Construção de Compiladores
Período Especial
Aula 16: Análise Sintática Descendente (Top-Down)

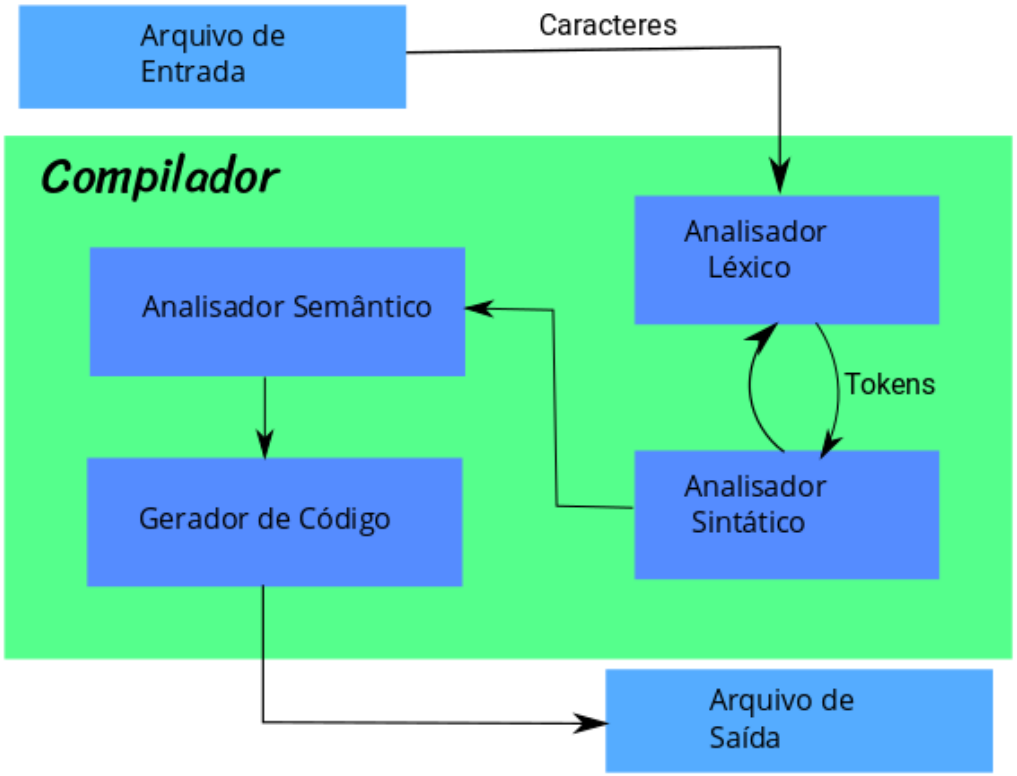
Bruno Müller Junior

Departamento de Informática
UFPR

2020

Introdução

- Compilador é composto por várias ferramentas.



Ferramentas

- Compilador é composto por várias ferramentas.
- Neste ponto da disciplina, já foi possível observar que a ferramenta que rege todo o processo é o analisador sintático.
- Assim, dada uma gramática "basta"construir o analisador sintático para reconhecer se uma sentença pertence ou não a aquela gramática.
- Analisador semântico e gerador de código são implementados como "ações semânticas"dentro do analisador sintático

Ferramentas

- Até o momento, só consideramos o uso das ferramentas flex e bison para desenvolver o compilador.
- Em outras palavras: só foi mostrado como construir um compilador na linguagem "C".
- Mas, se for necessário implementar um compilador para outra linguagem que não tenha flex e bison?
- O objetivo desta aula é apresentar um mecanismo genérico de implementação de um analisador sintático para uma gramática dada utilizando a abordagem Top-Down.
- Não será apresentado COMO construir uma gramática, que é alvo da área de computação denominada "linguagens formais e autômatos".

Analisadores Sintáticos Top-Down

- Esta categoria de analisadores sintáticos tem as seguintes características:
  - lêem a entrada da esquerda para a direita (*left to right parsing*);
  - constróem a árvore de derivação de cima para baixo substituindo sempre o terminal mais à esquerda (*leftmost derivation*)

Ferramentas

- Para escrever um programa capaz fazer a análise sintática Top-Down, existem várias abordagens:
  - Tentar todas as possibilidades (algoritmo da força bruta). Complexidade combinatorial;
  - Adaptar a gramática para utilizar um Analisador Sintático preditivo que tem complexidade linear;
  - Utilizar compilador de compiladores disponíveis nas linguagens de programação, como por exemplo (Javacc).
- Vamos nos concentrar na adaptação do algoritmo da força bruta.

- Também conhecido como Analisador Sintático com Retrocessos (*backtracking*);
- Tenta construir todas as árvores possíveis onde os símbolos de entrada consigam ser pendurados;
- Não é útil na prática, mas uma análise detalhada auxilia na compreensão do processo como um todo.
  - Algoritmo recursivo: Coloque S como raiz da árvore de derivação;
  - Seja X a variável mais à esquerda;
  - Selecione uma produção do tipo $X \rightarrow ABC$
  - Se houverem alguma, pendure;
  - Se não houver, descarte a última produção.
  - Reaplique o algoritmo;

Força Bruta

$G_1 = \{$

(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

Força Bruta

$$G_1 = \{ \quad \textcircled{0} \quad E' \rightarrow E\#$$
$$\textcircled{1} \quad E \rightarrow T + E$$
$$\textcircled{2} \quad E \rightarrow T$$
$$\textcircled{3} \quad T \rightarrow F * T$$
$$\textcircled{4} \quad T \rightarrow F$$
$$\textcircled{5} \quad F \rightarrow a$$
$$\textcircled{6} \quad F \rightarrow b$$
$$\textcircled{7} \quad F \rightarrow (E)\}$$

$\alpha =$ a + b #

- Aumenta a gramática (# = fim de arquivo)

Força Bruta

$$G_1 = \{ \quad (0)\ E' \to E\# \quad (1)\ E \to T + E \quad (2)\ E \to T \quad (3)\ T \to F * T \quad (4)\ T \to F \quad (5)\ F \to a \quad (6)\ F \to b \quad (7)\ F \to (E)\}$$

$\alpha =$ a + b #

- Obtém primeiro token: a

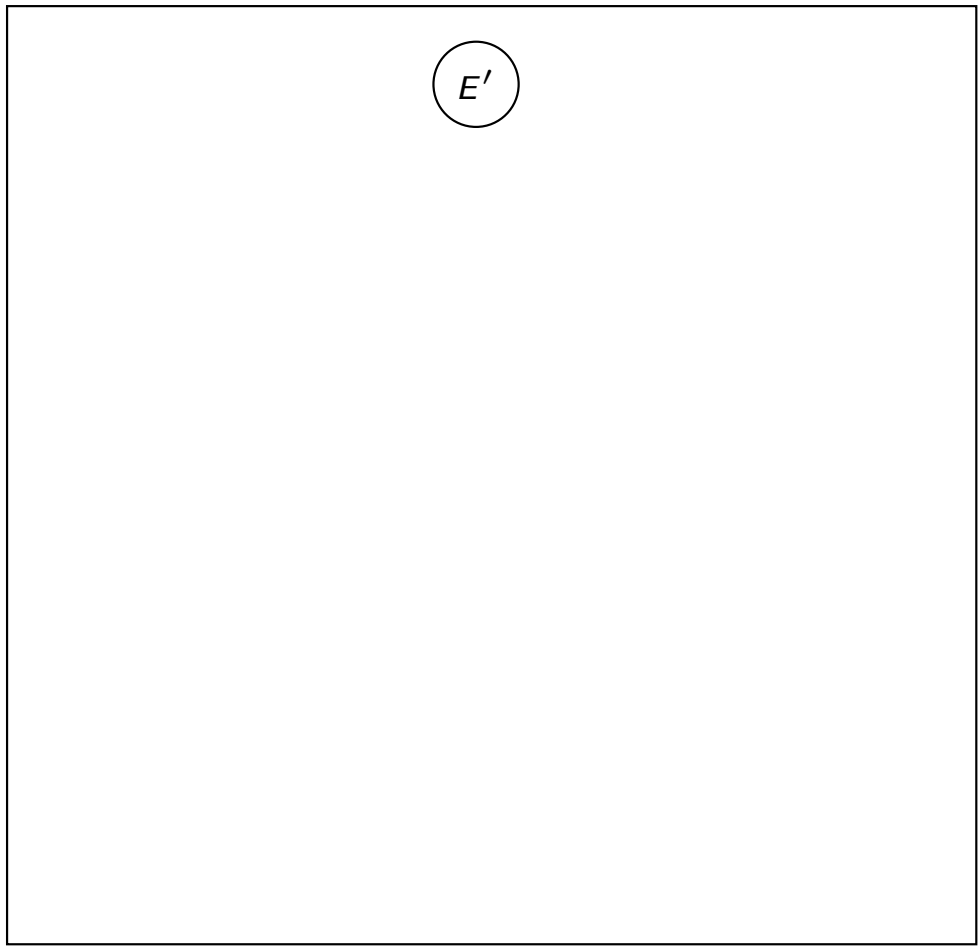
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura regras que derivam de E'
- só tem uma: regra (0)

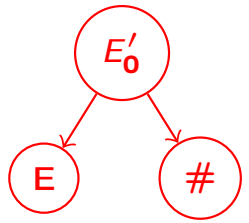


Força Bruta

$$G_1 = \{ \quad (0)\ E' \to E\# \quad (1)\ E \to T + E \quad (2)\ E \to T \quad (3)\ T \to F * T \quad (4)\ T \to F \quad (5)\ F \to a \quad (6)\ F \to b \quad (7)\ F \to (E)\}$$

$\alpha =$ a + b #

- Anota a derivação e indica qual regra usou: E'_0



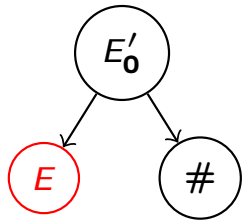
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;



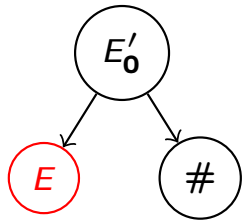
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: (1) e (2).



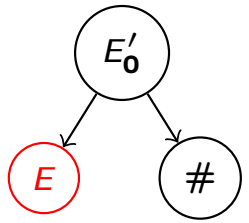
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: (1) e (2).
- Tenta uma por vez.

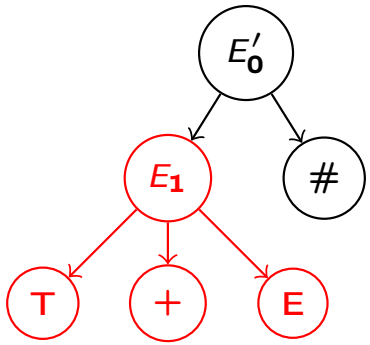


Força Bruta

$$G_1 = \{ \quad (0)\ E' \to E\# \quad (1)\ E \to T + E \quad (2)\ E \to T \quad (3)\ T \to F * T \quad (4)\ T \to F \quad (5)\ F \to a \quad (6)\ F \to b \quad (7)\ F \to (E)\}$$

$\alpha =$ a + b #

- Deriva usando regra (1).



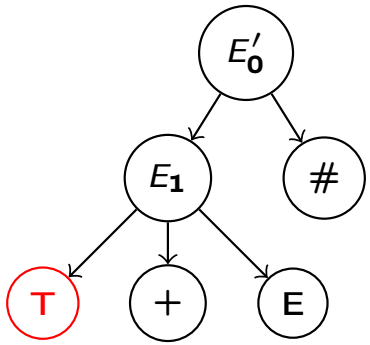
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;



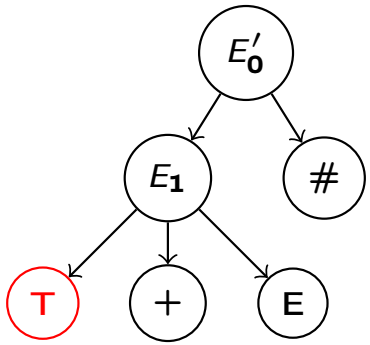
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: 3 e 4.



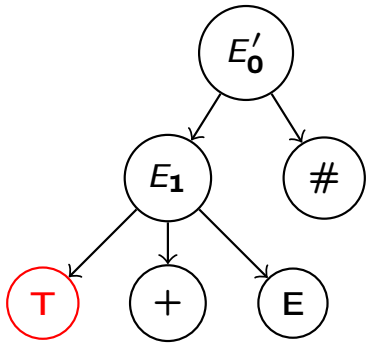
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: (3) e (4).
- Tenta uma por vez.



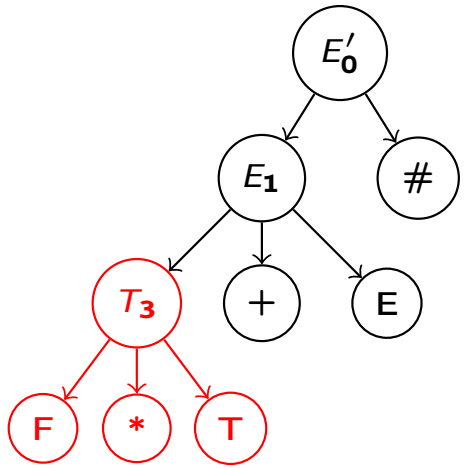
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Deriva usando regra (3).



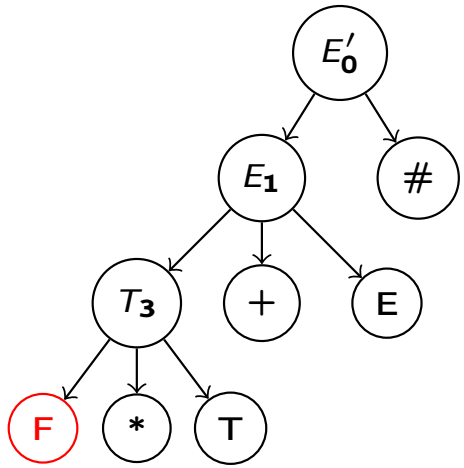
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;



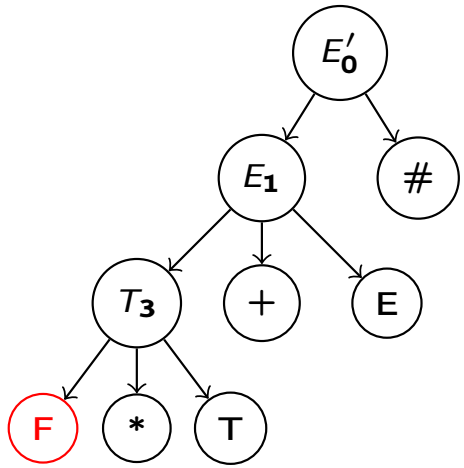
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: 5, 6 e 7.



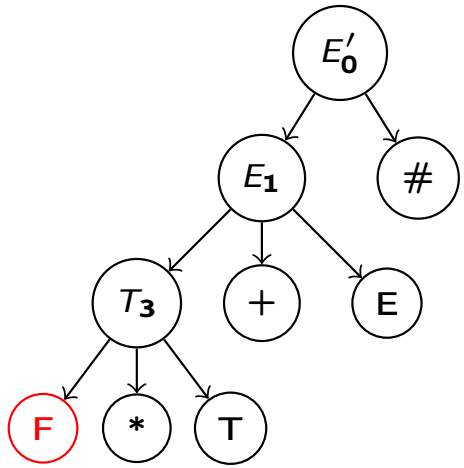
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- Seleciona regras que podem derivar daquela variável: (5), (6) e (7).
- Tenta uma por vez



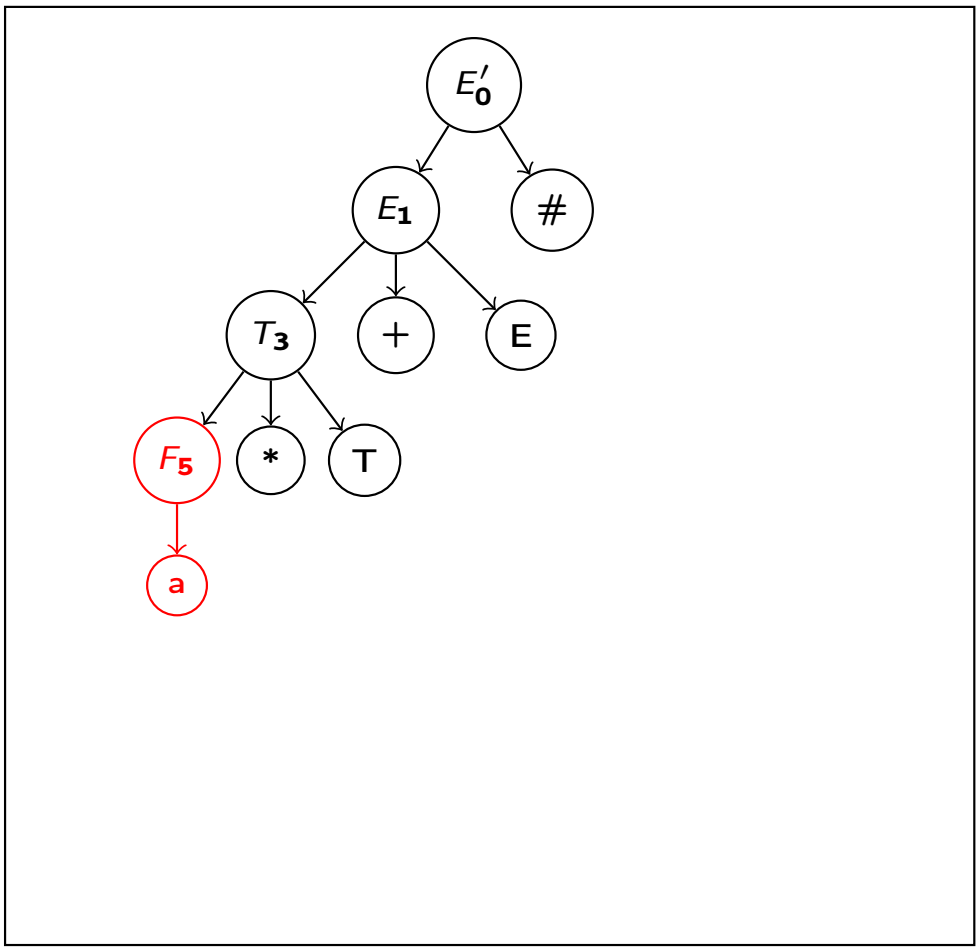
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Deriva usando regra (5).



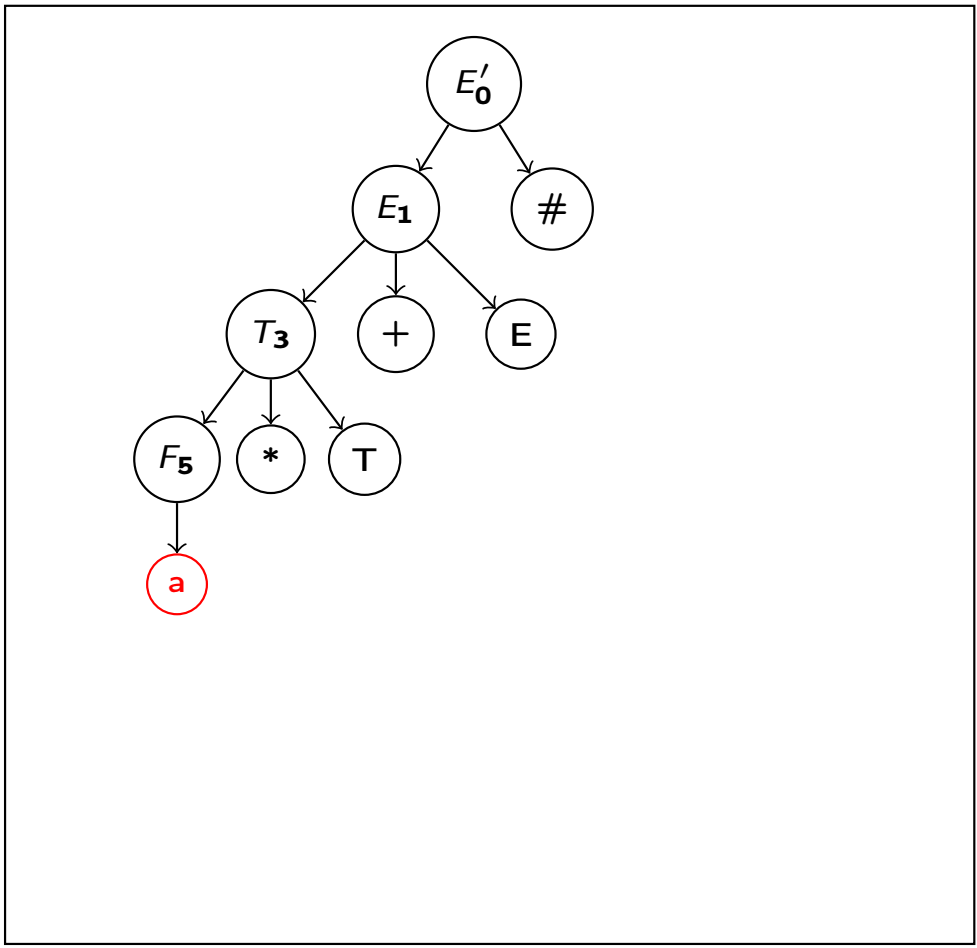
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;

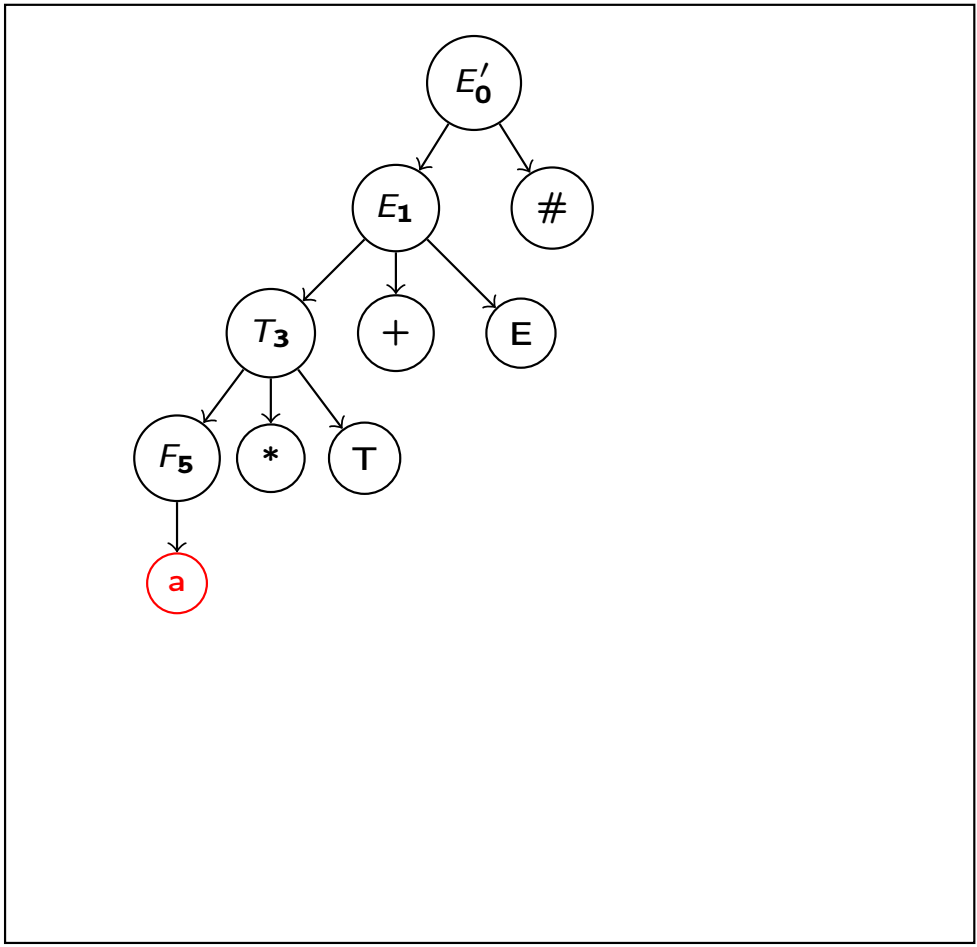


Força Bruta

$$
\begin{aligned}
G_1 = \{ \quad & (0) \quad E' \rightarrow E\# \\
& (1) \quad E \rightarrow T + E \\
& (2) \quad E \rightarrow T \\
& (3) \quad T \rightarrow F * T \\
& (4) \quad T \rightarrow F \\
& (5) \quad F \rightarrow a \\
& (6) \quad F \rightarrow b \\
& (7) \quad F \rightarrow (E)\}
\end{aligned}
$$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- É um vértice que representa um terminal;



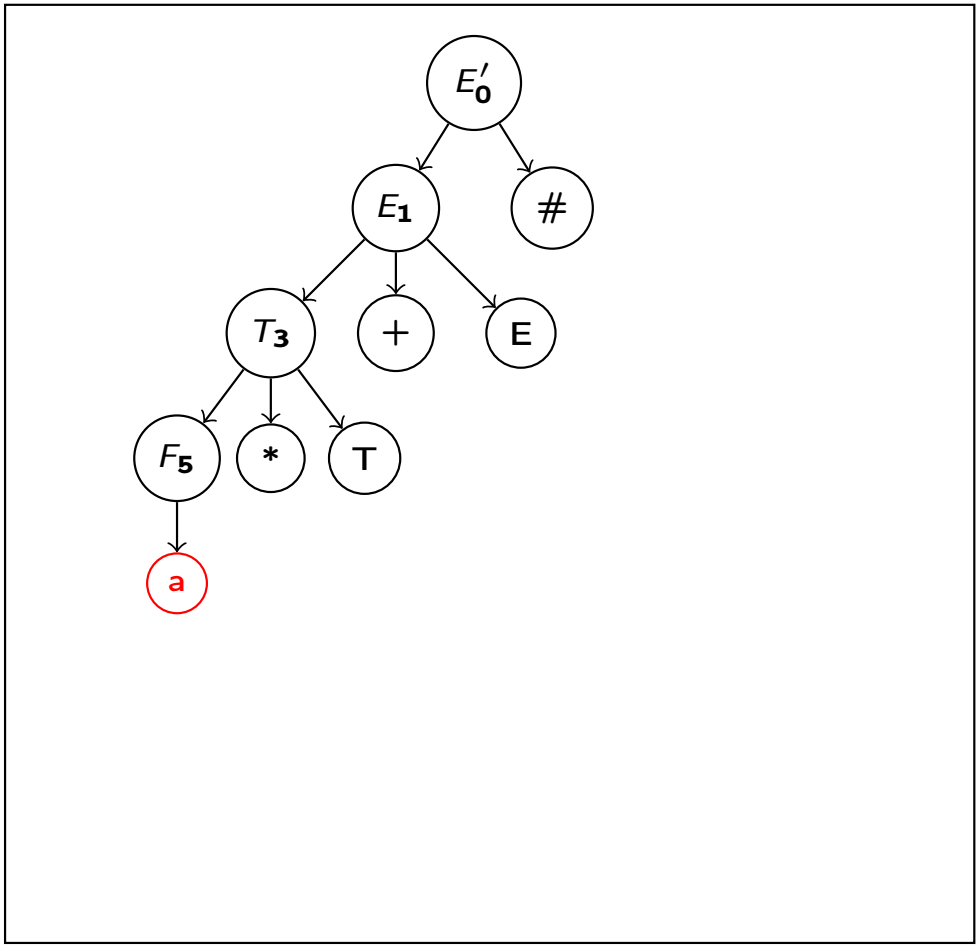
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Procura inorder a primeiro vértice não usado;
- É um vértice que representa um terminal;
- Se for igual ao token, retira da entrada e pendura na árvore;

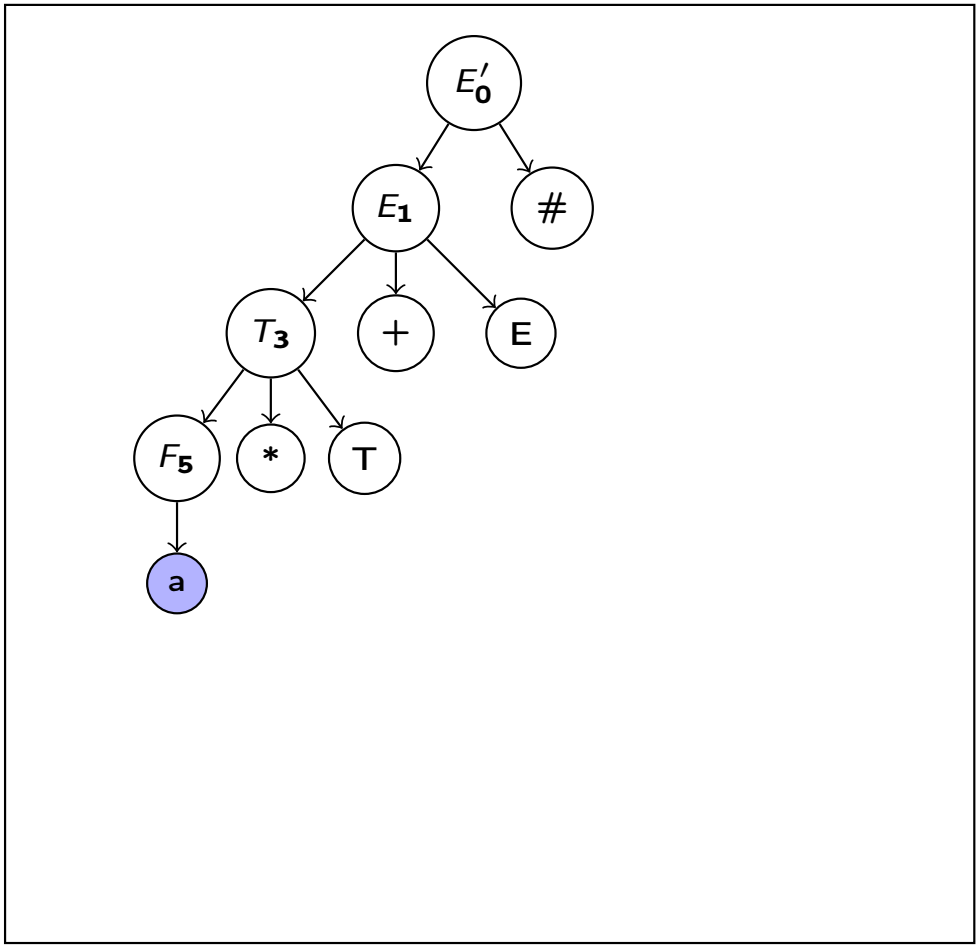


Força Bruta

$$G_1 = \{$$

0. $E' \rightarrow E\#$
1. $E \rightarrow T + E$
2. $E \rightarrow T$
3. $T \rightarrow F * T$
4. $T \rightarrow F$
5. $F \rightarrow a$
6. $F \rightarrow b$
7. $F \rightarrow (E)\}$

$\alpha =$ + b #

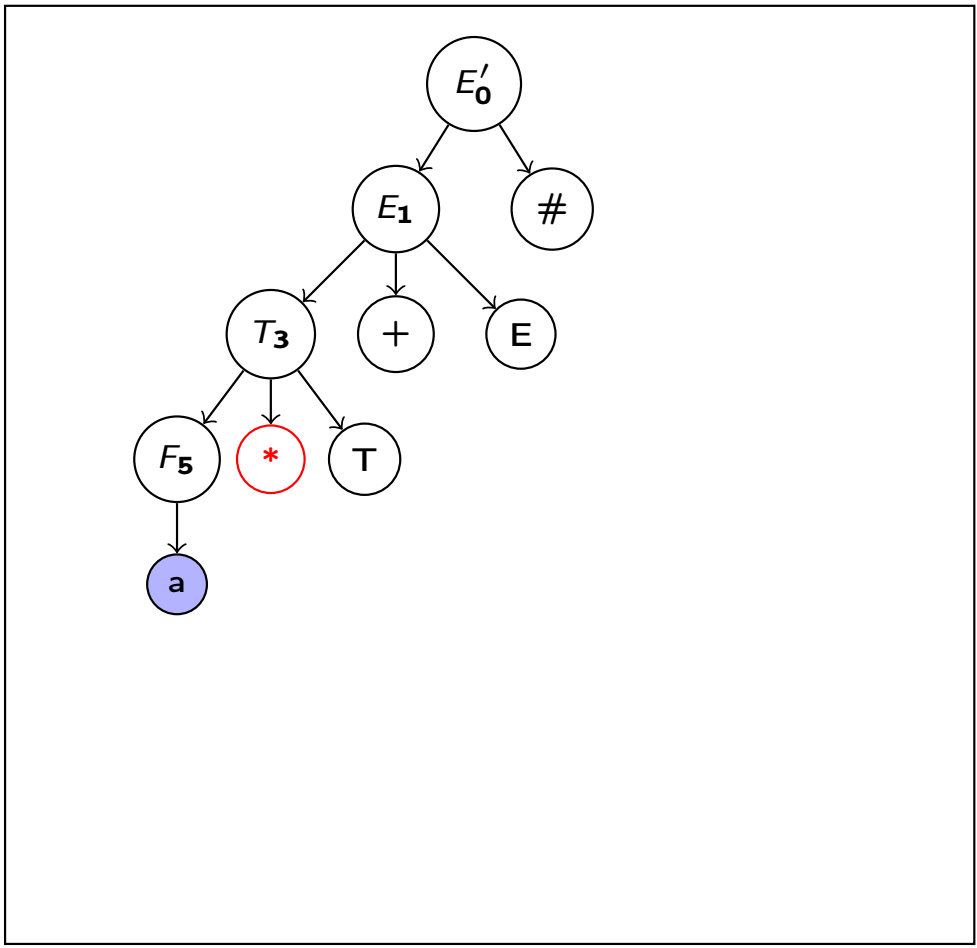


Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

- $\alpha =$ + b #
- Procura inorder a primeiro vértice não usado;



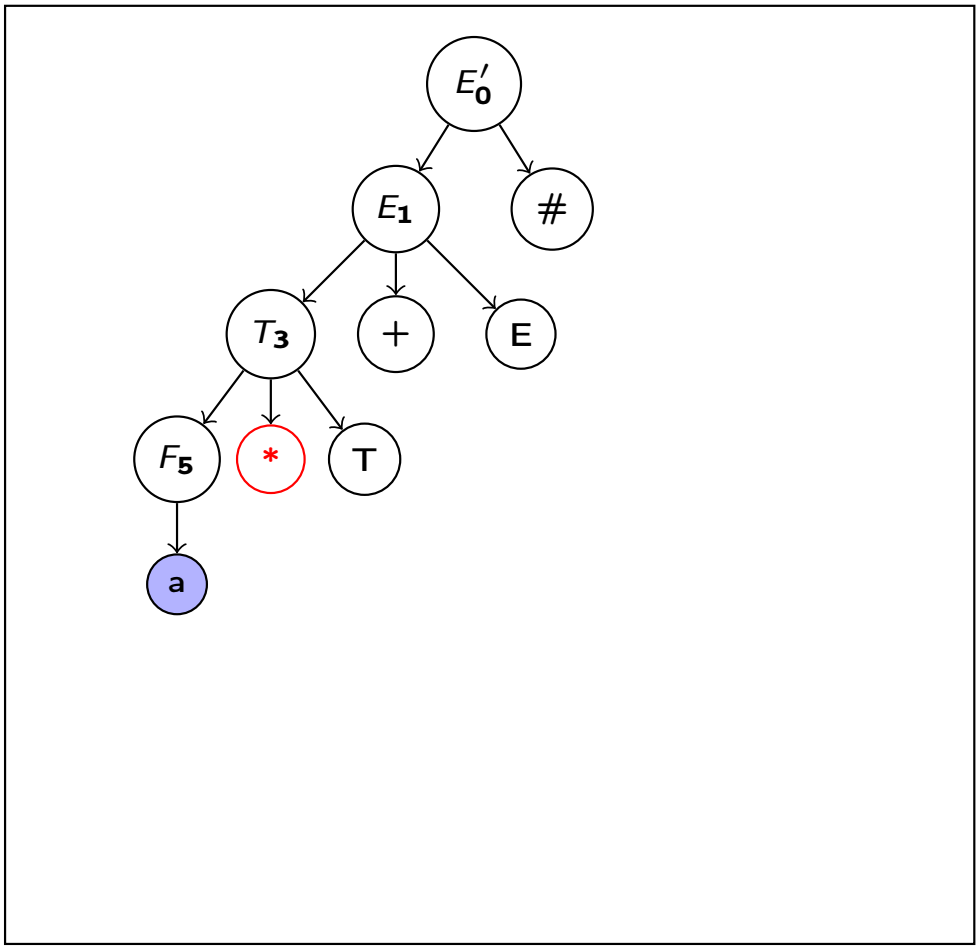
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ + b #

- Procura inorder a primeiro vértice não usado;
- É um vértice que representa um terminal ((*));

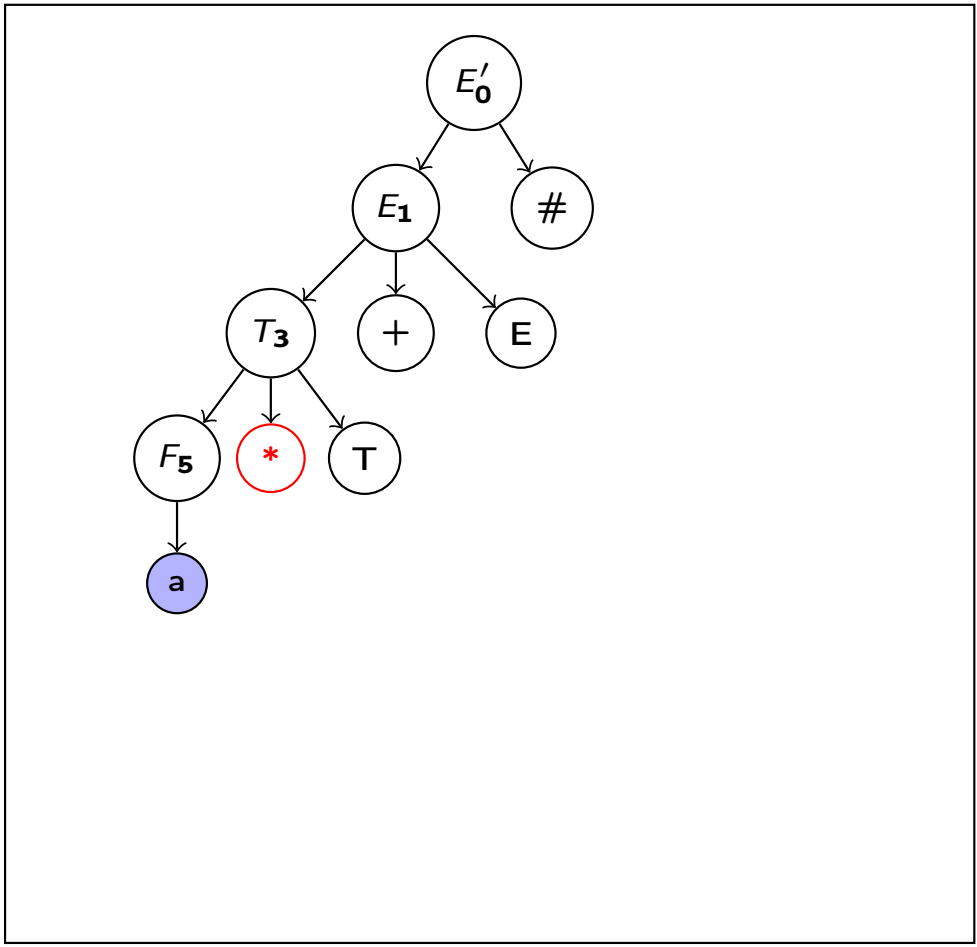


Força Bruta

$$G_1 = \{ \quad \textcircled{0}\ E' \to E\# \quad \textcircled{1}\ E \to T + E \quad \textcircled{2}\ E \to T \quad \textcircled{3}\ T \to F * T \quad \textcircled{4}\ T \to F \quad \textcircled{5}\ F \to a \quad \textcircled{6}\ F \to b \quad \textcircled{7}\ F \to (E)\}$$

$\alpha =$ + b #

- Procura inorder a primeiro vértice não usado;
- É um vértice que representa um terminal ($\textcircled{*}$);
- Não é igual ao token, volta um passo (retrocesso);



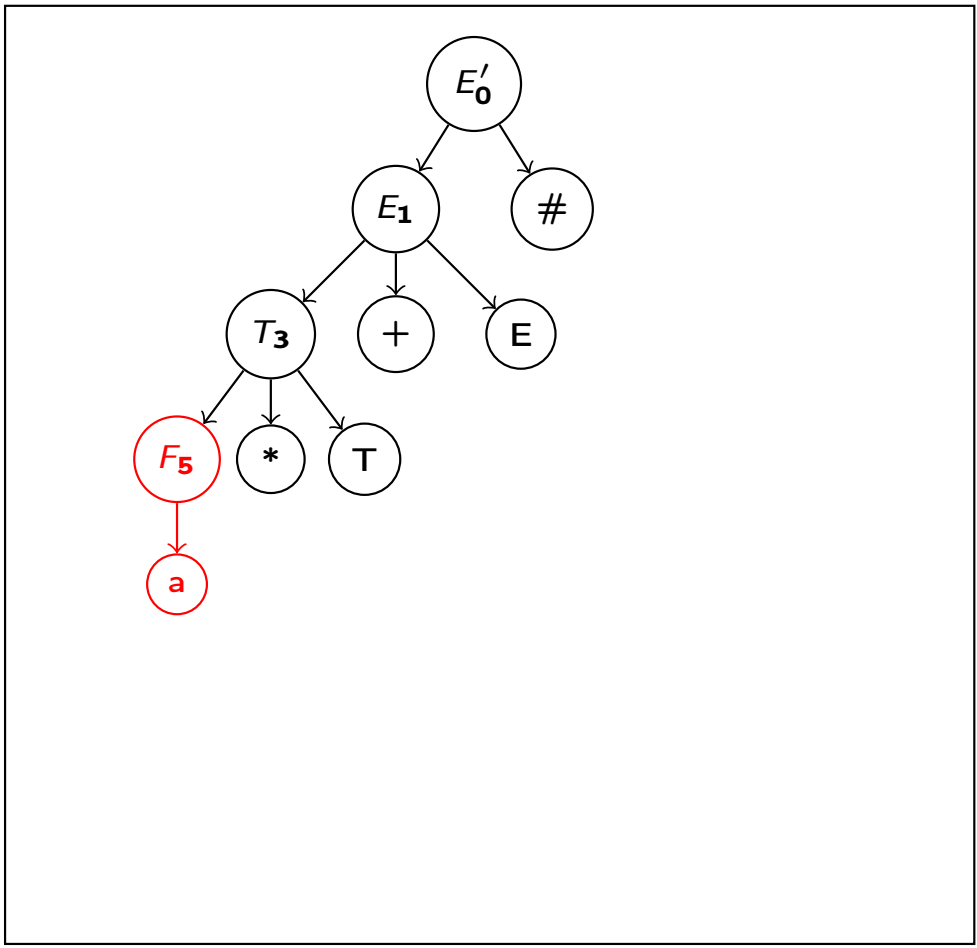
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Devolve o token para a entrada;



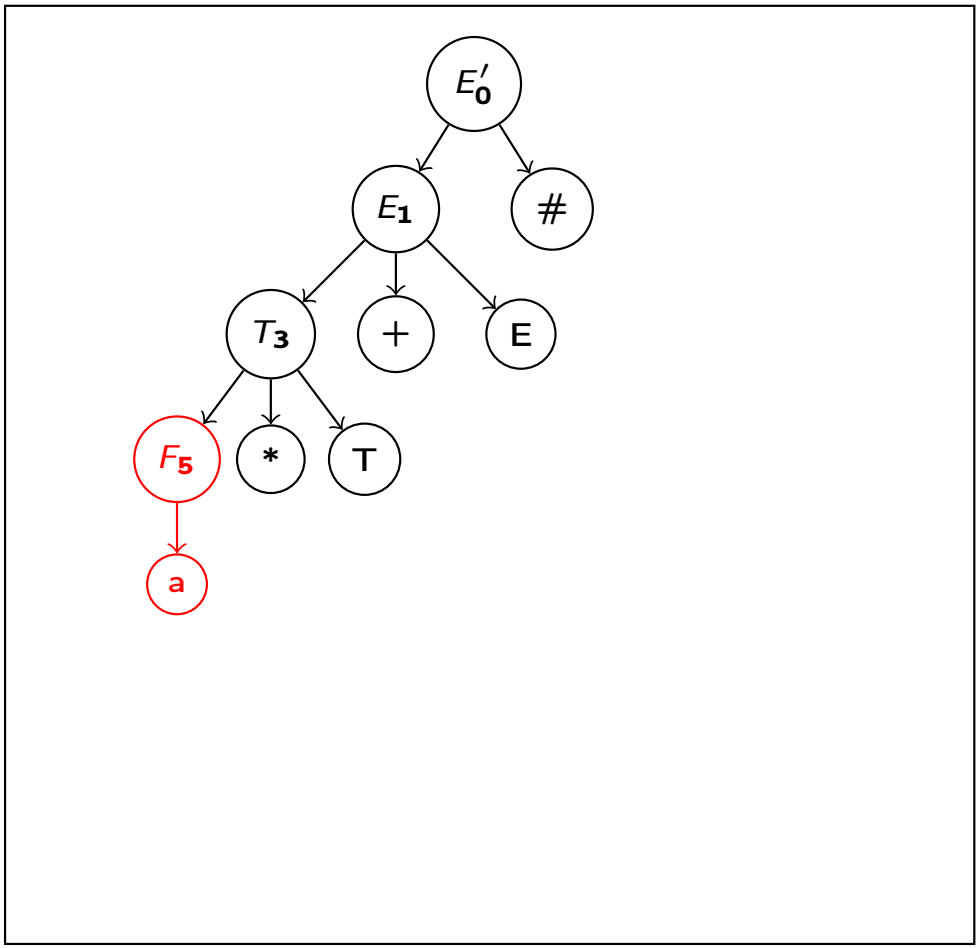
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Devolve o token para a entrada;
- Talvez o erro tenha sido utilizar a regra (5).



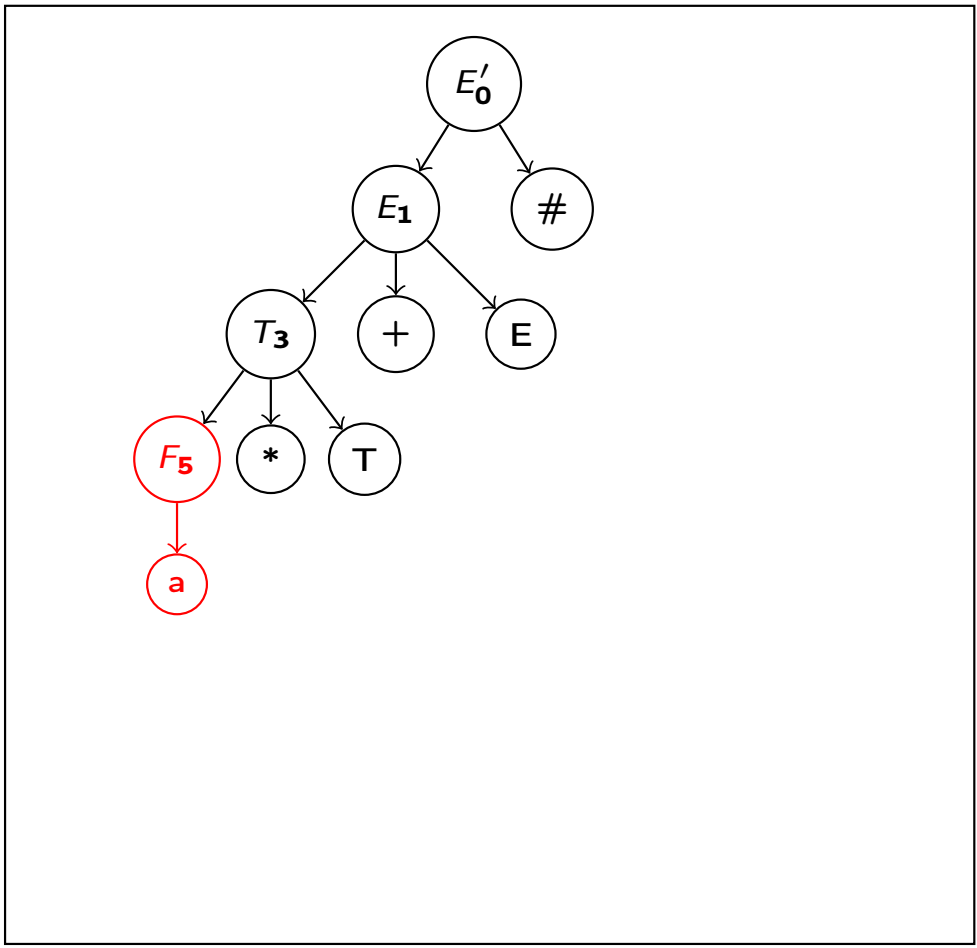
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$	
1	$E \rightarrow T + E$	
2	$E \rightarrow T$	
3	$T \rightarrow F * T$	
4	$T \rightarrow F$	
5	$F \rightarrow a$	
6	$F \rightarrow b$	
7	$F \rightarrow (E)\}$	

$\alpha =$ a + b #

- Devolve o token para a entrada;
- Talvez o erro tenha sido utilizar a regra (5).
- Vamos tentar as demais regras para F: (6) e (7).



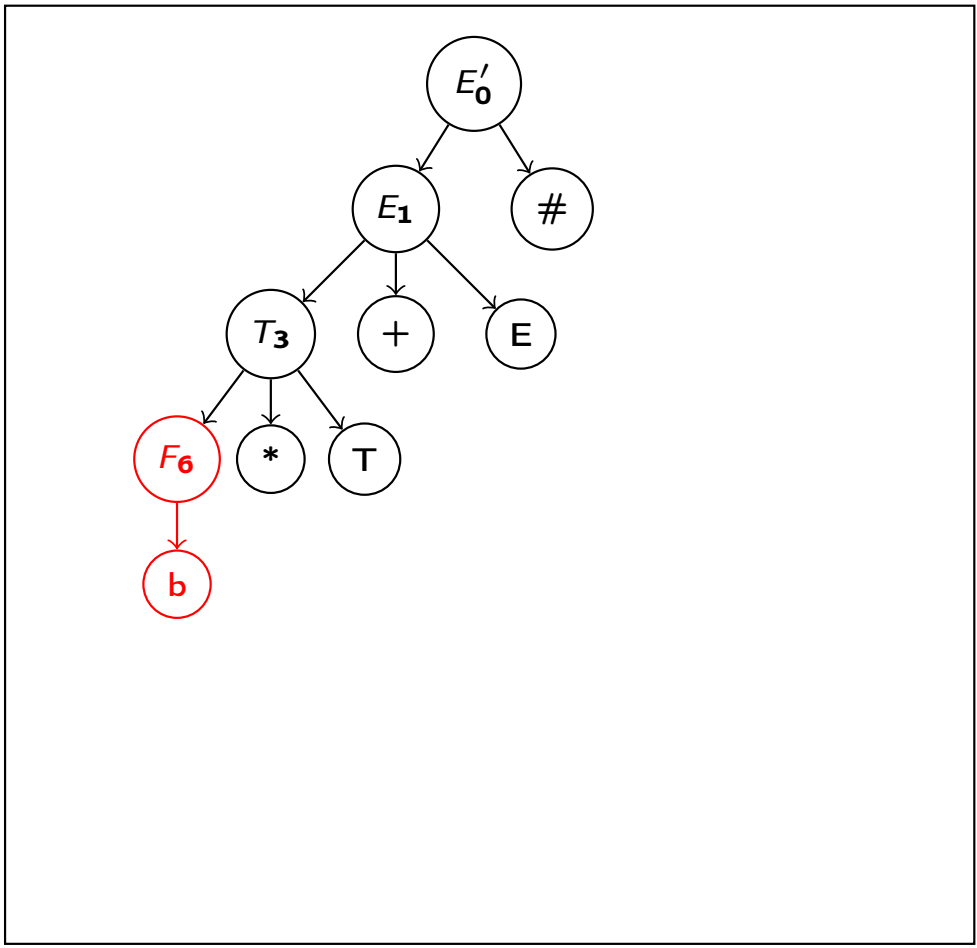
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (6).



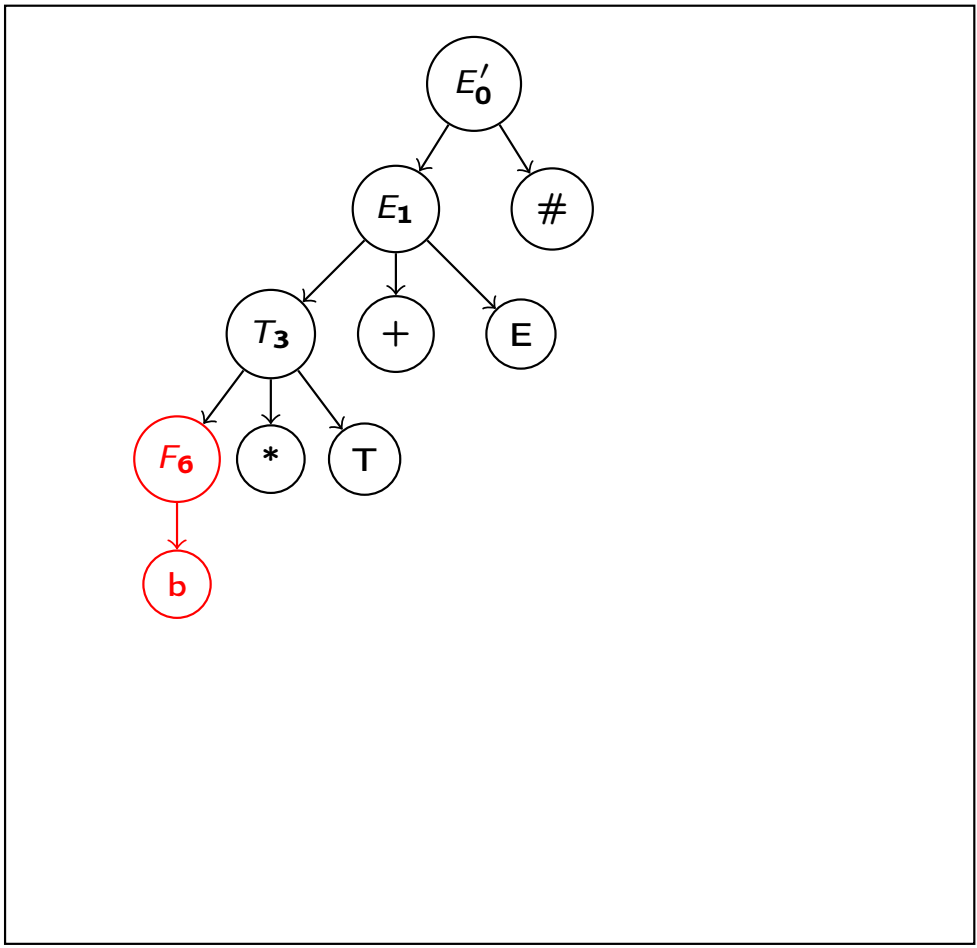
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (6).
- Não bate com o token.



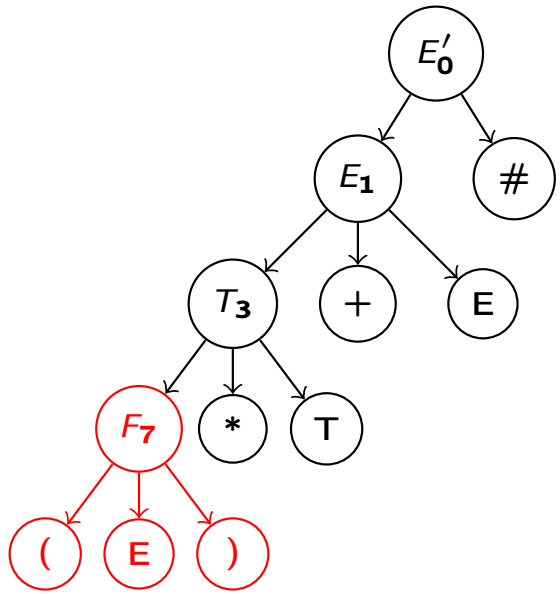
Força Bruta

$G_1 = \{$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (7).



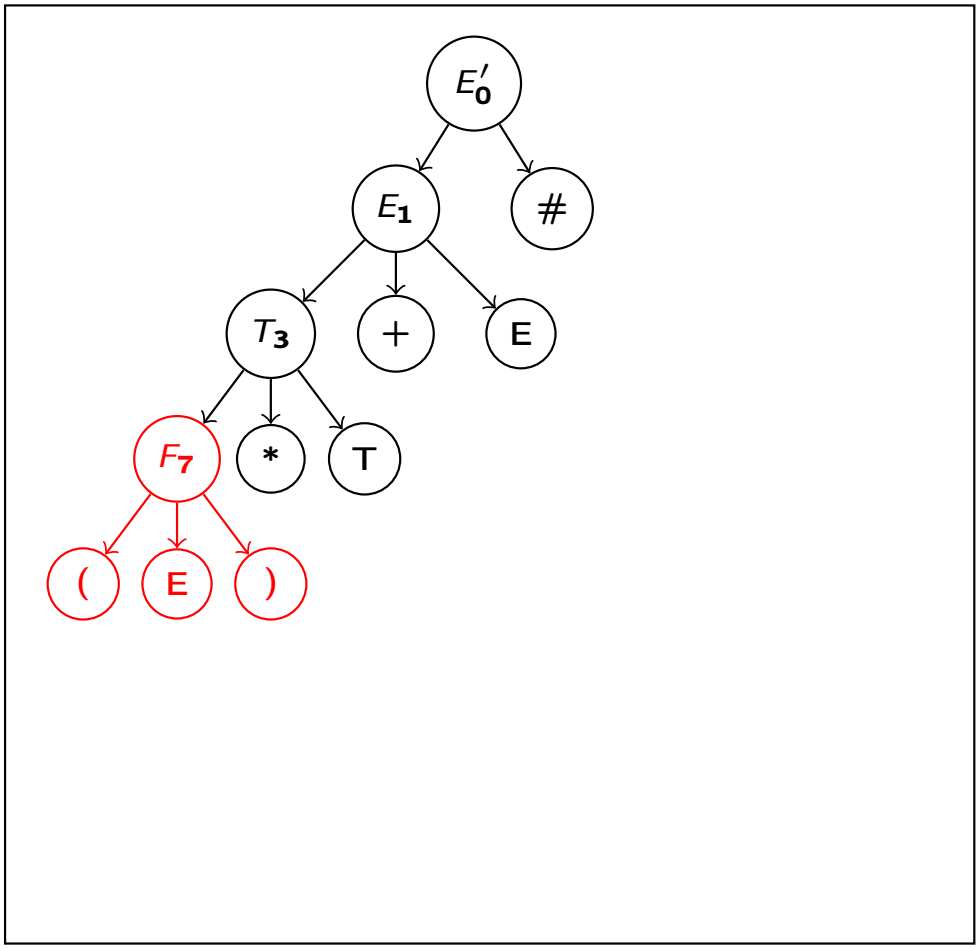
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (7).
- Não bate com o token.



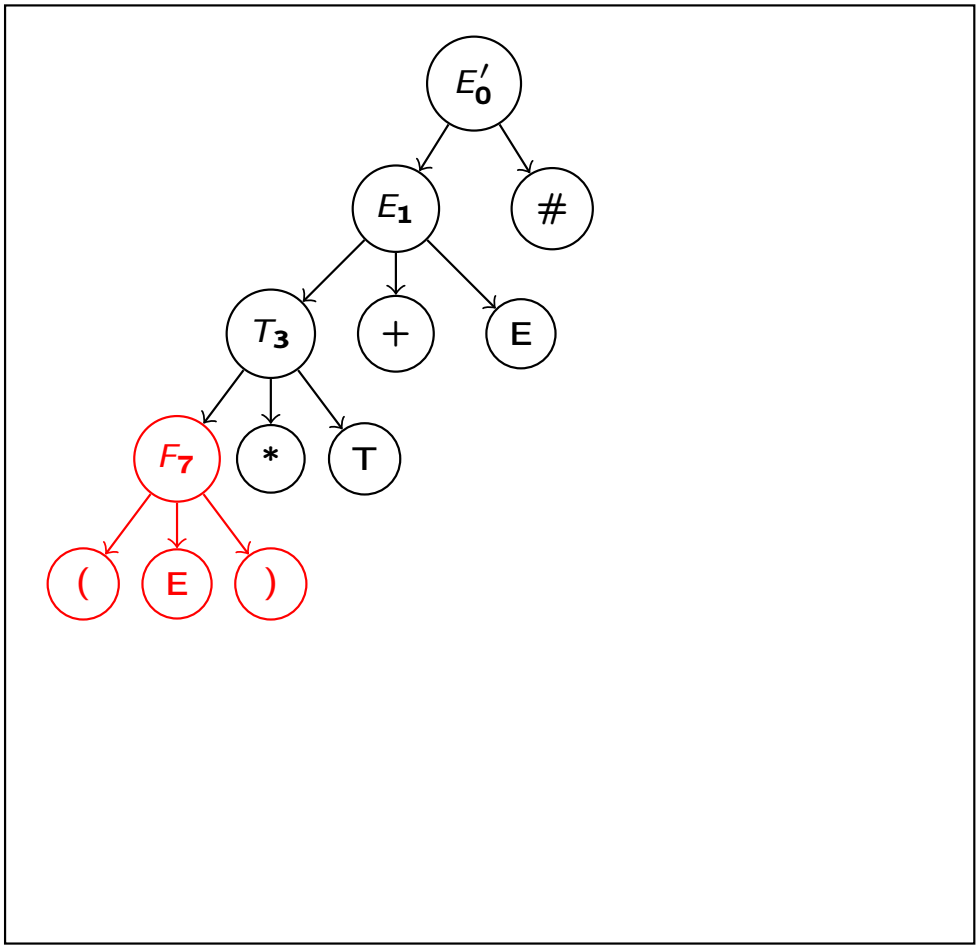
Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (7).
- Não bate com o token.
- Com isto, foram esgotadas todas as possibilidades de derivação de F. O problema não foi aqui.



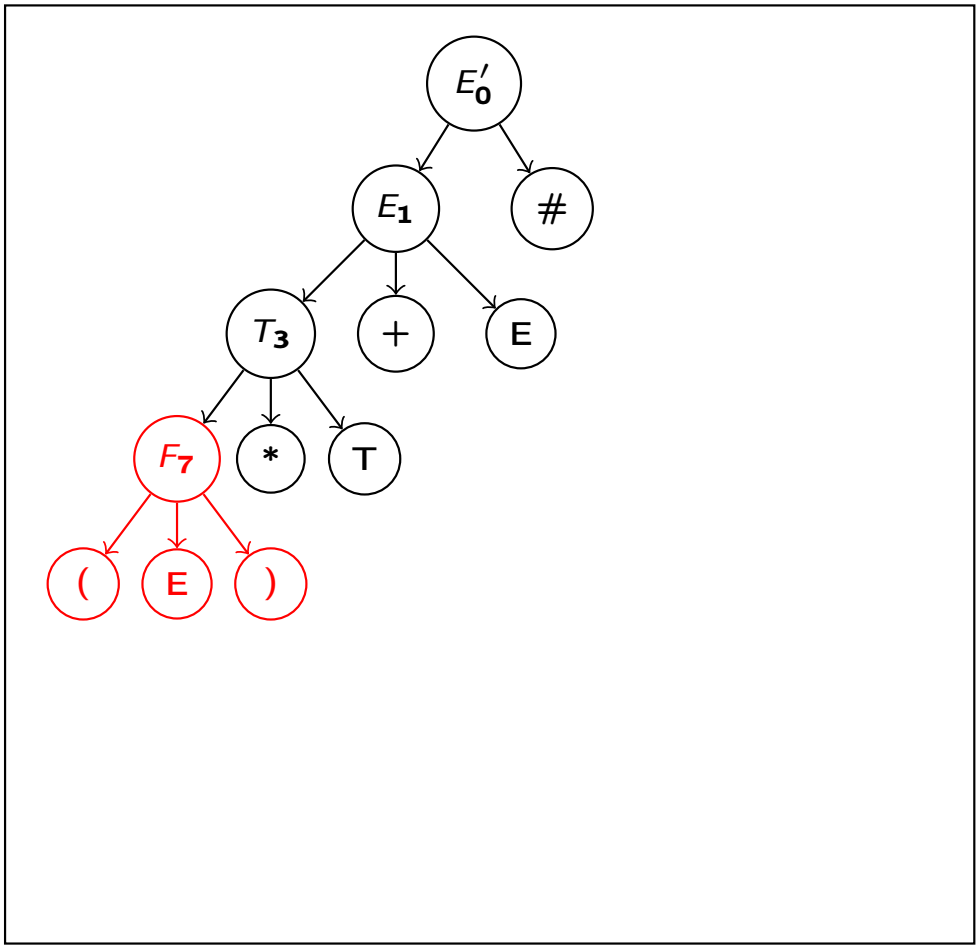
Força Bruta

$G_1 = \{$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Tentando regra (7).
- Não bate com o token.
- Com isto, foram esgotadas todas as possibilidades de derivação de F. O problema não foi aqui.
- Retrocedendo um passo.



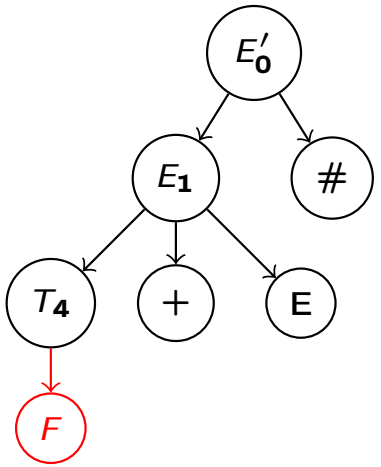
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Talvez o erro tenha sido utilizar a regra (3).



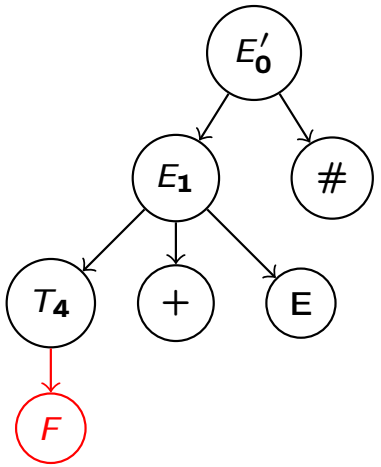
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Talvez o erro tenha sido utilizar a regra (3).
- Vamos tentar a outra regra para T: (4).



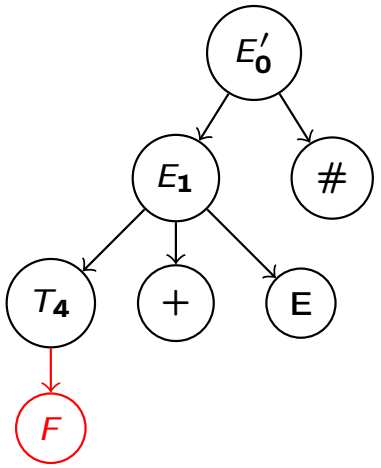
Força Bruta

$$G_1 = \{$$

0	$E' \rightarrow E\#$
1	$E \rightarrow T + E$
2	$E \rightarrow T$
3	$T \rightarrow F * T$
4	$T \rightarrow F$
5	$F \rightarrow a$
6	$F \rightarrow b$
7	$F \rightarrow (E)\}$

$\alpha =$ a + b #

- Talvez o erro tenha sido utilizar a regra (3).
- Vamos tentar a outra regra para T: (4).
- Aplicando seguidamente de algoritmo, eventualmente se chega a árvore de derivação desejada.

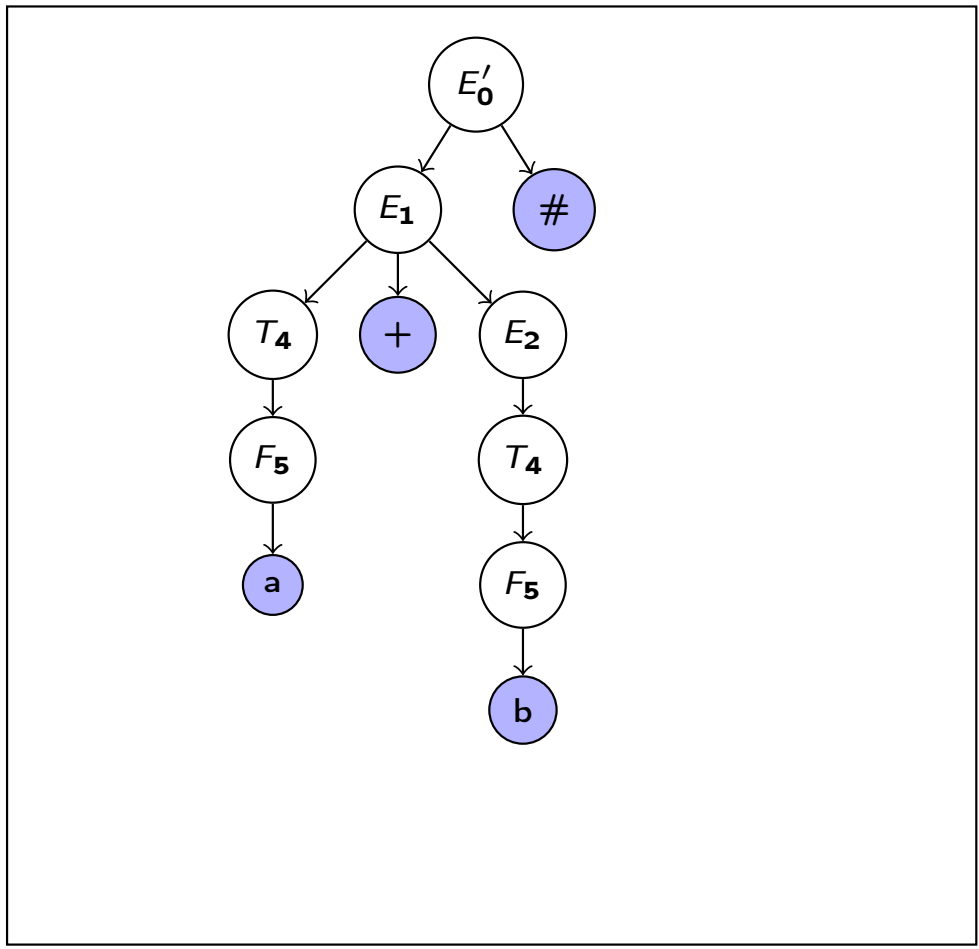


Força Bruta

$$G_1 = \{$$

(0)	$E' \rightarrow E\#$
(1)	$E \rightarrow T + E$
(2)	$E \rightarrow T$
(3)	$T \rightarrow F * T$
(4)	$T \rightarrow F$
(5)	$F \rightarrow a$
(6)	$F \rightarrow b$
(7)	$F \rightarrow (E)\}$

$$\alpha =$$



- Aqui todos os tokens foram pendurados na árvore;
- a árvore está "decorada";
- Este é o termo utilizado na bibliografia.

A.S.Preditivo

- É possível adaptar este algoritmo para ser mais inteligente, por exemplo utilizando derivações somente quando são viáveis.
- Esta classe de algoritmos é chamada analisadores sintáticos preditivos;
- escolhem uma produção baseados somente em duas informações: o token corrente e a árvore já construída.
- a gramática tem que ser modificada para um determinado formato denominado LL(1):
  - *Left to right parsing*;
  - *Leftmost derivation*;
  - Examinam um único token à frente;

A.S.Preditivo

- Modificações na gramática;
- Seja $G_1 \notin LL(1)$ uma gramática. Aplicar as modificações abaixo para obter $G_2 \in LL(1)$;
- As modificações garantem que $L(G_1) = L(G_2)$;
  - Eliminar Retrocessos;
  - Fatoração;
  - Eliminar Recursão Esquerda;
- Escrever analisador sintático preditivo a partir de uma gramática no formato LL(1).

Conceito

- Considere a regra $A \rightarrow b|c$:
- O algoritmo da força bruta iria primeiro tentar a derivação $A \rightarrow b$ e depois $A \rightarrow c$:
- Um melhoramento seria escolher a derivação baseado no token corrente:

```
1  Caso token for
2      "b": A → b
3      "c": A → c
4      outros: Erro
5  FimCaso
```

FNGreibach

- Isto sugere alterar a gramática de entrada para o formato de uma gramática na forma normal de Greibach, onde todas as produções são da forma abaixo:

$$\{A \rightarrow a\}$$
$$\{A \rightarrow X_1\alpha | X_2\alpha | \ldots\}$$

- Infelizmente, a transformação gera um grande número de produções, onde é difícil inserir as ações semânticas.

Descobrindo o primeiro símbolo

- Considere as produções abaixo:

$$\begin{aligned} \{A &\rightarrow a|BA\} \\ \{B &\rightarrow b|CB\} \\ \{C &\rightarrow c\} \end{aligned}$$

- Não é difícil de ver que os primeiros terminais válidos para aplicar uma derivação são:
  - A: a, b, c
  - B: b, c
  - C: c
- Faça a análise sintática da entrada $\alpha =$ "cba".
- Existe um mecanismo formal para calcular o primeiro símbolo.

Algoritmo Primeiro (*First*)

- Definição Informal: Primeiro(A) é o conjunto de todos os terminais que começam qualquer seqüência derivável de A.
- Definição Formal: se $A \stackrel{*}{\Rightarrow} x$ então $x \in$ Primeiro(A)

Algoritmo Primeiro (*First*)

- O primeiro símbolo terminal (ou só "Primeiro") de cada variável pode ser obtido com o seguinte algoritmo:
  1. Desenhe uma tabela com quatro colunas;
  2. Preencha a primeira coluna de cada linha com uma variável;
  3. Preencha a segunda coluna de cada linha com os terminais ou variáveis que podem ser obtidos em uma derivação.
  4. Preencha a terceira coluna com o fecho transitivo da segunda;
  5. Copie os terminais da terceira coluna para a quarta coluna.
- A quarta coluna contém os terminais válidos de cada variável.

Exemplo

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passos 1 e 2: Criar a tabela, preencher primeira coluna

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB		
A	aC		
B	bd		
C	c		

Exemplo

$$
\begin{array}{lcl}
S & \rightarrow & AS|BA \\
A & \rightarrow & aB|C \\
B & \rightarrow & bA|d \\
C & \rightarrow & c
\end{array}
$$

- Passo 3: Fecho transitivo.
- Início: copiar primeira coluna:

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	AB	
A	aC	aC	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 3: Fecho transitivo.
- Linha S: Aplicar fecho na variável A.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaC	
A	aC	aC	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{array}{lcl}
S & \rightarrow & AS|BA \\
A & \rightarrow & aB|C \\
B & \rightarrow & bA|d \\
C & \rightarrow & c
\end{array}
$$

- Passo 3: Fecho transitivo.
- Linha S: Aplicar fecho na variável B.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaCbd	
A	aC	aC	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 3: Fecho transitivo.
- Linha S: Aplicar fecho na variável C.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaCbdc	
A	aC	aC	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 3: Fecho transitivo: Fim Linha S.
- Terceira coluna contém os terminais válidos.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaCdbc	adbc
A	aC	aC	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{array}{lcl}
S & \rightarrow & AS|BA \\
A & \rightarrow & aB|C \\
B & \rightarrow & bA|d \\
C & \rightarrow & c
\end{array}
$$

- Passo 3: Fecho transitivo: Próxima linha.
- Linha A: Aplicar fecho na variável C.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABadbc	adbc
A	aC	aCc	
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 3: Fecho transitivo: Fim Linha A.
- Terceira coluna contém os terminais válidos.

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABadbc	adbc
A	aC	aCc	ac
B	bd	bd	
C	c	c	

Exemplo

$$
\begin{array}{lcl}
S & \rightarrow & AS|BA \\
A & \rightarrow & aB|C \\
B & \rightarrow & bA|d \\
C & \rightarrow & c
\end{array}
$$

- Passo 3: Fecho transitivo
- Linhas B e C só tem terminais: copia

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABadbc	adbc
A	aC	aCc	ac
B	bd	bd	bd
C	c	c	c

Resultado

$$
\begin{array}{lcl}
S & \rightarrow & AS|BA \\
A & \rightarrow & aB|C \\
B & \rightarrow & bA|d \\
C & \rightarrow & c
\end{array}
$$

- Primeiro (S) = adbc
- Primeiro (A) = ac
- Primeiro (B) = bd
- Primeiro (C) = c
- Agora já é possível fazer a análise sintática com complexidade $O(n)$. Verifique com a entrada $\alpha =$ "*abcdad*"

Fatoração

- Considere a regra $S \rightarrow aAS|aSA$;
- Como o analisador sintático só tem acesso a um Token, se este token for "a", ele teria de tentar as duas derivações.
- Porém, a gramática pode ser escrita fatorando o primeiro termo comum, resultando em regras com duas notações possíveis.

$$S \rightarrow a(AS|SA) \qquad \begin{aligned} S &\rightarrow aX \\ X &\rightarrow AS|SA \end{aligned}$$

- É importante destacar que os símbolos "("e ")"não podem fazer parte do alfabeto da gramática.

Exemplo

- Fatore a gramática G_1 abaixo.

$$\begin{aligned} G_1 = \{E &\rightarrow T + E | T \\ T &\rightarrow F * T | F \\ F &\rightarrow a | (E)\} \end{aligned}$$

-

$$\begin{aligned} G_2 = \{E &\rightarrow T[+E|\epsilon] \\ T &\rightarrow F[*T|\epsilon] \\ F &\rightarrow a|(E)\} \end{aligned}$$

$$\begin{aligned} G_3 = \{E &\rightarrow TE_1 \\ E_1 &\rightarrow +E|\epsilon \\ T &\rightarrow FT_1 \\ T_1 &\rightarrow *T|\epsilon \\ F &\rightarrow a|(E)\} \end{aligned}$$

Observações

- O símbolo ϵ (epsilon) indica a cadeia vazia.
  Não é um símbolo de entrada, e deve ser entendido como "qualquer outra coisa" e não consome tokens.
- Tanto a notação de G_2 e de G_3 são válidas, porém iremos adotar G_2 por ser mais sintética.
- Observe que em G_2 foram inserido os símbolos "[" e "]", que delimitam a parte fatorada. Não precisa ser este, mas não pode ser um token válido da linguagem (como "(" e ")" de G_1).