

Compiladores - Análise SLR

Fabio Mascarenhas – 2015.2

<http://www.dcc.ufrj.br/~fabiom/comp>

Análise SLR

- A ideia da análise SLR é usar o conjunto FOLLOW do não-terminal associado a um item de redução para resolver conflitos
- A intuição é que só faz sentido reduzir se o próximo token (o lookahead) estiver nesse FOLLOW, ou a redução estará errada
- Para ver que isso é verdade, basta lembrar da definição de FOLLOW:

$\text{FOLLOW}(A) = \{ x \text{ é terminal ou EOF} \mid S \xrightarrow{*} wAxv \text{ para algum } w \text{ e } v \}$

- Se a redução for válida então o próximo token tem que estar em FOLLOW(A)!

$A \rightarrow abc$


$uvxv \rightarrow wabc|kv$

Implicações da análise SLR

- Um estado do autômato pode ter vários itens de redução contanto que sejam de não-terminais diferentes, e seus conjuntos FOLLOW sejam disjuntos
- Um estado pode ter itens de *shift* (com um terminal seguindo a marca) misturados a itens de redução contanto que o terminal não pertença ao FOLLOW de nenhum dos não-terminais dos itens de redução
- Toda gramática sem conflitos LR(0) é uma gramática sem conflitos SLR
- Ainda há margem para muitos conflitos shift-reduce e reduce-reduce! A análise SLR já é bem melhor que a LR(0), mas ainda é fraca

Gramática de Expressões

- A gramática de expressões que vimos na aula passada é SLR:

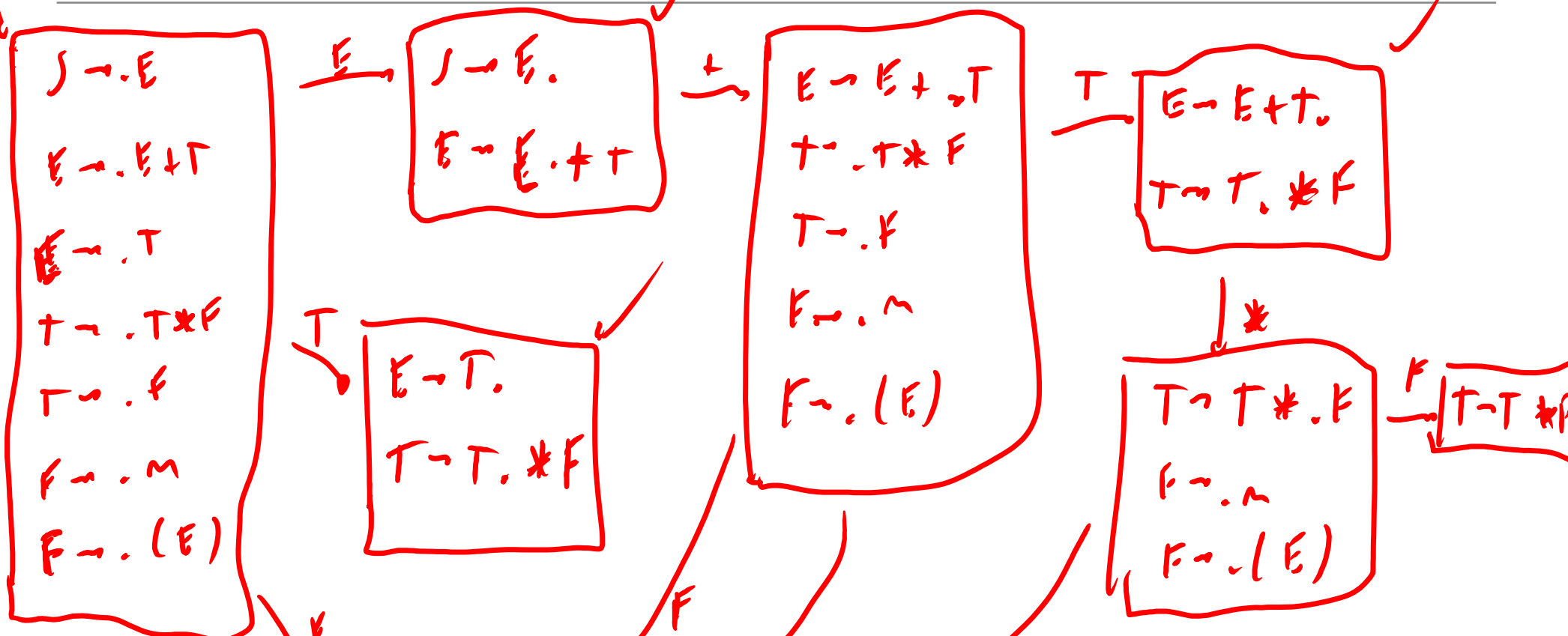


S \rightarrow E
E \rightarrow E + T
E \rightarrow T
T \rightarrow T * F
T \rightarrow F
F \rightarrow num
F \rightarrow (E)

- Podemos construir o autômato dela e verificar

$K_{\text{mov}}(+)=\{e_{q_1, t_1}, *\}$, $K_{\text{mov}}(*)=\{e_{q_1, t_1}, *\}$, $K_{\text{mov}}(s)=\{e_{q_1, t_1}\}$, $K_{\text{mov}}(E)=\{e_{q_1, t_1}\}$

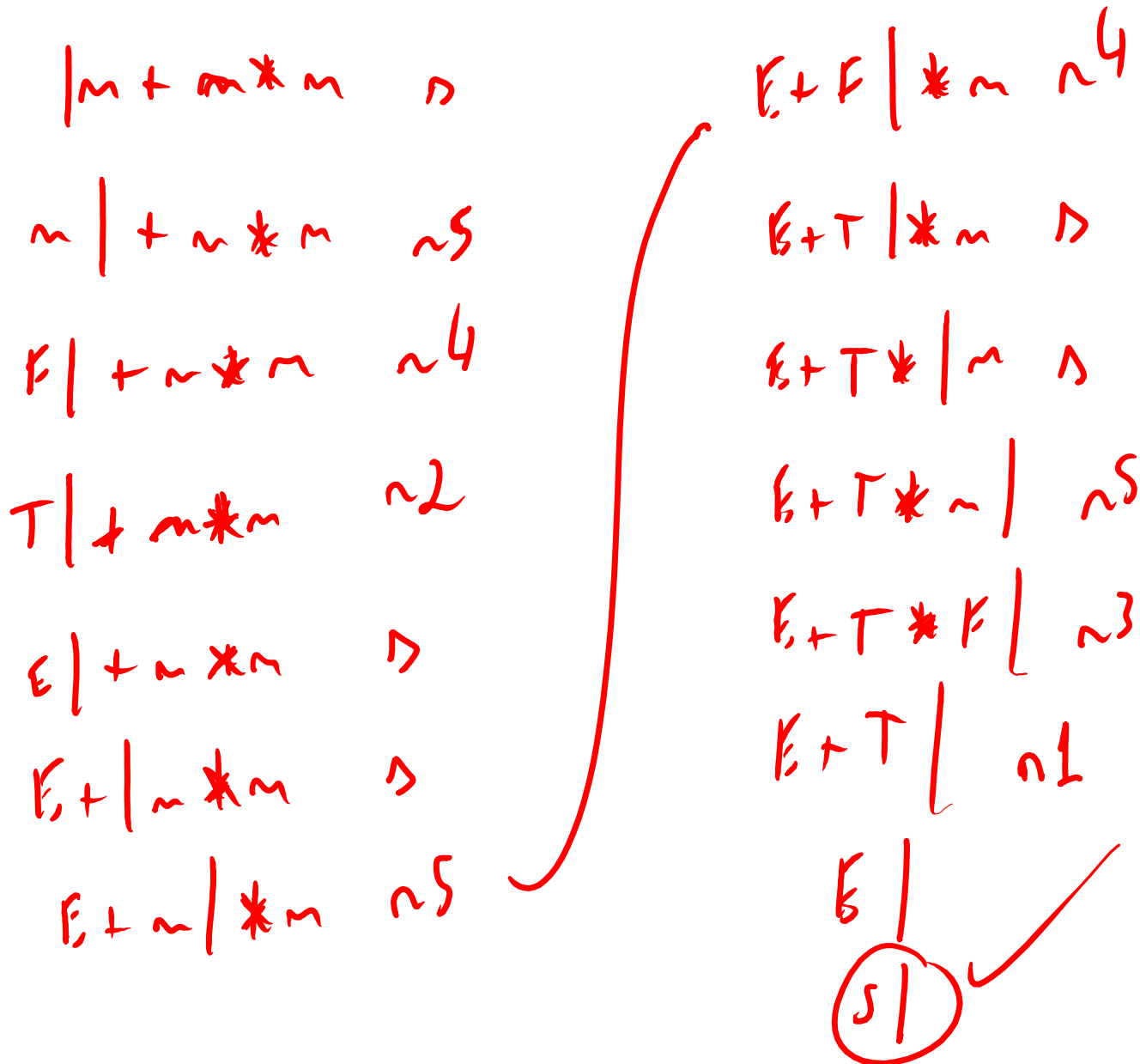
Autômato da gramática de expressões



- S -> E
- E -> E + T
- E -> T
- T -> T * F
- T -> F
- F -> num
- F -> (E)

Analisando uma entrada

$m + m * m$



- 0 S -> E
- 1 E -> E + T
- 2 E -> T
- 3 T -> T * F
- 4 T -> F
- 5 F -> num
- 6 F -> (E)

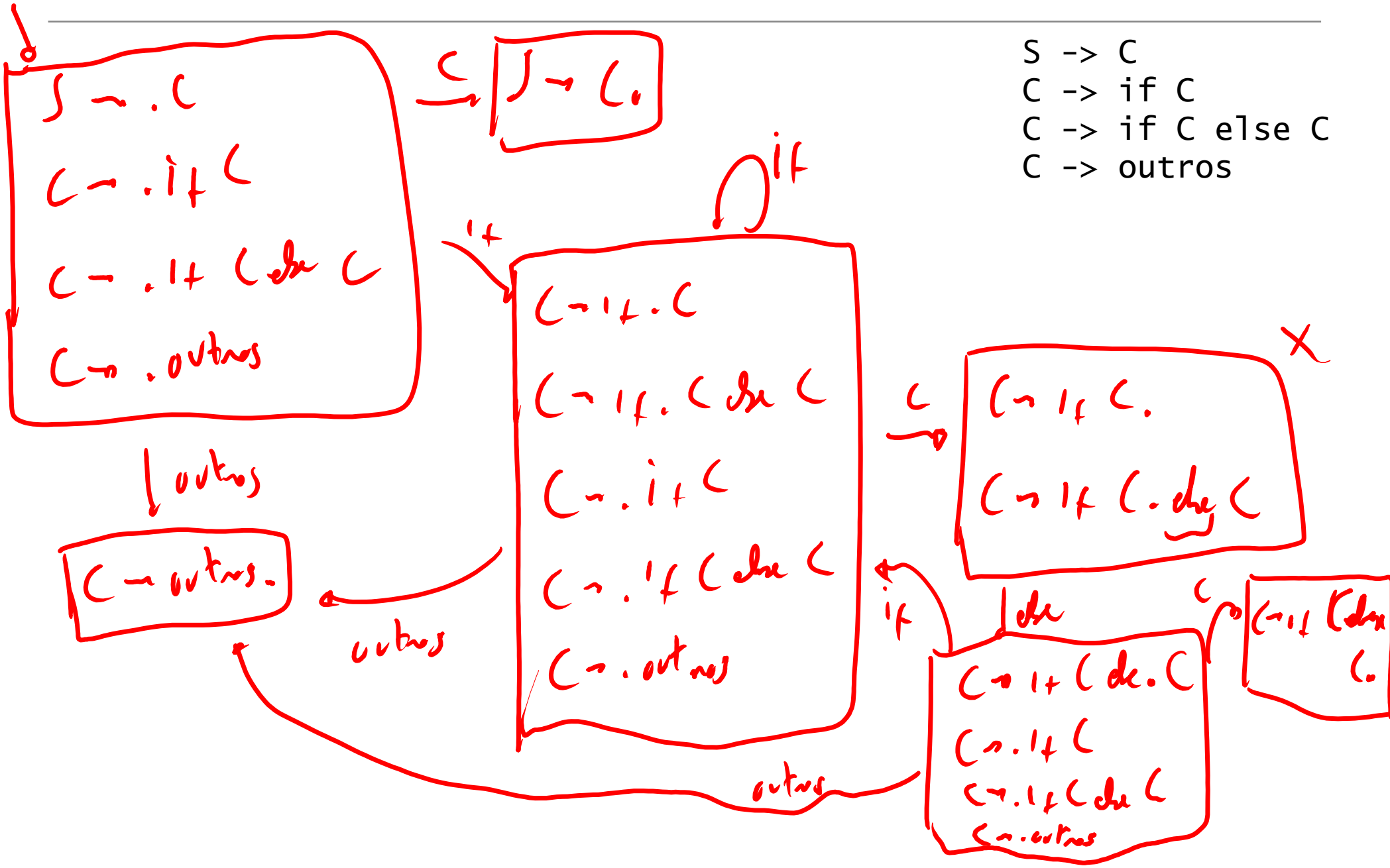
Resolvendo ambiguidade

- Uma gramática ambígua nunca é SLR
- Vamos ver o que acontece com a ambiguidade do if-else:

```
S -> C  
C -> if C  
C -> if C else C  
C -> outros
```

$Follow(C) = \{ \epsilon, \text{else} \}$

Autômato da gramática do if-else



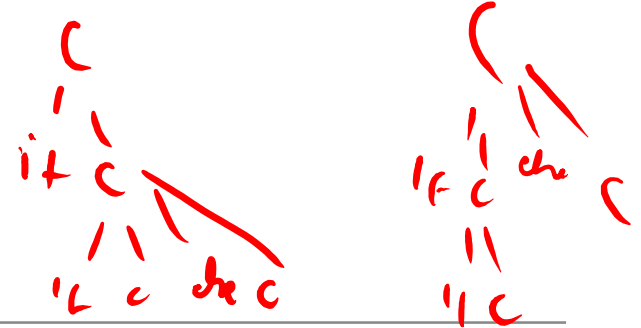
Resolução de conflitos

- A gramática do if-else tem um conflito shift-reduce
- Um analisador SLR tipicamente resolve esse conflito sempre escolhendo shift
- Vamos ver o que isso implica com um exemplo

if if outros else outros



Analisando uma entrada ambígua



| if if outros else outros Δ

- 0 S \rightarrow C
- 1 C \rightarrow if C
- 2 C \rightarrow if C else C
- 3 C \rightarrow outros

if if outros else outros Δ

if if outros else outros Δ

if if outros else outros Δ

if C else outros Δ

if C else outros Δ

if C else outros Δ

if C else C Δ

C Δ

if if C else outros Δ (resolvendo conflito)

if if C else outros Δ

if if C else outros Δ

if if C else C Δ

if C Δ

C Δ

5

Gramáticas de expressões ambíguas

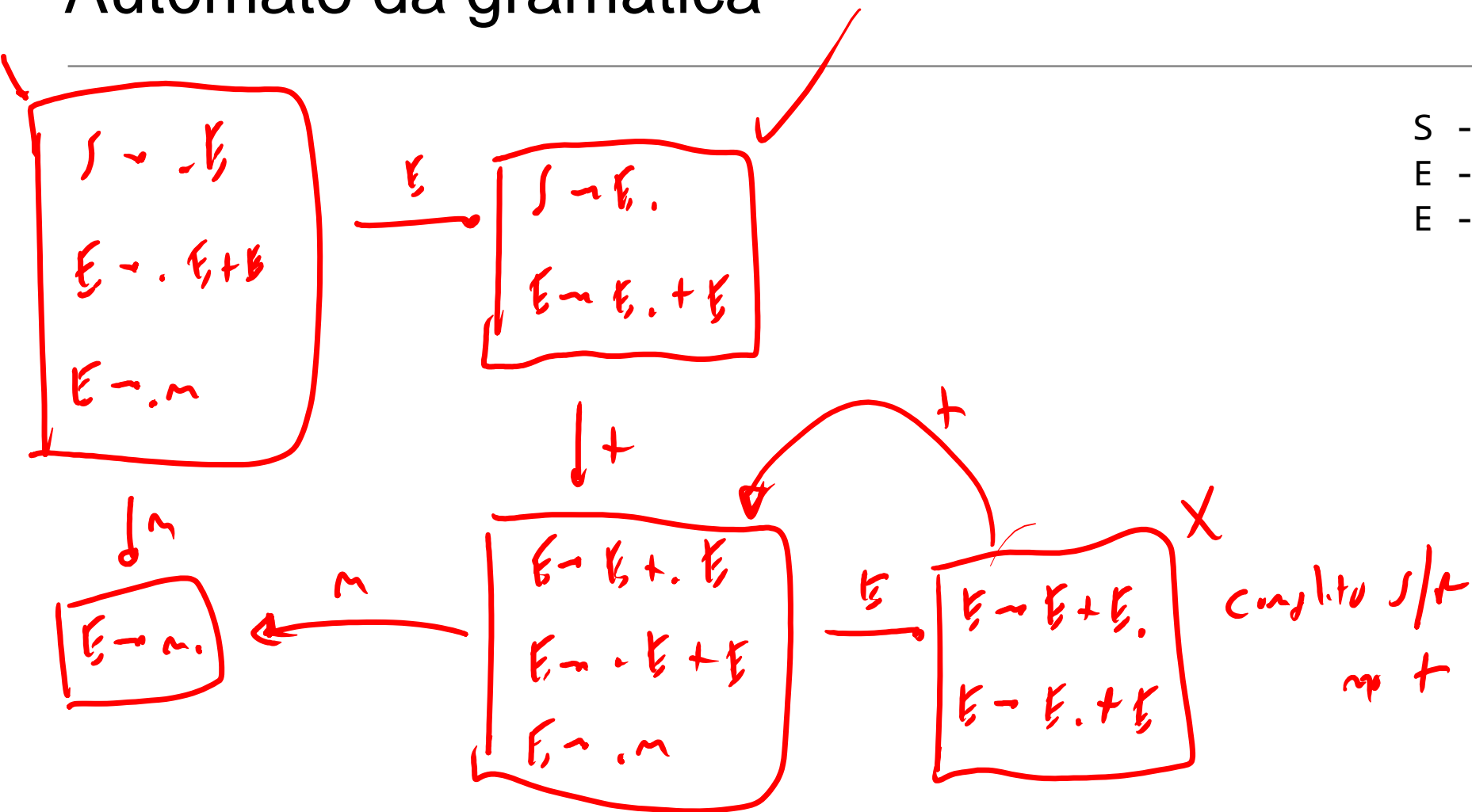
- Vamos agora examinar a gramática ambígua abaixo:

S \rightarrow E
E \rightarrow E + E
E \rightarrow num

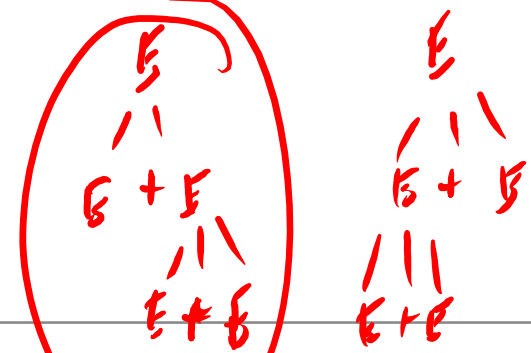
$Follow(S) = \{ \$ \}$

Autômato da gramática

$S \rightarrow E$
 $E \rightarrow E + E$
 $E \rightarrow num$

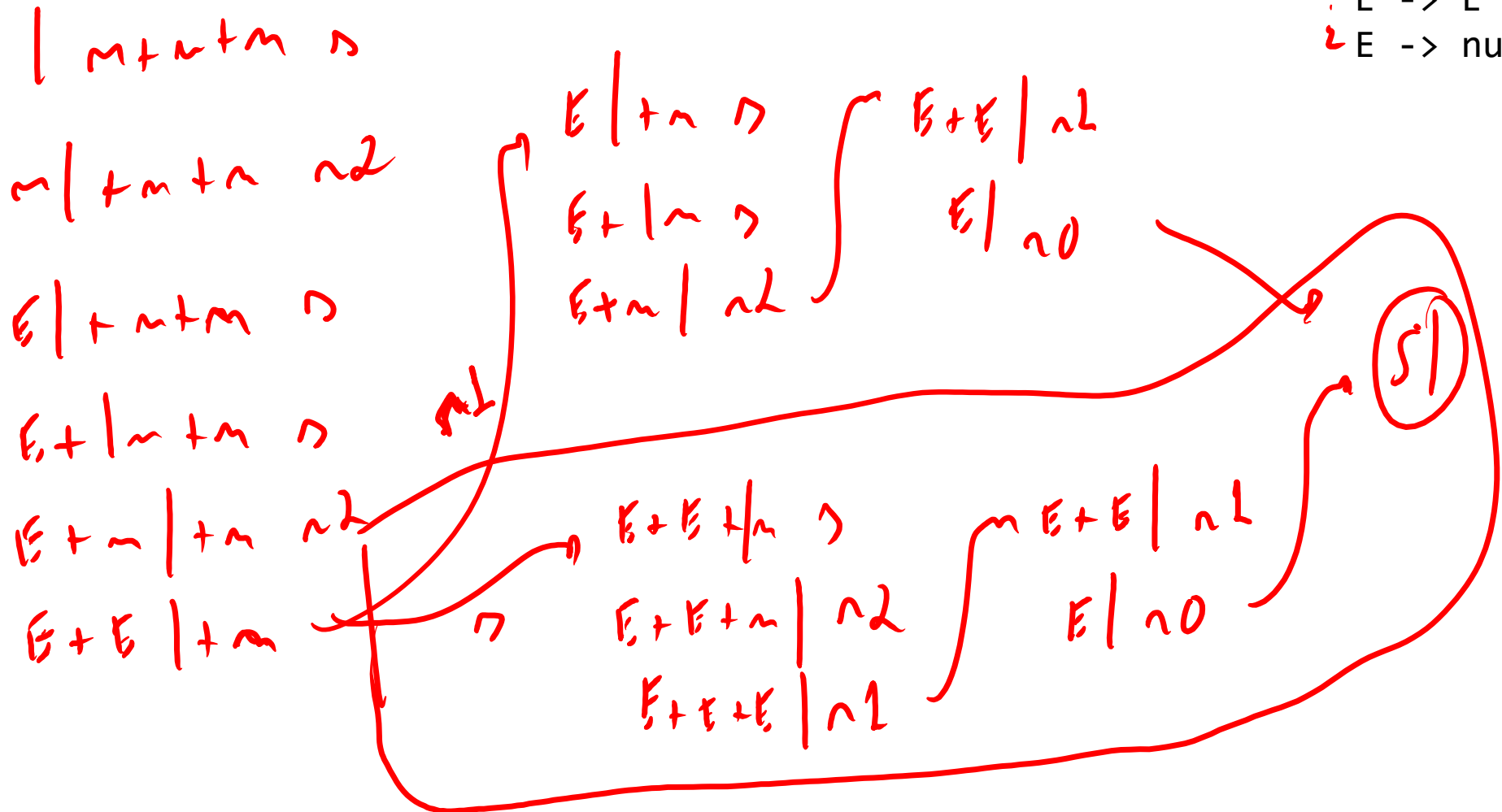


Analizando uma entrada ambígua

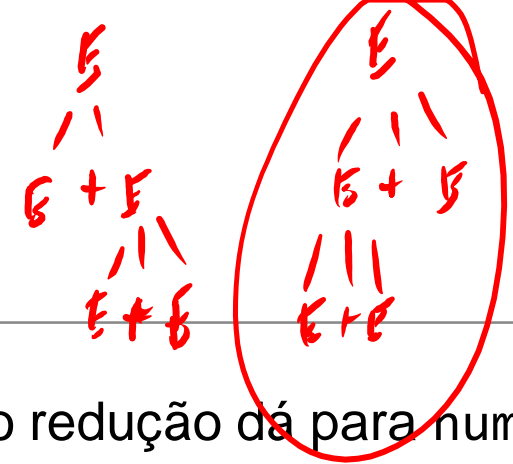


- Ela também tem um conflito shift-reduce, vamos ver o que a solução de conflitos normal dá para $num + num + num$

- 0 S \rightarrow E
- ! E \rightarrow E + E
- 2 E \rightarrow num



Analisando uma entrada ambígua



- Agora vamos ver o que resolvendo o conflito escolhendo redução dá para num + num + num

- 0 S -> E
- ! E -> E + E
- 2 E -> num

