

Compiladores - Análise SLR


Fabio Mascarenhas - 2013.1

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

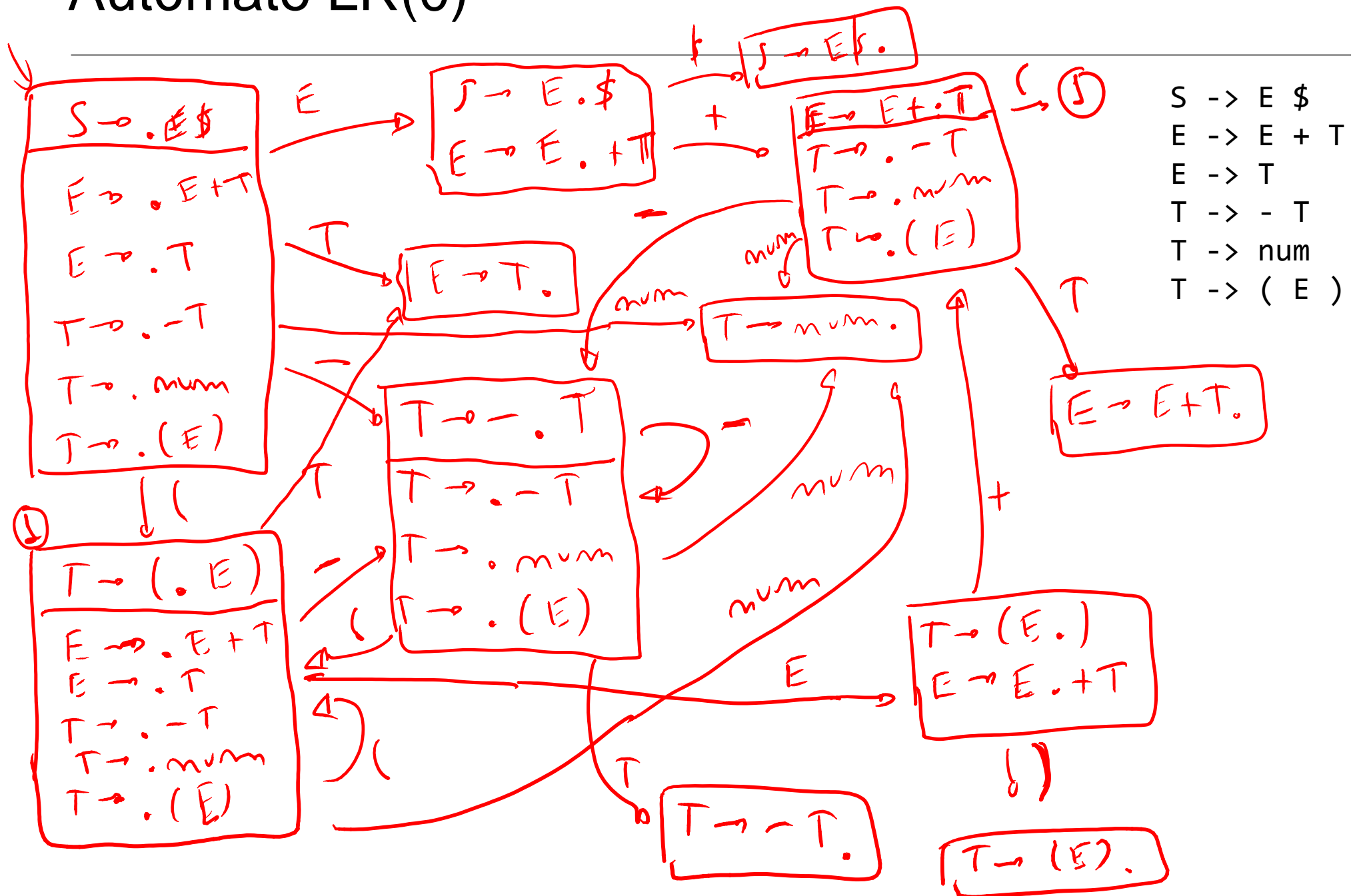
Um exemplo que funciona

- Todo estado com um item de redução e algum outro item causa conflito LR(0)!
- A técnica LR(0) é bem fraca, mas ainda assim existem gramáticas que ela consegue analisar mas que as técnicas de análise descendente não:

$S \rightarrow E \$$
 $E \rightarrow E + T$
 $E \rightarrow T$
 $T \rightarrow - T$
 $T \rightarrow \text{num}$
 $T \rightarrow (E)$

- Vamos construir o autômato de itens LR(0) dessa gramática e usá-lo para analisar - (num + num) + num 

Autômato LR(0)



Analizando uma entrada

$1 - (num + num) + num \$ S$
 $- 1 (num + num) + num \$ S$
 $- (1 num + num) + num \$ S$
 $- (num 1 + num) + num \$ S$
 $- (T 1 + num) + num \$ R$
 $- (E 1 + num) + num \$ R$
 $- (E + 1 num) + num \$ S$
 $- (E + num 1) + num \$ R$
 $- (E + num ||) + num \$ R$
 $- (E + T 1) + num \$ R$

$- (E 1) + num \$ S$
 $- (E) 1 + num \$ R$
 $- T 1 + num \$ R$
 $T 1 + num \$ R$
 $E 1 + num \$ S$
 $E + 1 num \$ S$
 $E + num 1 \$ R$
 $E + T 1 \$ R$
 $E 1 \$ S$
 $E 1 \$ R$

✓ok

(51)

Análise SLR

terminal
 $A \rightarrow x \cdot (y) \xrightarrow{\text{terminal}} A \rightarrow x y \cdot$
 $A \rightarrow w \cdot$
FOLLOW

- 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)!

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~~ 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

$E \rightarrow T.$
 $T \rightarrow T.*F$

$\rightarrow \text{FOLLOW}(F) = \{+, -, \text{EOF}\}$

Gramática de Expressões

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

$S \rightarrow E$

$E \rightarrow E + T$

$E \rightarrow E - T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

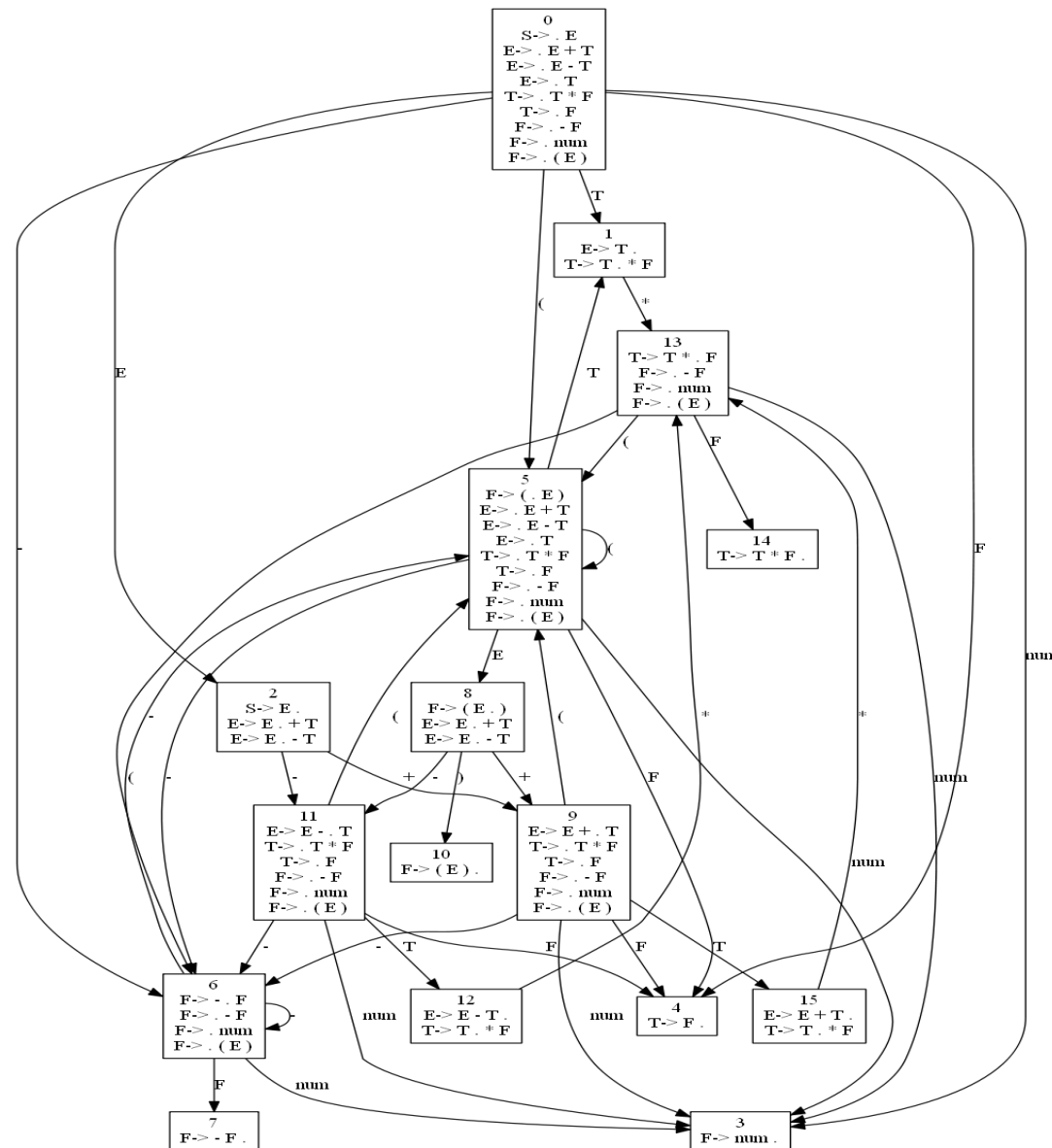
$F \rightarrow - F$

$F \rightarrow \text{num}$

$F \rightarrow (E)$

- Podemos construir o autômato dela e verificar

Autômato da gramática de expressões



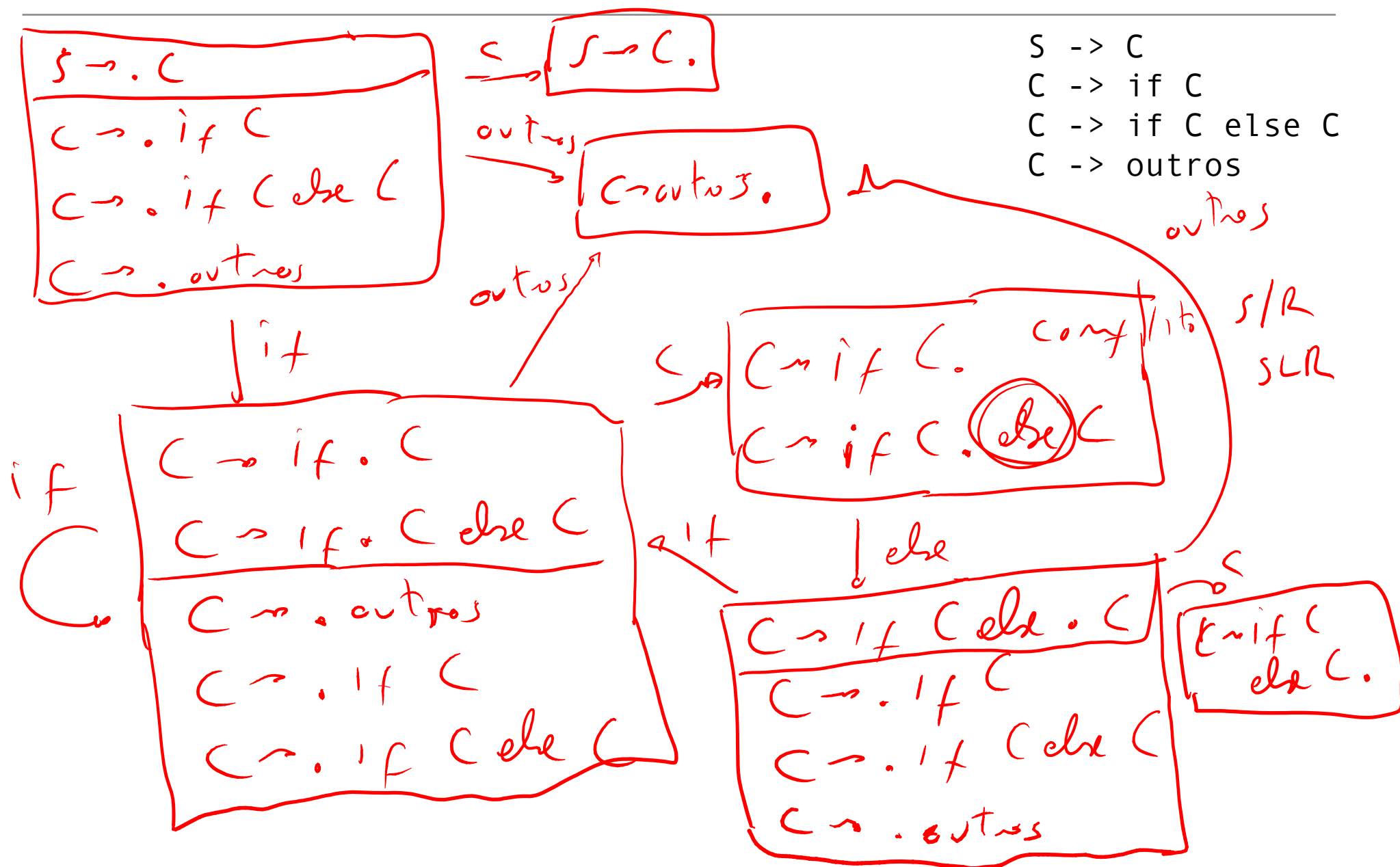
Resolvendo ambiguidade

- 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) =
{EOF, ϵ }

others

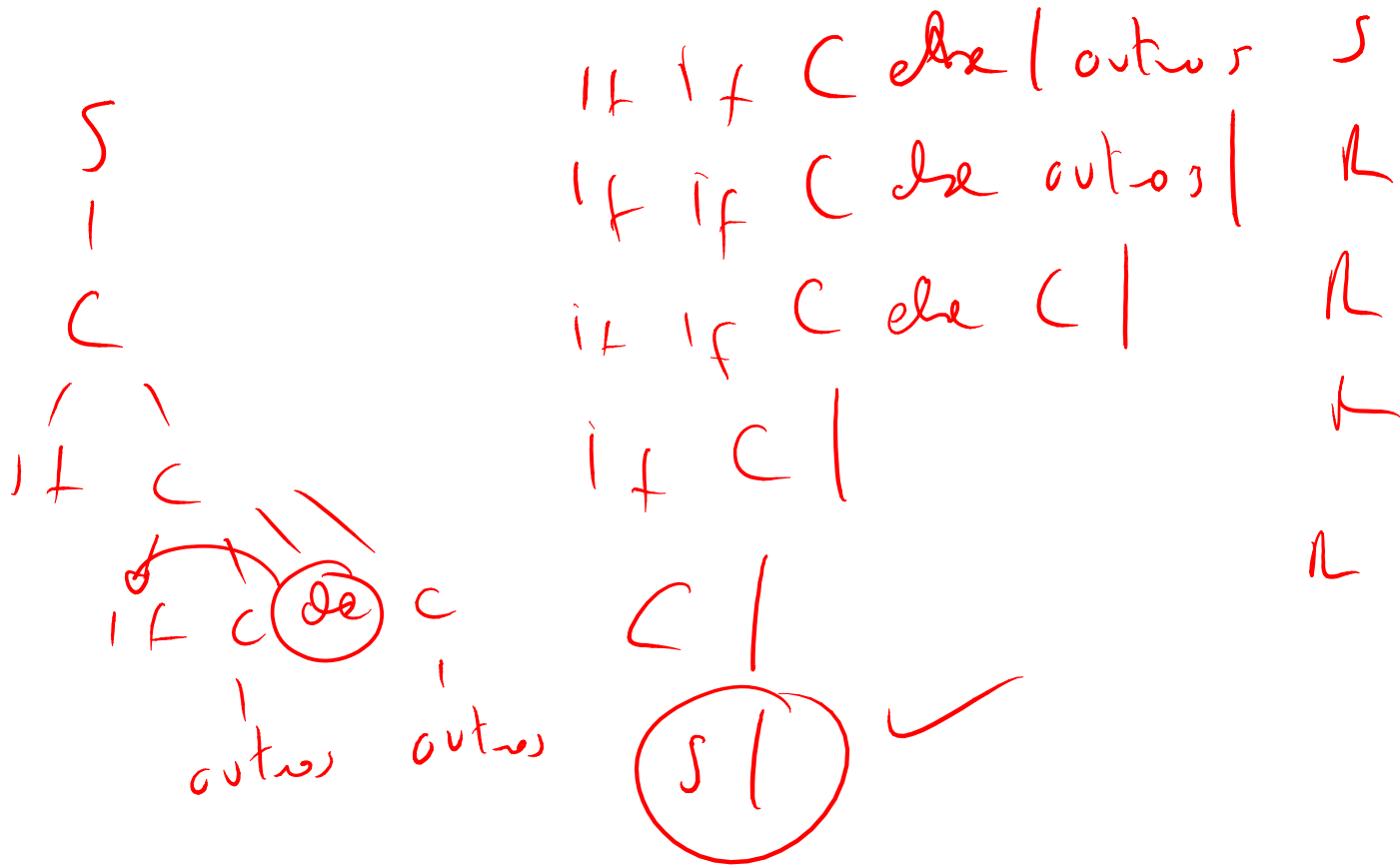


Resolução de conflitos

- 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 S
 if | if outros ~~else~~ outros S
 if if | outros ~~else~~ outros S
 if if outros | ~~else~~ outros R
 if if C | ~~else~~ outros — (S) — R

Analizando uma entrada ambígua

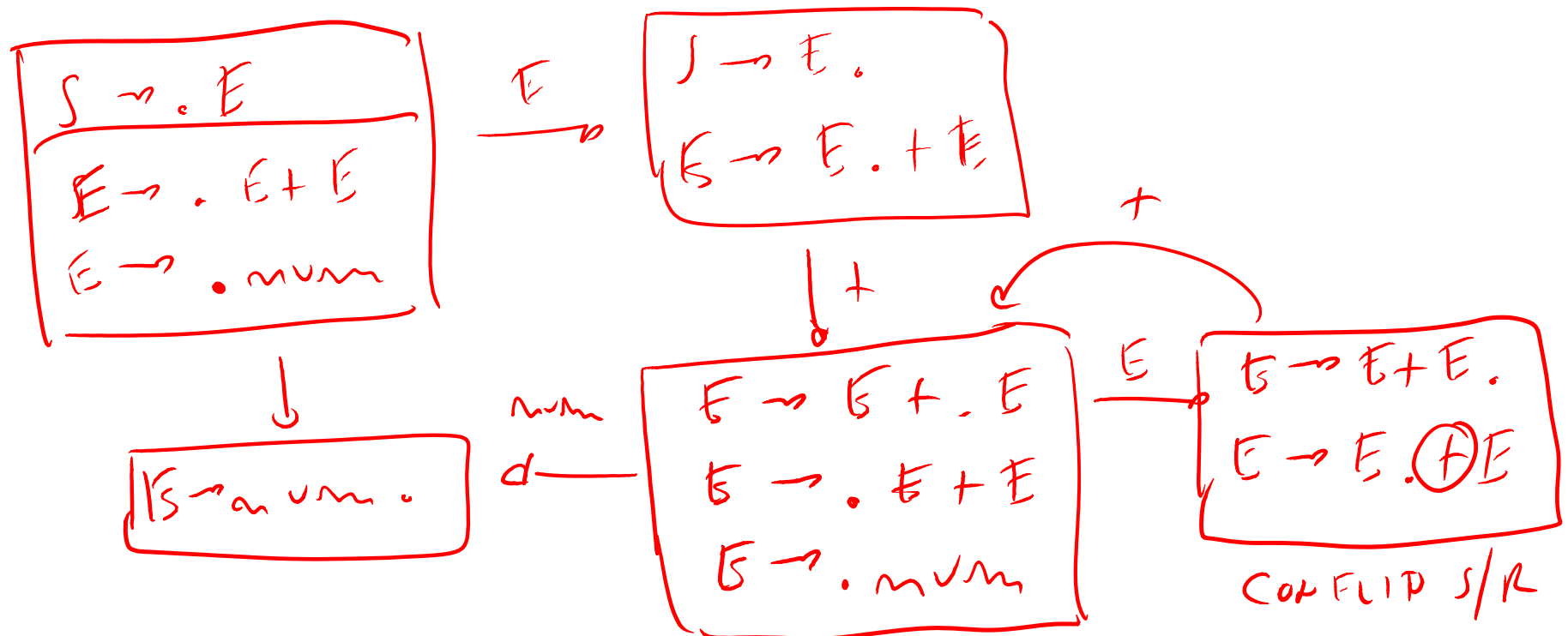


Gramáticas de expressões ambíguas

- Vamos agora examinar a gramática ambígua abaixo:

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

Follow(E):
 $\{E, +, \text{num}\}$



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) \rightarrow associativo à direita!



| num + num + num S

num | + num + num R

E | + num + num S

E + | num + num S

E + num | + num R

(E + E | + num S)

E + E + | num S

E + E + num | R

E + E + E | R

conflicto E + E | R

(S |) R

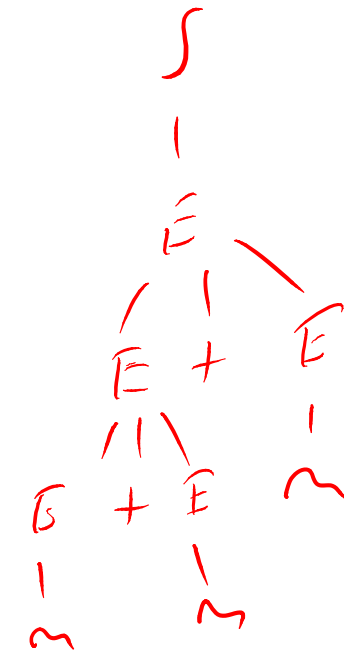
Controlando a associatividade

$1 + (1 + \text{outros})$ de outros
 $1 + (1 + \text{outros de outros})$

- Agora vamos ver como fica o mesmo exemplo se resolvermos o conflito escolhendo a redução ao invés do shift

$E \mid + \text{num } S$
 $ES + \mid \text{num } S$
 $ES + \text{num} \mid R$
 $ES + E \mid R$
 $ES \mid R$
 $(S \mid) R$

$(\text{num} + \text{num}) + \text{num}$



assoc. à esquerda!

Precedência de operadores

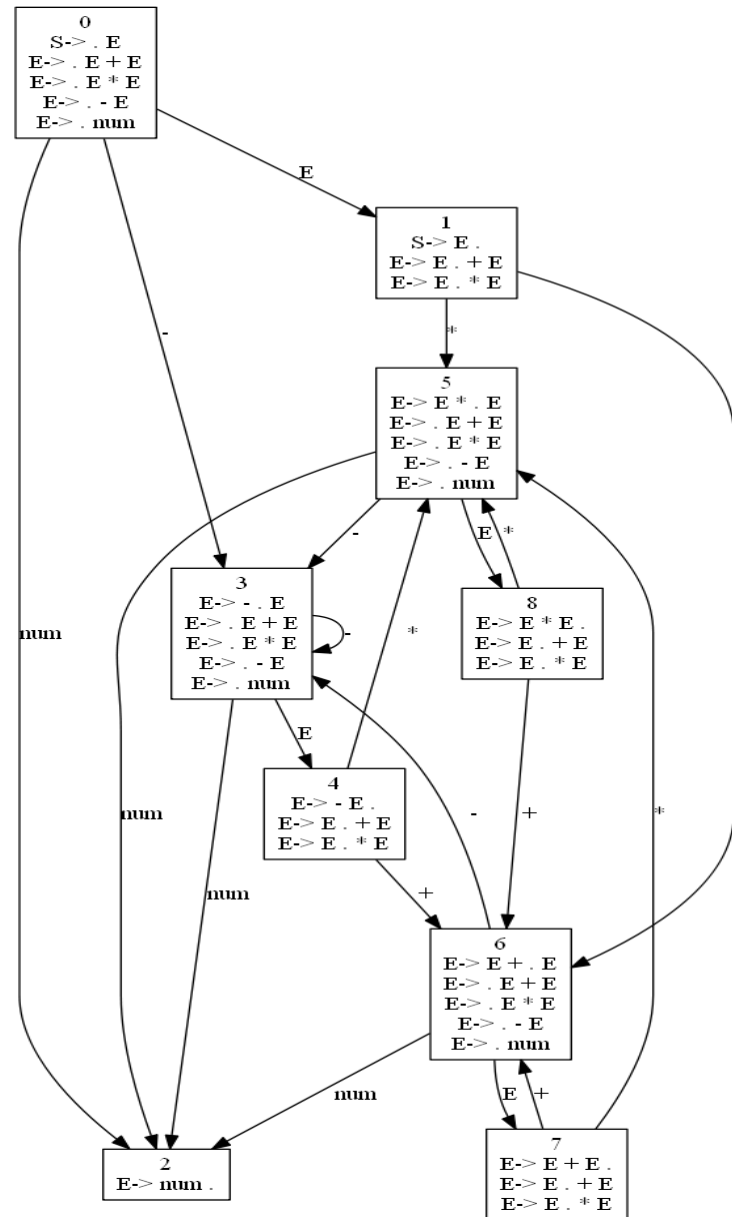
- Vamos agora ver uma gramática mais complexa:

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow - E \\ E &\rightarrow \text{num} \end{aligned}$$

- Qual será o comportamento dessa gramática nas entradas:

$$\begin{aligned} &\text{num} + \text{num} * \text{num} \\ &\text{num} * \text{num} + \text{num} \\ &- \text{num} + \text{num} \end{aligned}$$

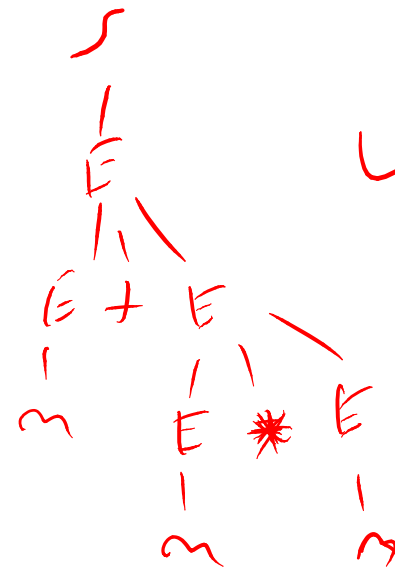
Autômato SLR



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

Analísado num +(num * num)

$| \text{num} + \text{num} * \text{num} \quad S$
 $\text{num} \mid + \text{num} * \text{num} \quad (R)$
 $E \mid + \text{num} * \text{num} \quad S$
 $T \mid + \text{num} * \text{num} \quad S$
 $E + \text{num} \mid * \text{num} \quad (R)$
 $(E + T \mid * \text{num}) \quad S$
 $E + E \mid \text{num} \quad S$
 $T + E * \text{num} \mid R$
 $T + E * E \mid R$
 $E + E \mid R$
 $E \mid R \quad (S)$



Analizando $(num * num) + num$

$| num * num + num \ S$

$num | * num + num \ R$

$E | * num + num \ S$

$E * | num + num \ S$

$E * num | + num \ R$

$E * E | + num \ S$

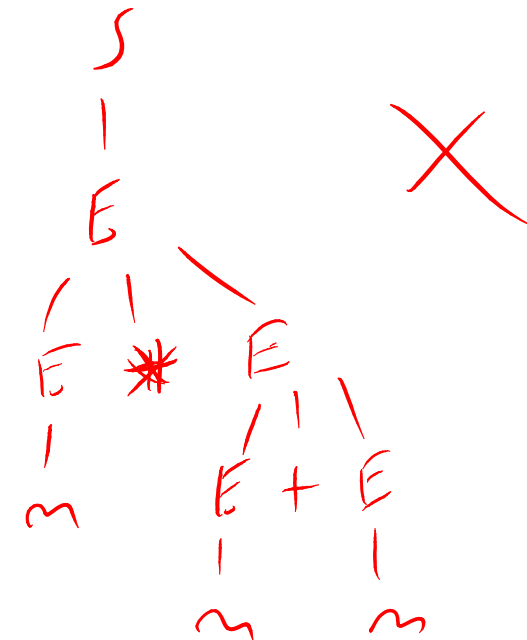
$E * E + | num \ S$

$E * E + num | \ R$

$E * E + E | \ R$

$E * E | \ R$

$E | \ R \ (S)$



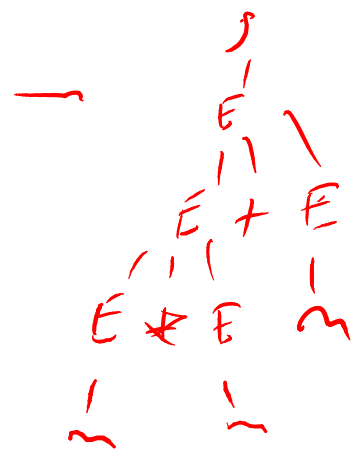
$E | + num \ S$

$E + num \ S$

$E + num | \ R$

$E + E | \ R$

$E | \ R \ (S)$



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

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

Otimizando o analisador SLR

- A implementação do analisador SLR não precisa executar o autômato em toda a pilha sempre
- Podemos associar um número de estado a cada elemento da pilha (com outra pilha, por exemplo), para ser o estado onde o autômato se encontra quando percorreu a pilha até aquele elemento
- Um shift empilha o estado resultante de fazer a transição do estado que estava no topo da pilha antes do shift
- Um reduce empilha o estado resultante de fazer a transição do estado que estava no topo da pilha depois de desempilhar o lado direito

Tabelas ACTION e GOTO

- Podemos construir uma grande tabela a partir do autômato, e guiar o analisador a partir dessa tabela
- As linhas são estados, as colunas símbolos (terminais e não-terminais)
- A parte da tabela dos terminais se chama ACTION
 - Ela diz o que o autômato deve fazer se o próximo token for o terminal
- A parte dos não-terminais se chama GOTO
 - Ela diz para qual estado ir após uma redução para aquele não-terminal

Preenchendo a tabela

- Para cada estado:
 - Transições em terminais viram entradas S_n para aquele terminal, onde n é o estado de destino (ACTION)
 - Transições em não-terminais viram entradas n para aquele não-terminal (GOTO)
 - Itens de redução viram entradas R_n para todos os terminais no FOLLOW do não-terminal da regra, onde n é o número de regra (ACTION)
 - Itens de redução para o símbolo inicial da gramática e o final da entrada geram entradas A , para *accept* (ACTION)

Tabelas ACTION e GOTO

- Tabela para a gramática:
 - $S \rightarrow E \$$
 - $E \rightarrow E + T$
 - $E \rightarrow T$
 - $T \rightarrow - T$
 - $T \rightarrow \text{num}$
 - $T \rightarrow (E)$

Analísadores LR de tabela

- Buracos na tabela indicam erros sintáticos
- Tentar adicionar uma entrada em uma célula já preenchida é um conflito, usar as regras para resolução
- Todos os métodos LR com um token de lookahead usam a mesma estrutura de tabela, o que varia é só o método de preenchimento, e o tamanho da tabela no caso da análise LR(1)
- As tabelas para analisadores LR(0), SLR e LALR de uma dada gramática têm o mesmo tamanho

Limitações do método SLR

- Existem gramáticas que não são SLR:

```
S -> C
C -> id
C -> V := E
V -> id
E -> V
E -> n
```

- Existem métodos de análise mais poderosos
- LALR associa um conjunto similar ao FOLLOW para cada item, mas mais preciso que o FOLLOW
- LR(1) e LR(k) mudam o conceito de item, gerando um autômato maior e mais preciso