Compiladores - Gramáticas

Fabio Mascarenhas – 2015.2

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

Ambiguidade

- Uma gramática é *ambígua* se existe alguma cadeia para qual ela tem mais de uma *árvore sintática*
  - De maneira equivalente, se existe mais de uma derivação *mais à esquerda* para uma cadeia
  - Ou se existe mais de uma derivação *mais à direita* para uma cadeia
  - As três definições são equivalentes
- Ambiguidade é ruim para uma linguagem de programação, pois leva a interpretações inconsistentes entre diferentes compiladores

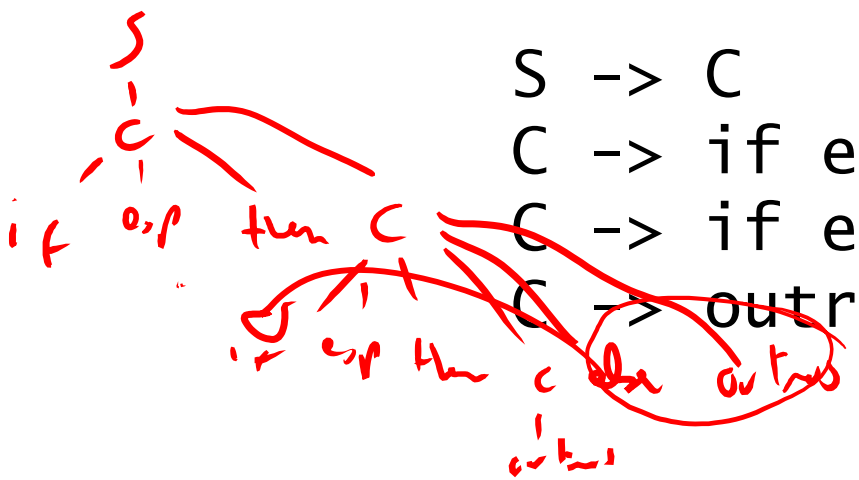
Detectando ambiguidade

- Infelizmente, não existe um algoritmo para detectar se uma gramática qualquer é ambígua ou não

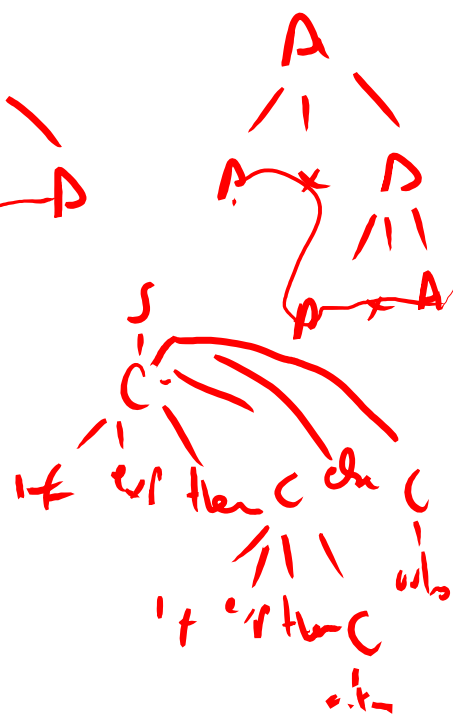
- Mas existem *heurísticas*, a principal delas é verificar se existe uma regra misturando *recursão à esquerda* e *recursão à direita*

  - É o caso da gramática de expressões

  - Às vezes isso é bem sutil: ambiguidade do if-else



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



Removendo ambiguidade

- Do mesmo modo, não há um algoritmo para remover ambiguidade
- Se a ambiguidade está na gramática, e não na própria linguagem, o jeito é encontrar a *fonte* da ambiguidade e reescrever a gramática para eliminá-la
- No caso de ambiguidade em gramáticas de expressões e operadores, a ambiguidade vem da gramática não estar levando em conta as regras de associatividade e precedência dos operadores
- Em uma gramática de expressões, cada nível de precedência tem que ganhar seu próprio não-terminal
- Operadores que devem ser associativos à esquerda precisam usar recursão à esquerda, e associativos à direita precisam de recursão à direita

Expressões simples, sem ambiguidade

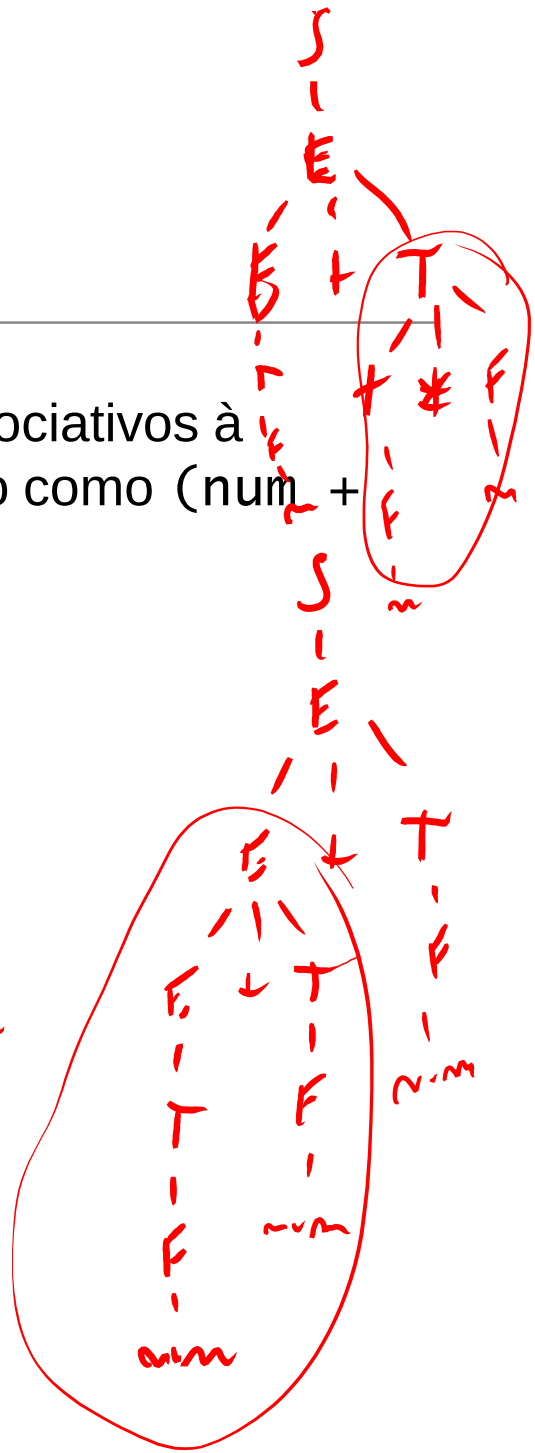
- Assumindo que * tem precedência sobre +, e ambos são associativos à esquerda (ou seja, `num + num + num` deve ser interpretado como `(num + num) + num`)

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

nível 1

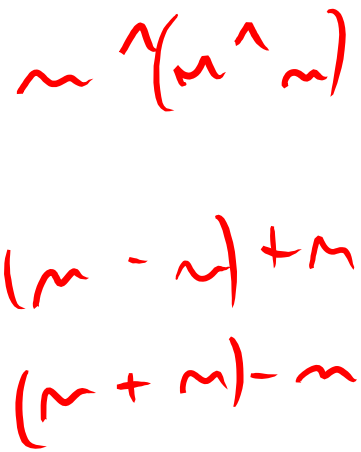
nível 2

nível 3



Expressões simples, sem ambiguidade

- Assumindo que ^ tem precedência sobre * que tem precedência sobre +, ^ é associativo à direita, * e + são associativos à esquerda (ou seja, `num + num + num` deve ser interpretado como `(num + num) + num`)



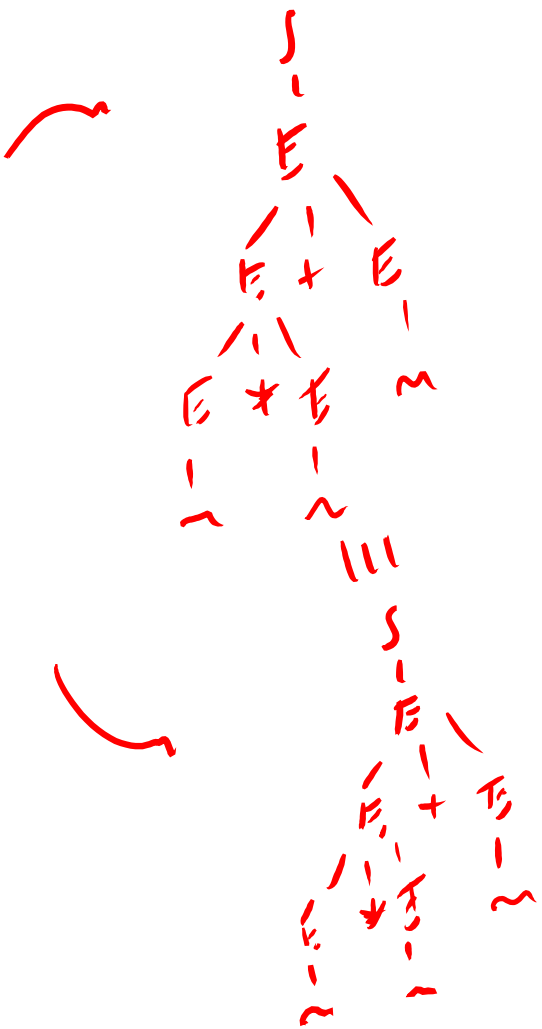
```
S -> E
E -> E + T
E -> E - T
E -> T
T -> T * F
T -> F
F -> A ^ F
F -> A
A -> ( E )
A -> num
```

Duas derivações, uma árvore

- Duas derivações de uma frase podem dar a mesma árvore de uma for mais à esquerda e outra mais à direita

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

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

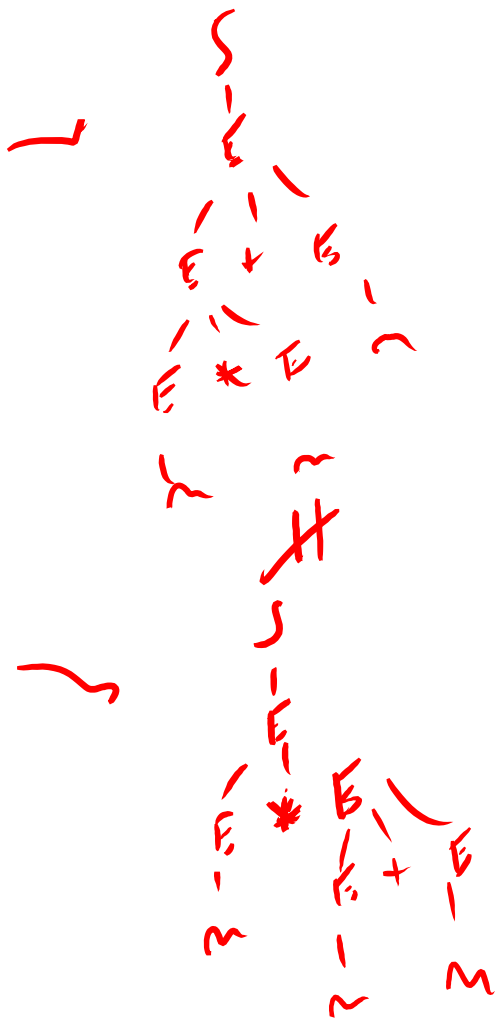


Duas derivações, duas árvores

- Duas derivações mais à esquerda dão duas árvores diferentes: gramática ambígua

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

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



If-else sem ambiguidade

- Uma solução adotada por diversas linguagens é acrescentar uma delimitador que fecha o if:

```
S -> C
C -> if exp then C end
C -> if exp then C else C end
C -> outros
```

- Uma desvantagem é que agora é necessário ter uma construção “else-if” para ter ifs em cascata sem uma multiplicação de ends

- E claro, estamos mudando a linguagem!

If-else sem ambiguidade, com a “cara de C”

- Uma solução adotada por diversas linguagens é acrescentar uma delimitador que fecha o if:

```
S -> C
C -> if exp { C }
C -> if exp { C } else { C }
C -> outros
```

- Uma desvantagem é que agora é necessário ter uma construção “else-if” para ter ifs em cascata sem uma multiplicação de ends

- E claro, estamos mudando a linguagem!

If-else sem mudar a linguagem

- Outra solução é separar os ifs em dois tipos, com não-terminais diferentes:

```
S  -> C
C  -> if exp then C
C  -> if exp then CE else C
C  -> outros
CE -> if exp then CE else CE
CE -> outros
```

- Notem a semelhança com a gramática não ambígua de expressões

Quiz

- Qual a versão não ambígua da gramática:

```
S -> S S
S -> a
S -> b
```

```
S -> S a
S -> S b
S ->
```

```
S  -> S S'
S' -> a
S' -> b
```

```
S  -> S
S  -> S'
S' -> a
S' -> b
```

```
S -> S a
S -> S b
S -> a
S -> b
```

Quiz

- Qual a versão não ambígua da gramática:

```
S -> S S
S -> a
S -> b
```

```
S -> S a
S -> S b
S ->
```

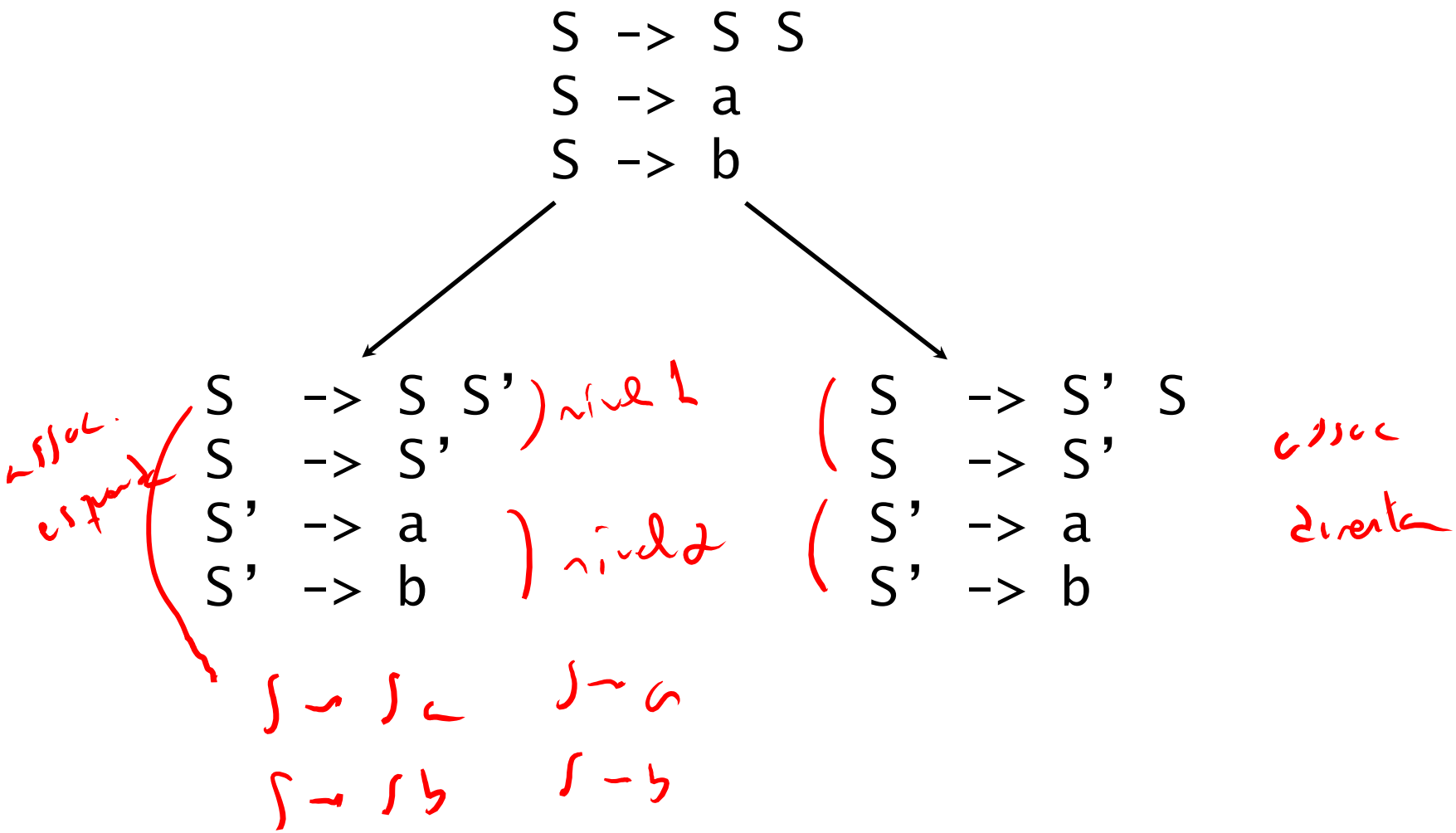
```
S  -> S S'
S' -> a
S' -> b
```

```
S  -> S
S  -> S'
S' -> a
S' -> b
```

```
* S -> S a
  S -> S b
  S -> a
  S -> b
```

Quiz

- Qual a versão não ambígua da gramática:



Contornando ambiguidade

- Na prática, um uso judicioso de ambiguidade pode simplificar a gramática, e deixar ela mais natural
- Tanto a versão ambígua da gramática de expressões simples quanto a gramática do if-else são mais simples que suas versões não ambíguas!
- Podemos eliminar a ambiguidade não na gramática, mas na *implementação do analisador sintático*
- As ferramentas de geração de analisadores possuem regras de eliminação de ambiguidade, e diretivas de precedência e associatividade que permitem controlar como essa eliminação é feita