência, alguns tipos de regras já eliminados podem reaparecer

- Exemplos
  a) Ao se eliminar regras λ podem aparecer regras unitárias

     **Ex.:** GLC com regras $A \rightarrow BC$ e $B \rightarrow \lambda$

  b) Ao se eliminar regras unitárias podem aparecer regras λ

     **Ex.:** GLC com $P \rightarrow \lambda$ (P: símbolo de partida) e a regra $A \rightarrow P$

  c) Ao se eliminar regras λ podem aparecer variáveis inúteis

     **Ex.:** o do item (a), caso $B \rightarrow \lambda$ seja a única regra de B

  d) Ao se eliminar regras unitárias podem aparecer var. inúteis

     **Ex.:** GLC que contém $A \rightarrow B$ e B não aparece do lado direito de nenhuma outra regra (B torna-se inútil)

Ao se eliminar variáveis inúteis, não podem aparecer novas regras

# Consistência das Eliminações

- A seguinte sequência de eliminações para uma GLC G = (V, Σ, R, P) garante sua consistência

  1) Adicionar a regra P' → P se P for recursivo
  2) Eliminar regras λ
  3) Eliminar regras unitárias
  4) Eliminar variáveis inúteis

- Essa sequência produz uma GLC equivalente cujas regras são da seguinte forma

$$P \rightarrow \lambda \quad \text{se } \lambda \in L(G)$$
$$X \rightarrow a \quad \text{para } a \in \Sigma$$
$$X \rightarrow w \quad \text{para } |w| \geq 2$$