Compiladores - Análise SLR

Fabio Mascarenhas – 2015.1

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:

  FOLLOW(A) = { x é terminal ou EOF | S -*-> wAxv para algum w e v }

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

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:

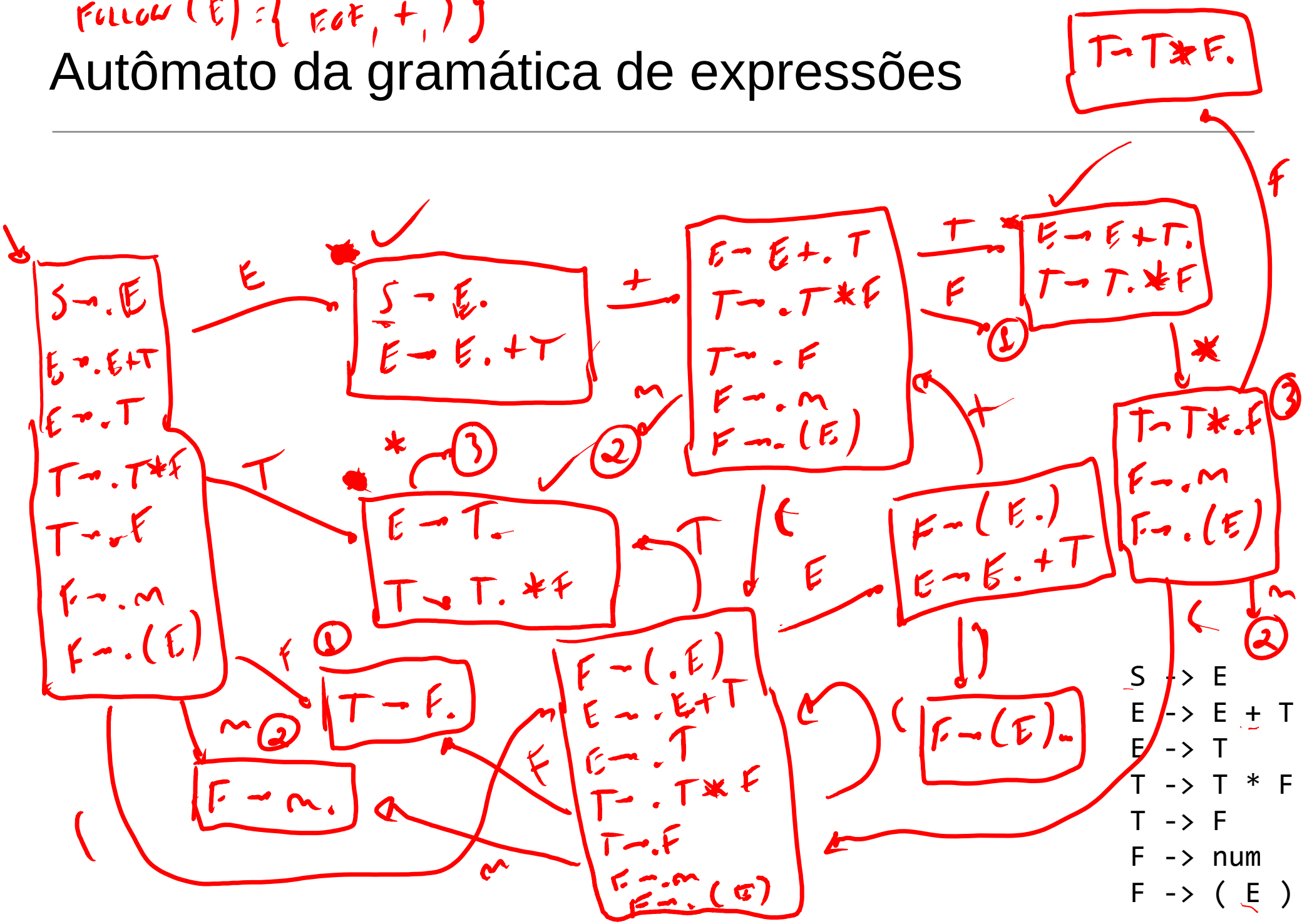
```
S -> E
E -> E + T
E -> T
T -> T * F
T -> F
F -> num
F -> ( E )
```

- Podemos construir o autômato dela e verificar

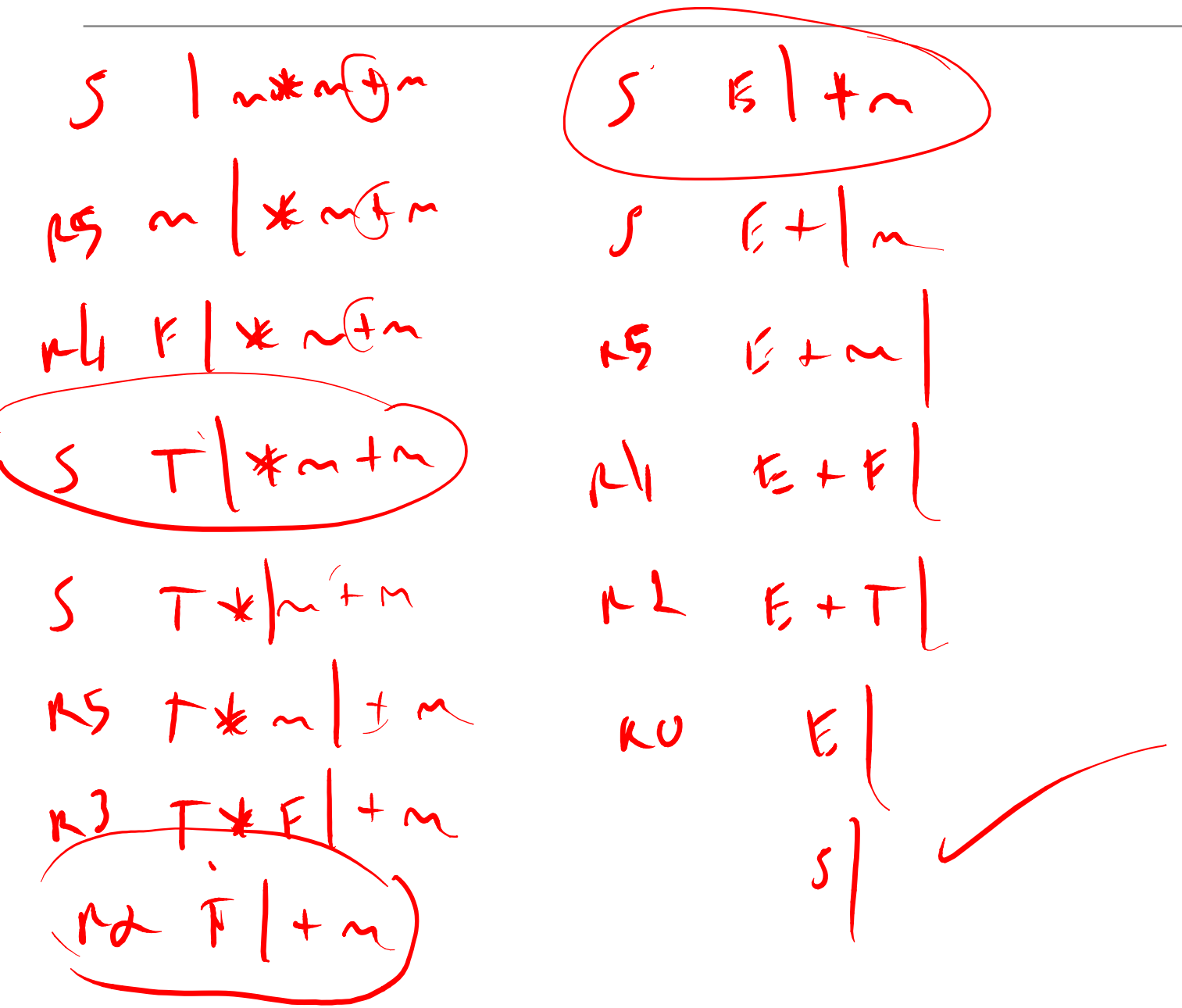
FOLLOW(S) = { EOF }

FOLLOW(E) = { EOF, +,) }

Autômato da gramática de expressões



Analisando uma entrada



```
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

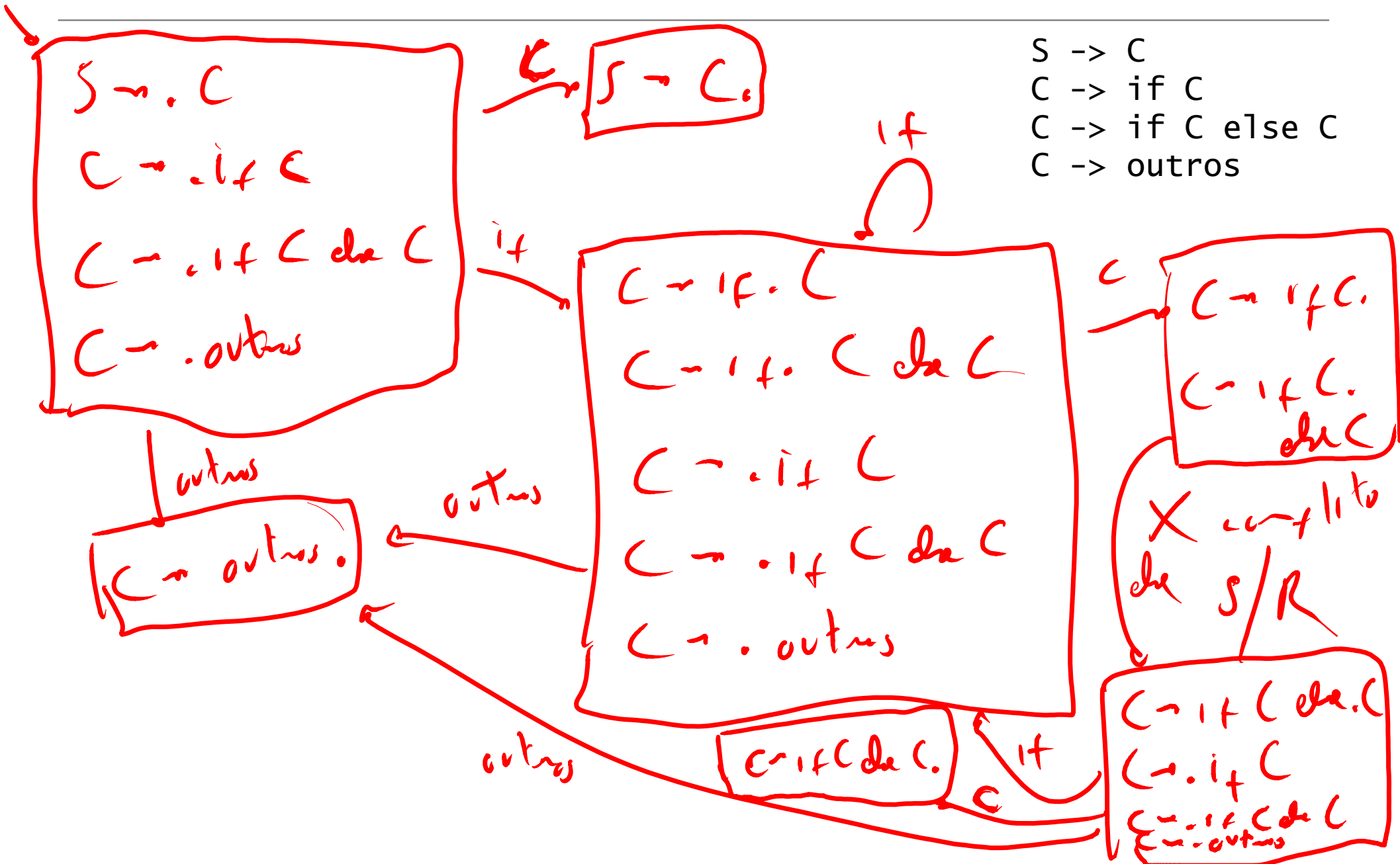
- Uma gramática ambígua nunca é SLR

- Vamos ver o que acontece com a ambiguidade do if-else:

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

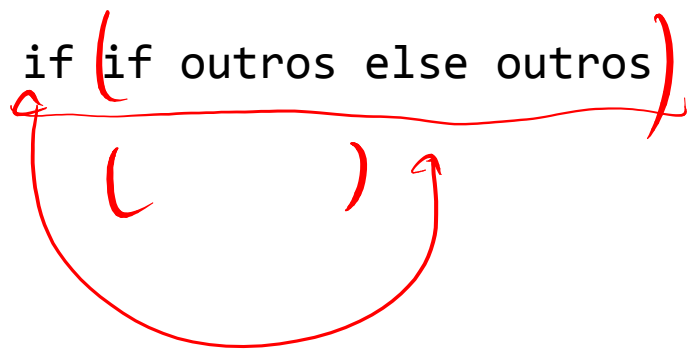
Autômato da gramática do if-else

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



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



S → C → if C → if if C else C → if if C else outros →
if if outros else outros

Analisando uma entrada ambígua

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

S | if if outros else outros

S if | if outros else outros

S if if | outros else outros

R3 if if outros | else outros

S if if C | else outros

S if if C else | outros

R3 if if C else outros |

R2 if if C else C |

R1 if C |

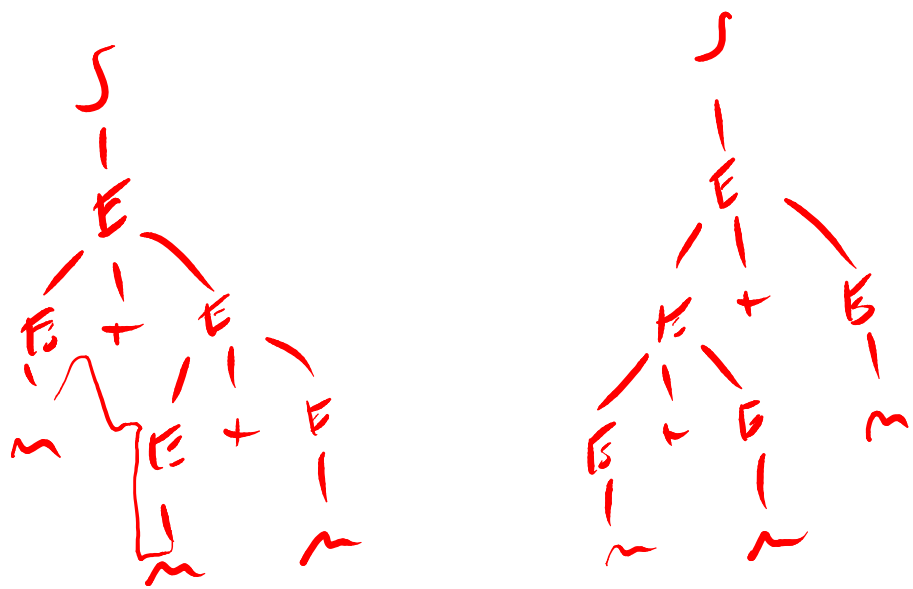
R0 C |

S | ✓

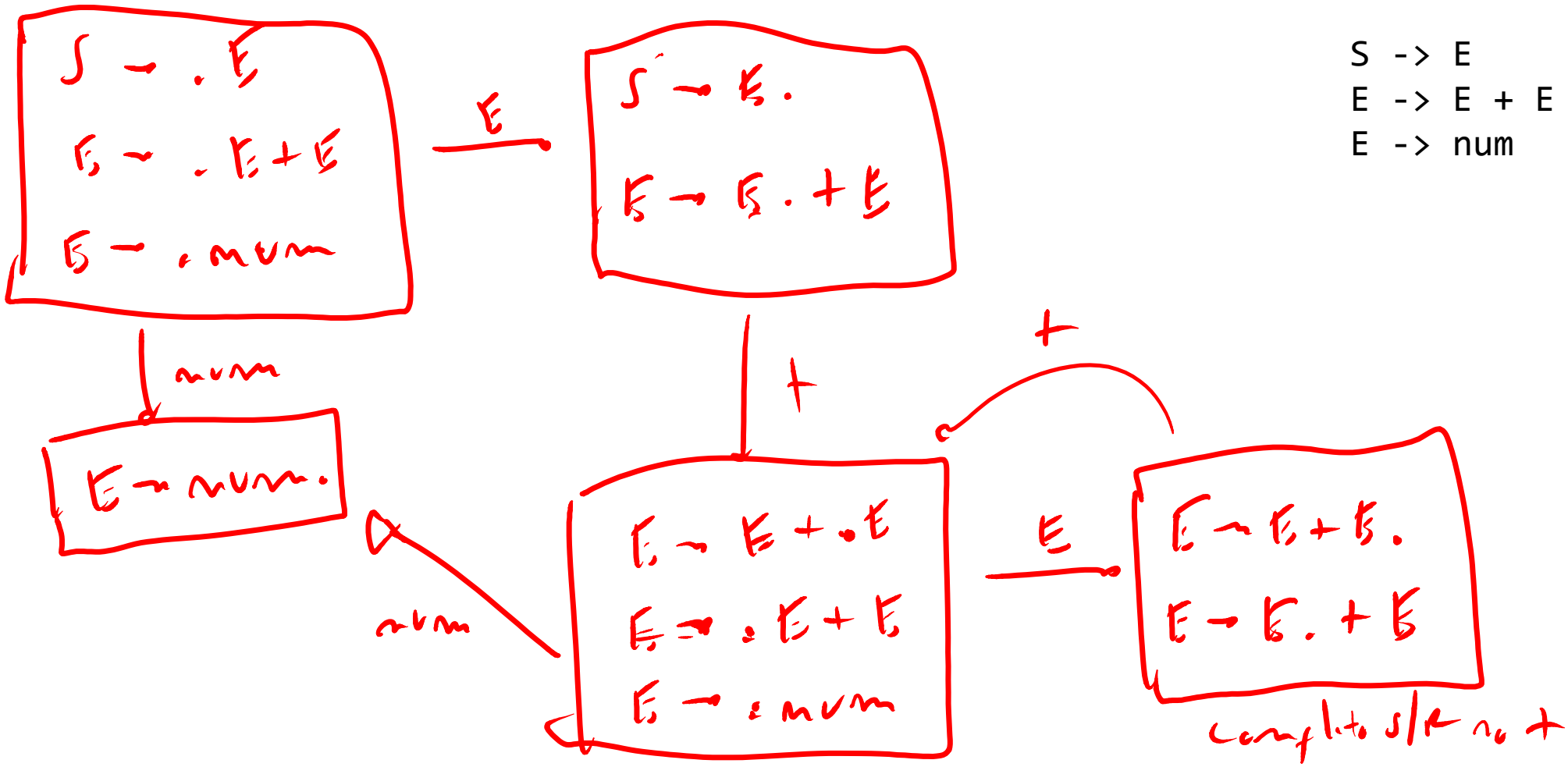
Gramáticas de expressões ambíguas

- Vamos agora examinar a gramática ambígua abaixo:

```
S -> E
E -> E + E
E -> num
```



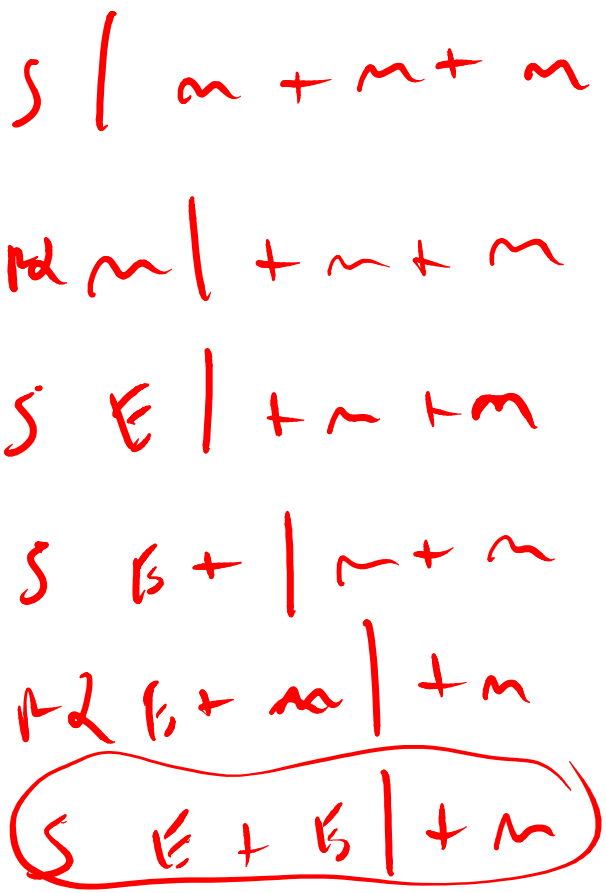
Autômato da gramática

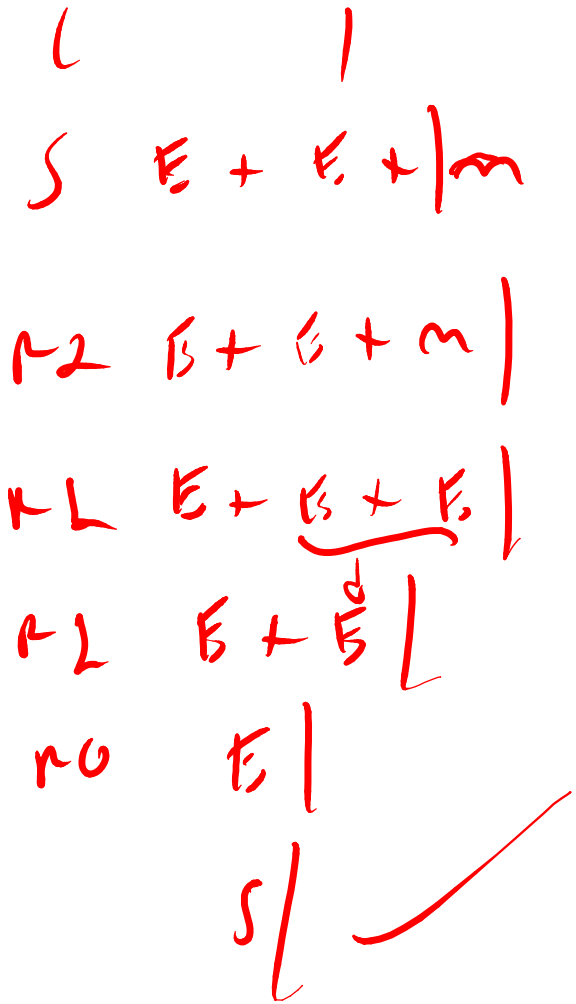


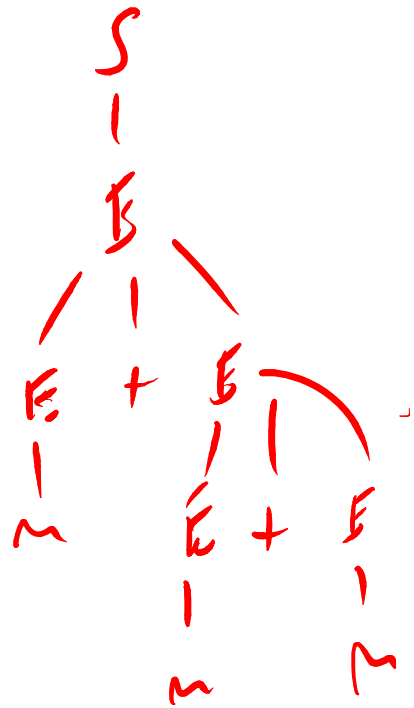
Analisando uma entrada ambígua

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

```
S -> E
E -> E + E
E -> num
```



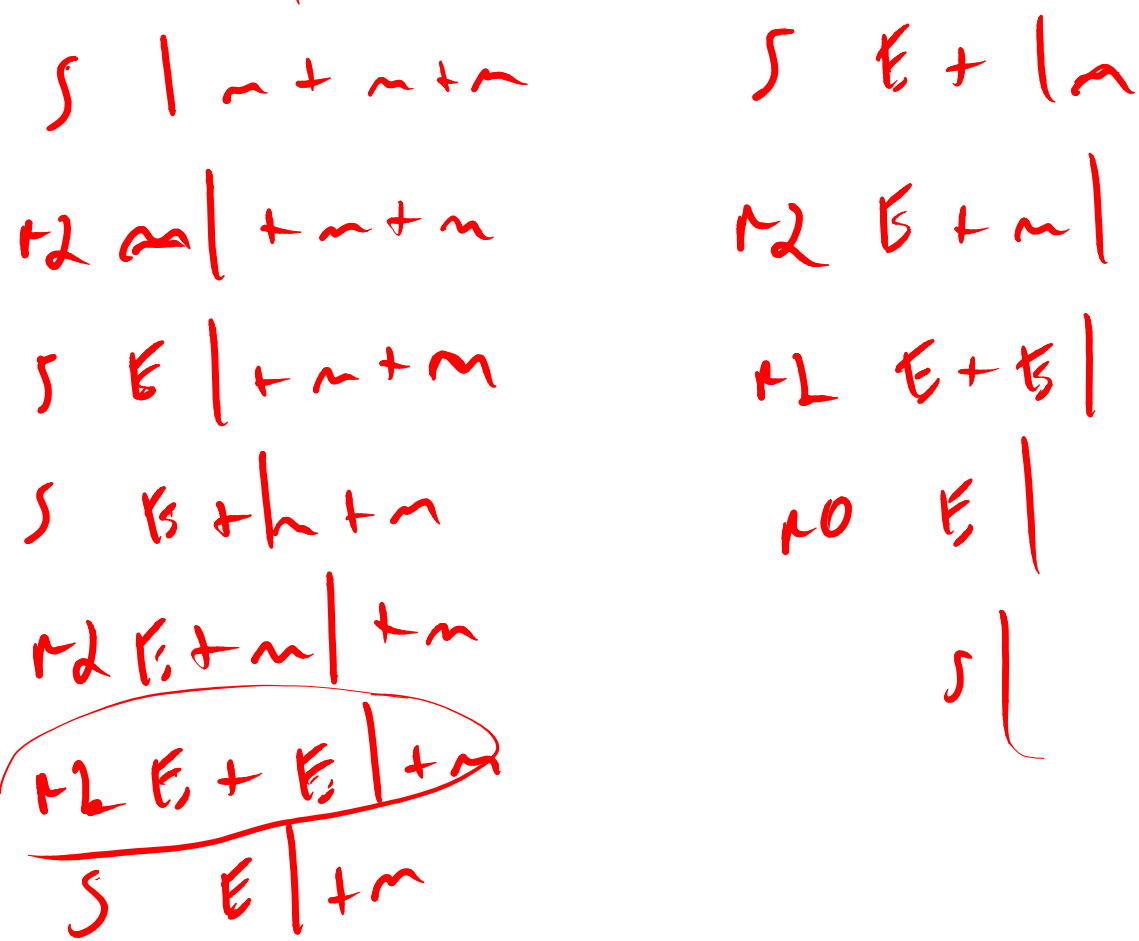


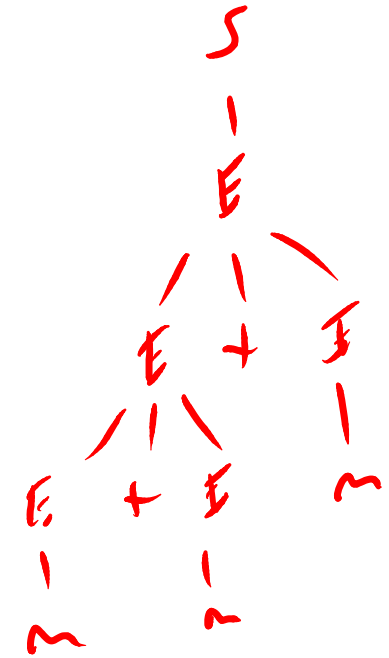


Analisando uma entrada ambígua

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

```
S -> E
E -> E + E
E -> num
```

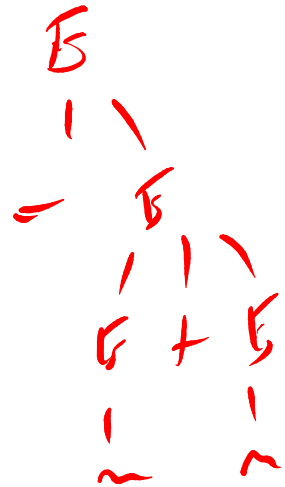




Precedência de operadores

- Vamos agora ver uma gramática mais complexa:

```
S -> E
E -> E + E
E -> E * E
E -> - E
E -> num
```

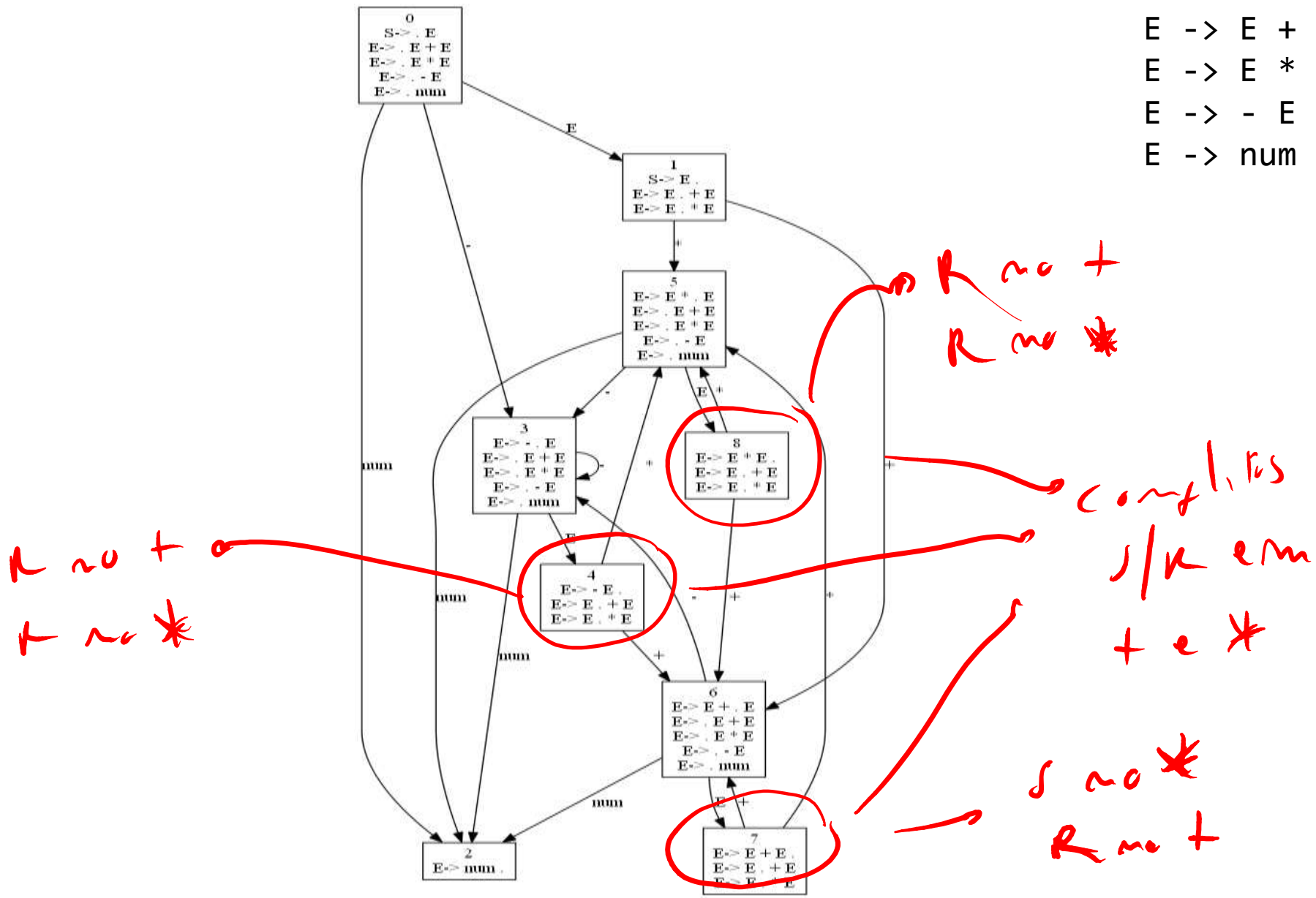


- Qual será o comportamento dessa gramática nas entradas:

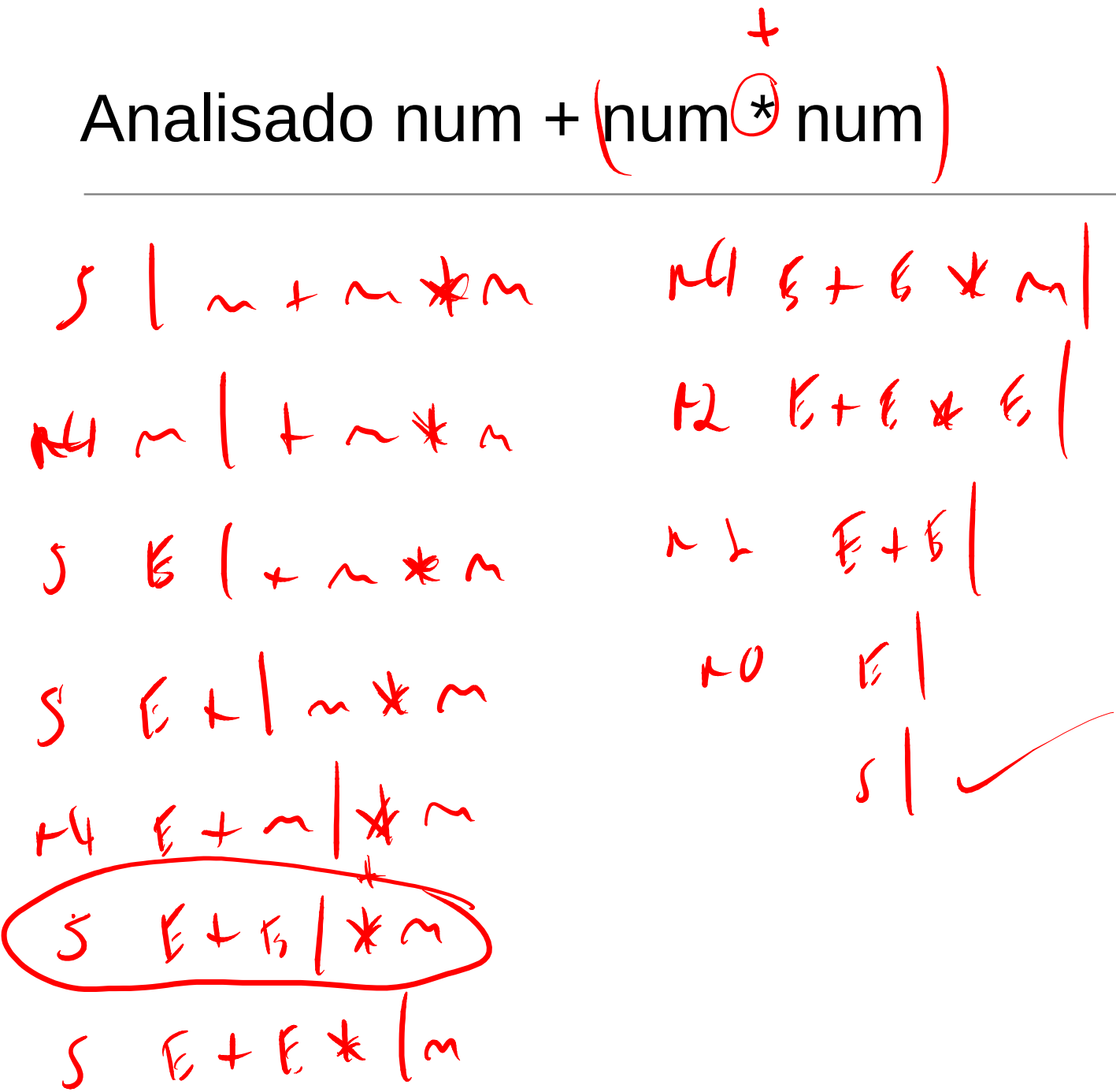
```
num + num * num
num * num + num
- num + num
```

Autômato SLR

```
S -> E
E -> E + E
E -> E * E
E -> - E
E -> num
```



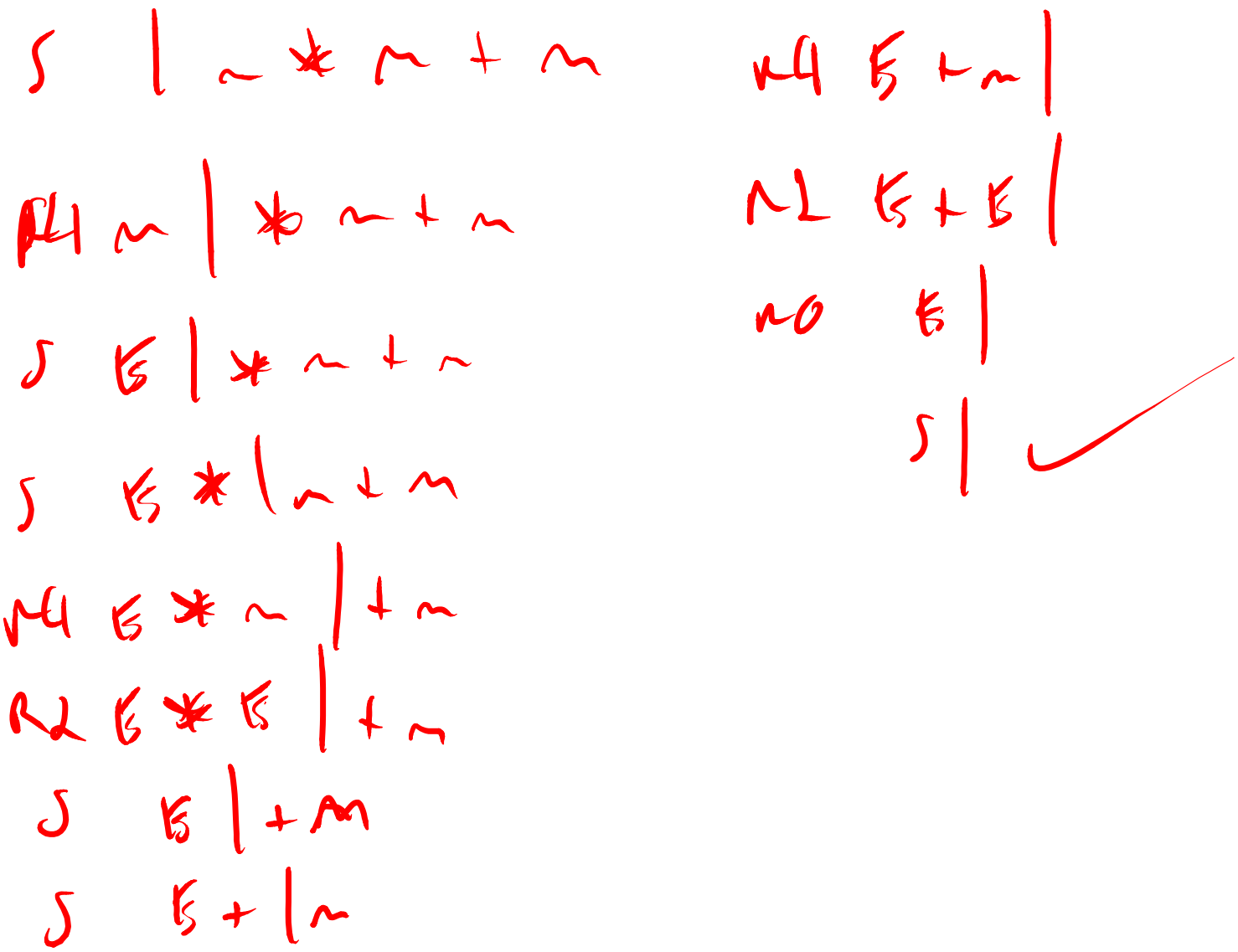
Analisado num + (num * num)



```
0 S -> E
1 E -> E + E
2 E -> E * E
3 E -> - E
4 E -> num
```

Analisando num * num + num

```
S -> E
E -> E + E
E -> E * E
E -> - E
E -> num
```

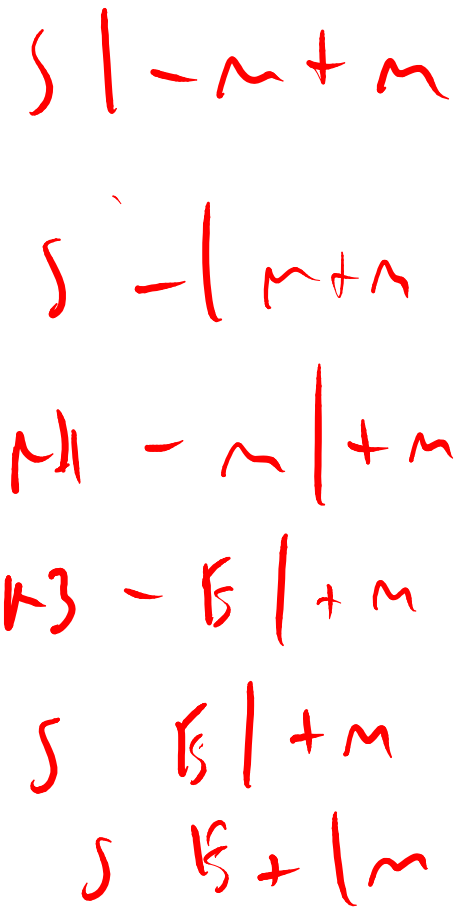


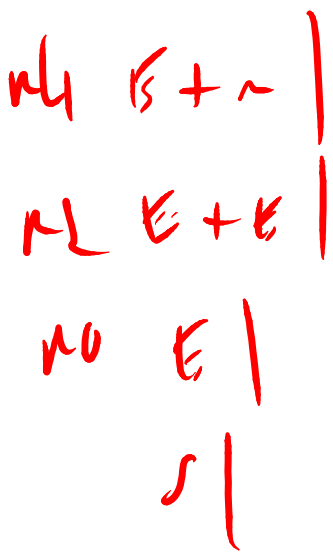
Controle de precedência

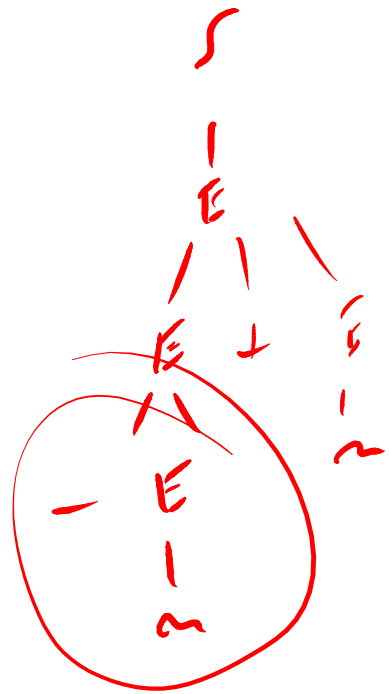
- Podemos levar em conta a precedência dos operadores na solução de conflitos shift-reduce
- Se o operador do shift tem precedência maior que a do operador do reduce, fazer shift, senão fazer o reduce
- Isso nos dá a árvore correta nos nossos exemplos, assumindo que a precedência de * é maior que a de +
- E quanto ao operador unário?

Analisando - num + num

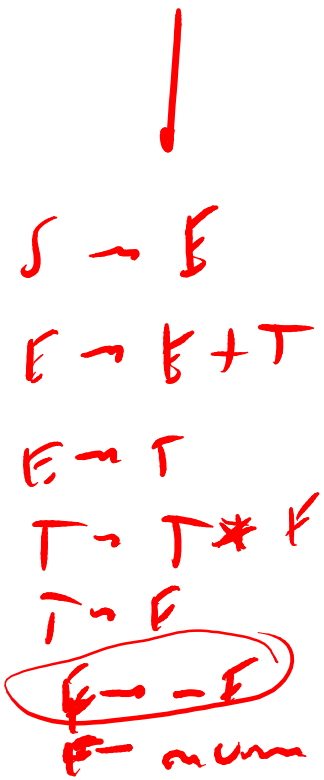
- Vai ser a mesma coisa, a precedência dele tem que ser maior que a dos operadores binários







```
S -> E
E -> E + E
E -> E * E
E -> - E
E -> num
```



Precedência e associatividade

- O controle da precedência e o da associatividade usam o mesmo mecanismo
- Podemos ter ambos no analisador: se um operador é associativo à direita é como se a precedência dele fosse maior do que a dele mesmo, e aí escolhemos shift
- Um resumo da resolução de conflitos shift-reduce:
  - Para o mesmo operador, shift dá associatividade à direita, reduce à esquerda
  - Para operadores diferentes, shift dá precedência ao próximo operador, reduce ao atual

Gramática SLR para TINY

- Podemos dar uma gramática mais simples para TINY se usarmos um analisador SLR com controle de precedência:

```
S -> CMDS
CMDS -> CMDS ; CMD
CMDS -> CMD
CMD  -> if EXP then CMDS end
CMD  -> if EXP then CMDS else CMDS end
CMD  -> repeat CMDS until EXP
CMD  -> id := EXP
CMD  -> read id
CMD  -> write EXP
```

```
EXP  -> EXP < EXP
EXP  -> EXP = EXP
EXP  -> EXP + EXP
EXP  -> EXP - EXP
EXP  -> EXP * EXP
EXP  -> EXP / EXP
EXP  -> ( EXP )
EXP  -> num
EXP  -> id
```