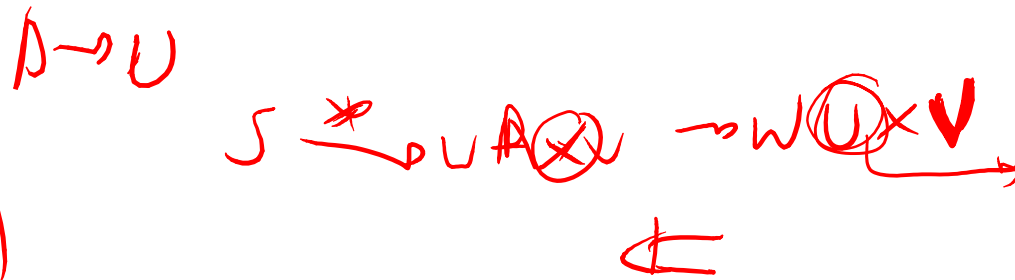


Compiladores - Análise SLR

Fabio Mascarenhas - 2013.2

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

Análise SLR (1)



- 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:

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

↑
↳ sufixo da entrada

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

Implicações da análise SLR

$$\boxed{\begin{array}{l} \alpha \rightarrow x \cdot \\ \beta \rightarrow y \cdot \end{array}} \quad \begin{array}{l} \text{FOLLOW}(\alpha) \cap \\ \text{FOLLOW}(\beta) = \emptyset \end{array}$$

- 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

$$\boxed{\begin{array}{l} E \rightarrow T \cdot \\ E \rightarrow E \cdot * T \end{array}} \quad * \notin \text{FOLLOW}(E)$$

Gramática de Expressões

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

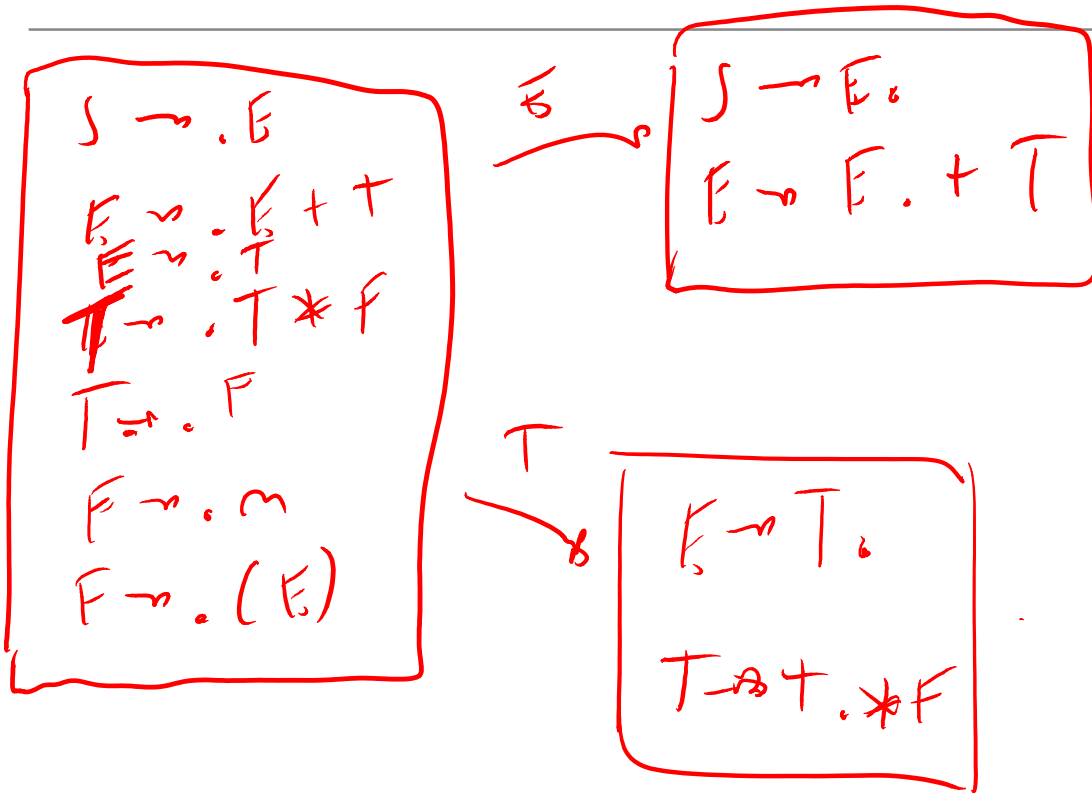
S \rightarrow E
E \rightarrow E + T
E \rightarrow T
T \rightarrow T * F
T \rightarrow F
F \rightarrow num
F \rightarrow (E)

- Podemos construir o autômato dela e verificar

Autômato da gramática de expressões

$$\text{Follow}(S) = \{ \langle \text{EOF} \rangle \}$$

$$+ \notin \text{Follow}(S)$$



$$\text{Follow}(E) = \{ +,), \langle \text{EOF} \rangle \}$$

$$* \notin \text{Follow}(E)$$

mpo há conflitos!

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

Resolvendo ambiguidade

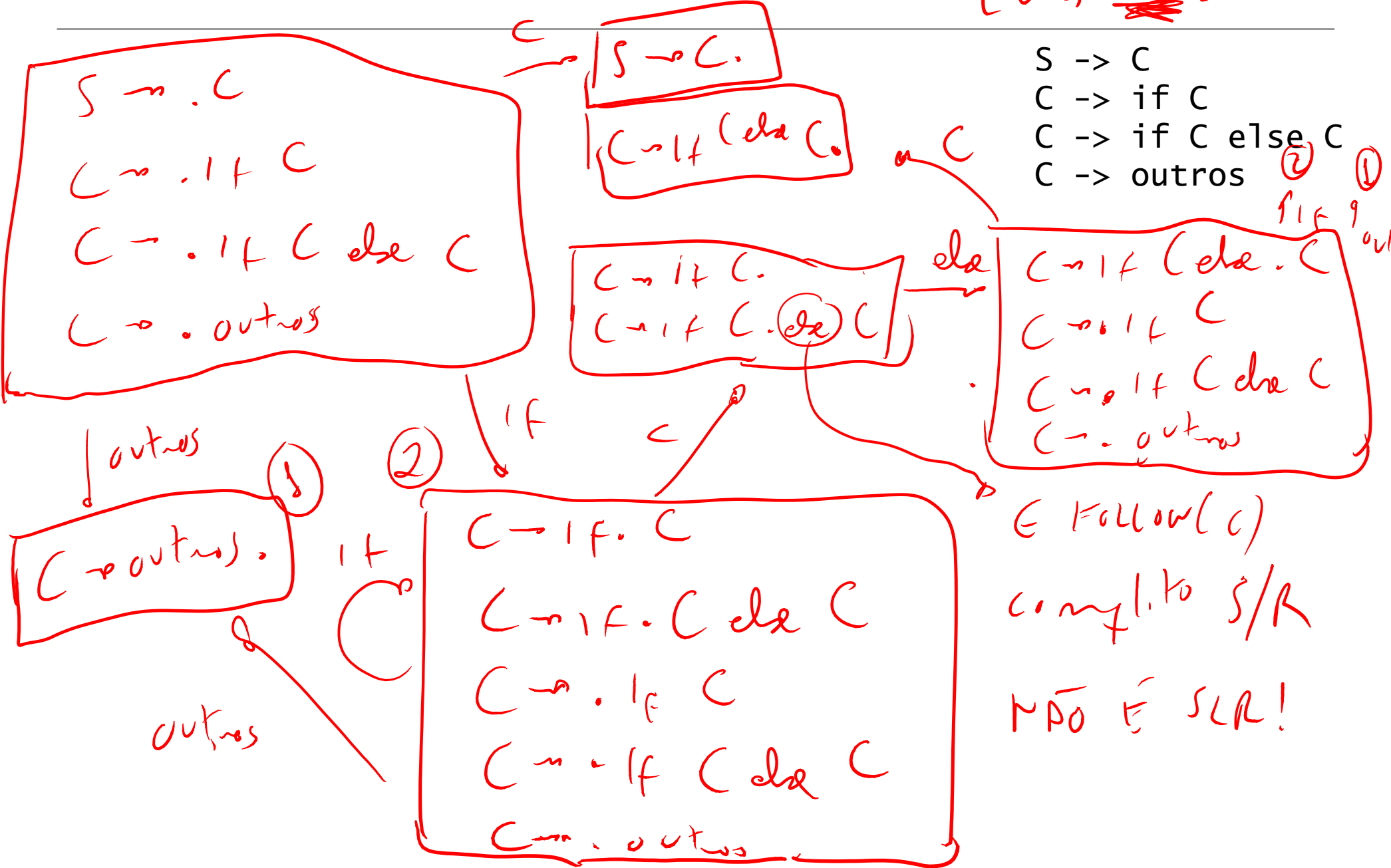
- Uma gramática ambígua nunca é SLR
- Vamos ver o que acontece com a ambiguidade do if-else:

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

Autômato da gramática do if-else

Follow(C) =
 { else, ~~C~~ }

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

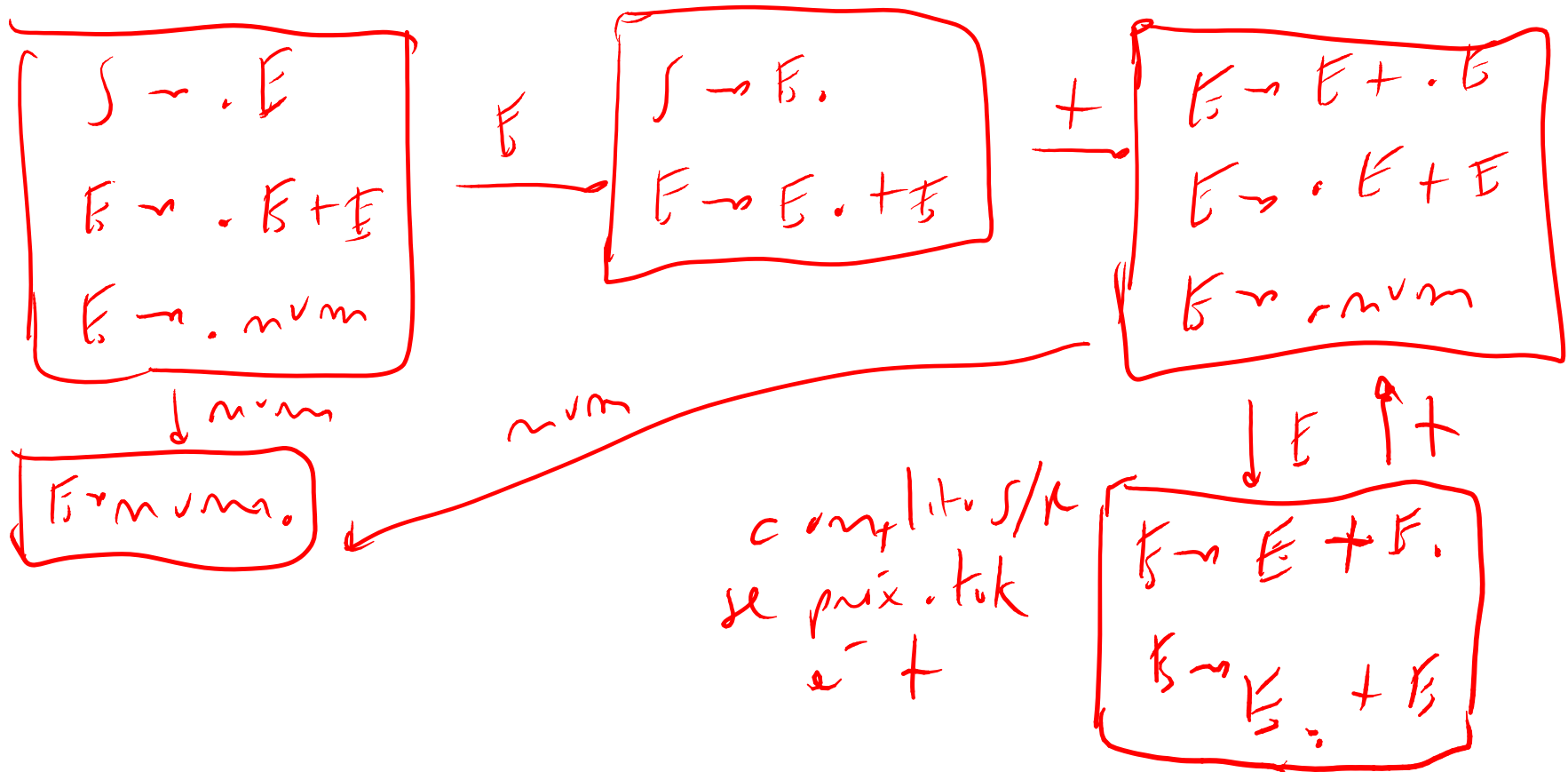
if (if outros else outros)

Gramáticas de expressões ambíguas

- Vamos agora examinar a gramática ambígua abaixo:

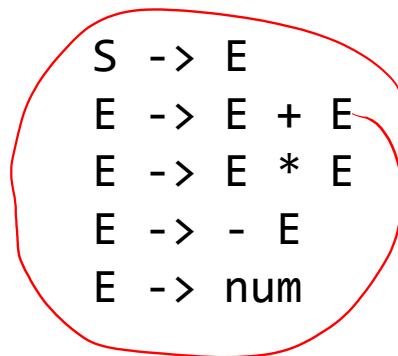
$S \rightarrow E$
 $E \rightarrow E + E$
 $E \rightarrow \text{num}$

$\text{follow}(S) = \{ \langle E \rangle \}$
 $\text{follow}(E) = \{ +, \langle E \rangle \}$



Precedência de operadores

- Vamos agora ver uma gramática mais complexa:

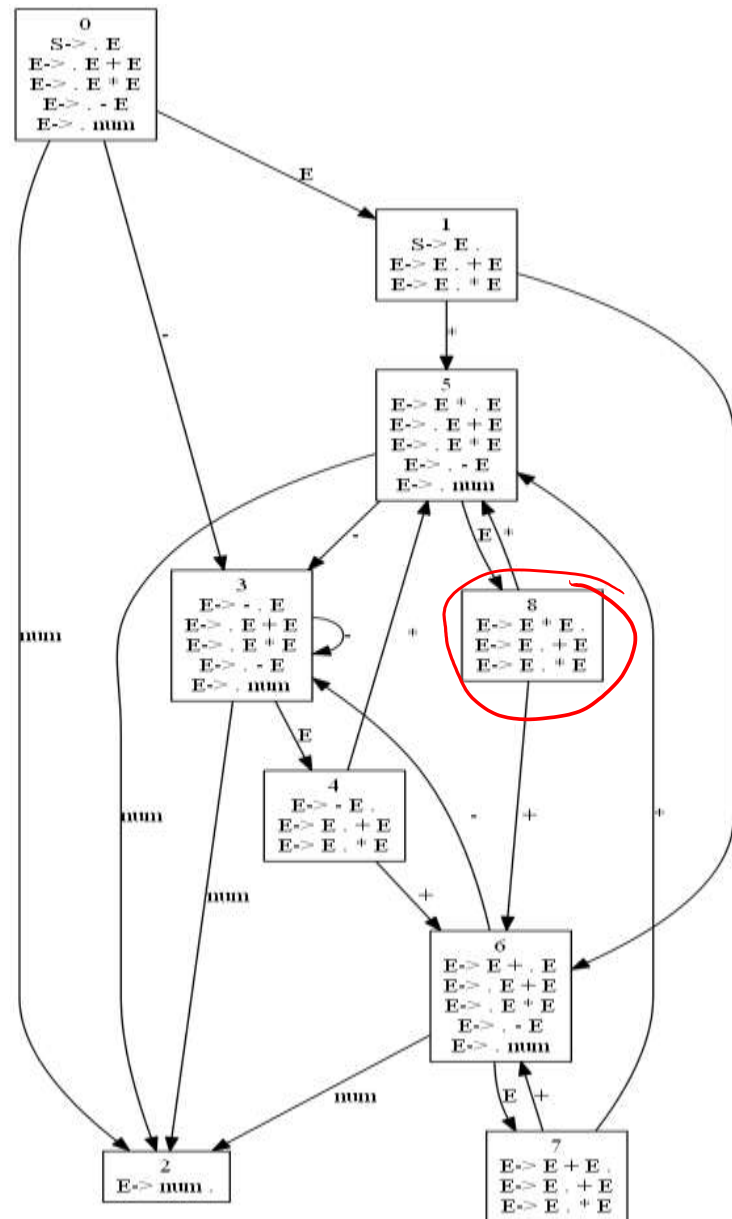


$S \rightarrow E$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow - E$
 $E \rightarrow \text{num}$

- Qual será o comportamento dessa gramática nas entradas:

• $\text{num} + \text{num} * \text{num}$
• $\text{num} * \text{num} + \text{num}$
• $- \text{num} + \text{num}$

Autômato SLR



$S \rightarrow E$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow - E$
 $E \rightarrow \text{num}$

Analisisado num + num * num

| n + n * n S

n | + n * n R

E | + n * n S

E + | n * n S

E + n | * n R

E + E | * n

S

R

R

E + E * | n S

E + E * n | R

E + E * E | R

E + E | R

E | R

S |

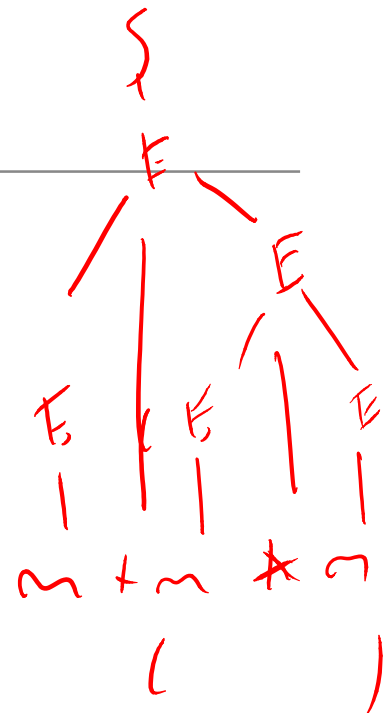
E | * n S

E * | n S

E * n | R

E * E | R

E | R S



Analizando num * num + num

| m * n + m | S S

m | * n + n | R

E | * n + m | S S

E * | n + m | S

E * n | + m | R

E * E | + n

E * E + | m | S

E * E + m | R

E * E + E | R

E * E | R

E | R
S | ✓

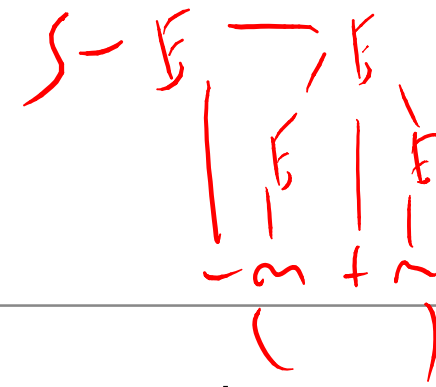
E | + m | S
E + | m | S

E + n | R E + E | R E | R S | ✓

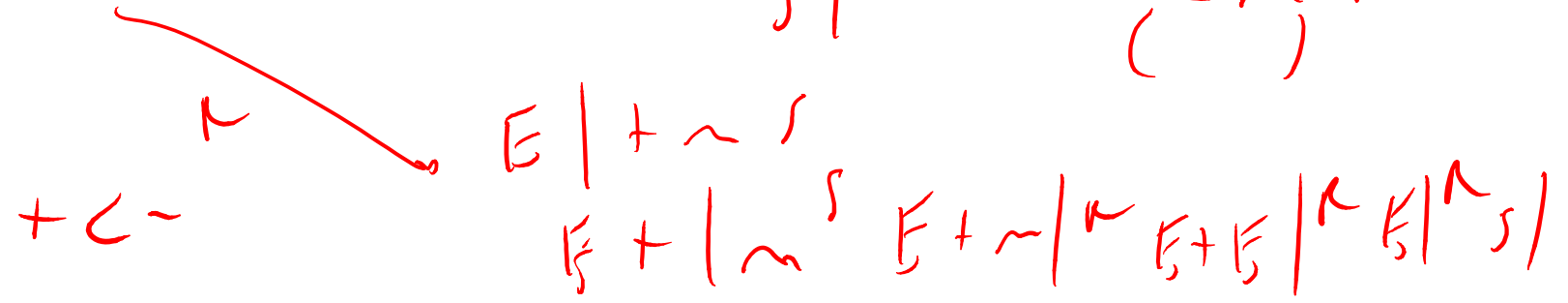
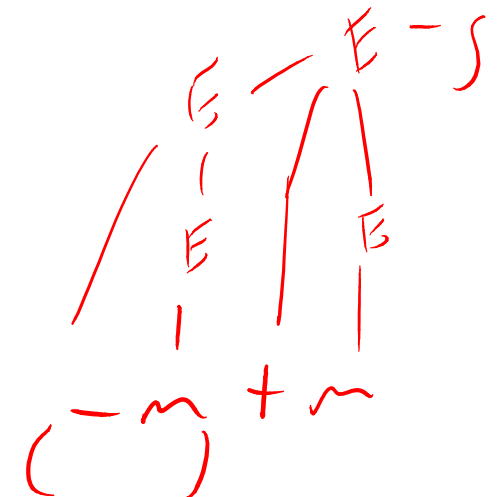
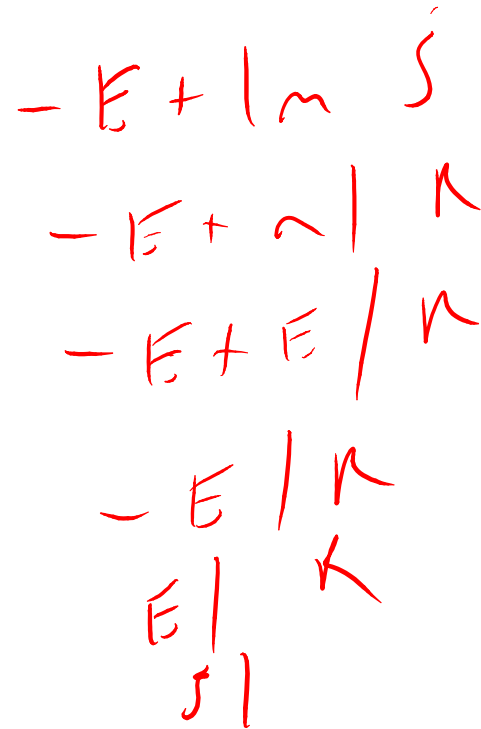
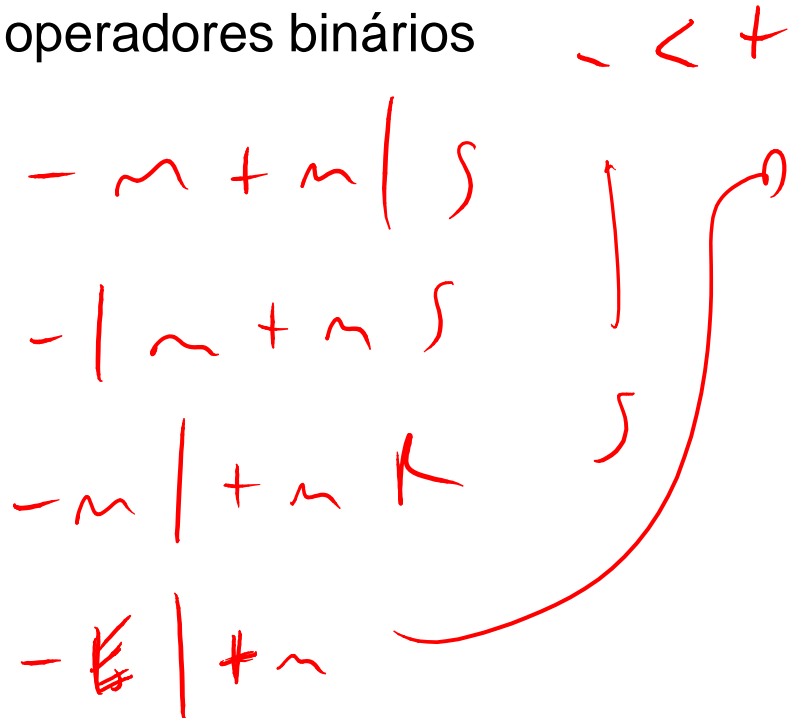
Controle de precedência

- 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?

Analizando - num + num



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



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	EXP -> EXP < EXP
CMDS -> CMDS ; CMD	EXP -> EXP = EXP
CMDS -> CMD	EXP -> EXP + EXP
CMD -> if EXP then CMDS end	EXP -> EXP - EXP
CMD -> if EXP then CMDS else CMDS end	EXP -> EXP * EXP
CMD -> repeat CMDS until EXP	EXP -> EXP / EXP
CMD -> id := EXP	EXP -> (EXP)
CMD -> read id	EXP -> num
CMD -> write EXP	EXP -> id

NÃO USAMOS EBNF