Análise Sintática

Parte I – Conceitos preliminares

Sintaxe

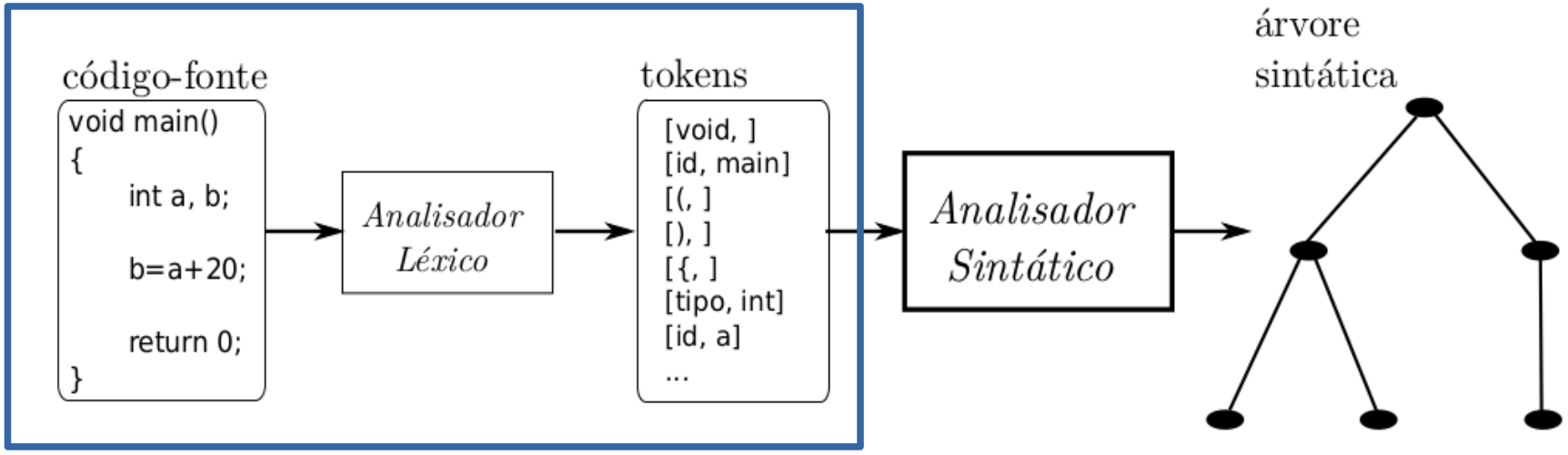
sin·ta·xe

(grego *súntaksis*, -eós, **ordenação, disposição, arranjo**) substantivo feminino

1) [Linguística] Parte da linguística que se dedica ao estudo das regras e dos princípios que regem a organização dos constituintes das frases.

2) [Informática] Conjunto de regras que regem a escrita de uma linguagem de programação.

Papel do Analisador Sintático



Já vimos anteriormente...

Papel do Analisador Sintático

- O analisador sintático (ou *parser*) é o segundo componente do *front-end* do compilador
- Recebe uma sequência de tokens do analisador léxico e determina se essa sequência é ou não uma sentença sintaticamente válida na linguagem-fonte
- Como saída se tem uma árvore sintática que representa a estrutura sintática do programa fonte
- Também é papel do analisador sintático sinalizar e/ou se recuperar de erros

Estrutura Sintática

- A estrutura sintática de um programa relaciona cada parte do programa com suas sub-partes componentes

- Exemplo:
  - Um comando *if* completo tem 3 partes que o compõe:

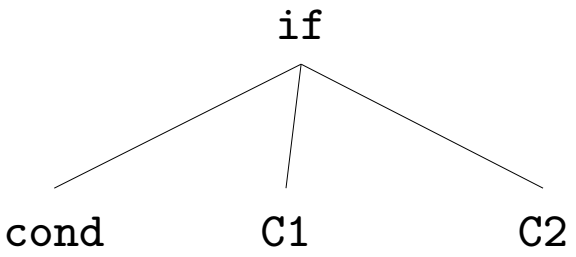
  1)Condição de teste

  2)Um comando para executar se a condição for verdadeira

  3)Um comando para executar se a condição for falsa

Estrutura Sintática

- Quando o compilador identifica um *if* no programa, é importante que ele possa acessar esses componentes para poder gerar código executável para o *if*
- Por isso, não basta apenas agrupar os caracteres em tokens, é preciso também agrupar os tokens em ***estruturas sintáticas***



Especificação da sintaxe

Gramáticas Livres de Contexto

A sintaxe de uma linguagem de programação é normalmente dada pelas regras gramaticais de uma gramática livre de contexto (GLC)

Especificação da sintaxe

Gramáticas Livres de Contexto

- A definição formal de uma gramática livre de contexto pode ser representada através dos seguintes componentes:

$$G = N, T, P, S$$

onde:

N - Conjunto finito de símbolos não-terminais

T - Conjunto finito de símbolos terminais

P - Conjunto de regras de produções

S - Símbolo inicial da gramática

Especificação da sintaxe

Gramáticas Livres de Contexto

Terminologias:

- **Símbolos terminais**: Conjunto finito de símbolos básicos que formam as palavras da linguagens, são representadas pelos *tokens reconhecidos pelo analisador léxico*
- **Símbolos não-terminais**: Conjunto finito de *variáveis utilizadas para representar os conjuntos da linguagem*, são formadas pelos terminas e pelos próprios símbolos não-terminais
- **Símbolo inicial**: Símbolo não-terminal, que representa o inicio da definição da linguagem
- **Regras de produção**: Conjunto de regras sintáticas que representam a definição da linguagem, indicam como símbolos terminais e não-terminais podem ser combinados

Especificação da sintaxe

Gramáticas Livres de Contexto

- As regras de produção P são representadas da seguinte forma:

$$A \rightarrow \alpha$$

onde:

- **‘A’** é uma variável - simbolo não-terminal
- **‘→’** é símbolo de produção ("pode ser derivado em")
- **‘α’** é uma combinação de símbolos terminais e não-terminais que representam a forma como uma string vai ser formada

Especificação da sintaxe

Gramáticas Livres de Contexto

Exemplo:

- A gramática G a seguir gera expressões aritméticas:

$$G = (\{E\}, \{+, -, *, /, (,), id, num\}, P, E)$$

sendo as produções P:

E → E + E |
E - E |
E * E |
E / E |
(E) |
id |
num

Observação:
No caso dos tokens *num* e *id*, o valor deles **não aparece** na gramática porque não é relevante para a estrutura sintática da linguagem!!

- Para a linguagem de expressões, temos tokens de quatro tipos: números (num), identificadores (id), operadores (+,-,*,/) e pontuação ((,))

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

BNF - Backus-Naur Form

- Notação mais usada para descrever formalmente as linguagens de programação
- "Metalinguagem" (linguagem usada para descrever linguagens)
- Na BNF, abstrações são usadas para representar classes de estruturas sintáticas
  - Agem como variáveis sintáticas (*não-terminais*)

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

BNF - Backus-Naur Form

- Não-terminais: BNF abstrações
- Terminais: lexemas e tokens
- Gramática: uma coleção não vazia de regras

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

BNF - Backus-Naur Form

- Símbolos delimitados por < > indica um não-terminal- termo que precisa ser expandido
- Símbolos não delimitados por < > são terminais
- Os símbolos "::=" ou "→" significam "*é definido como*" ou "pode se derivado em"
- O símbolo | significa "ou"
  - Usado para separar alternativas

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

Exemplo:

```
<program>::=<stmts>

<stmts>::=<stmt> | <stmt> ; <stmts>

<stmt>::=<var> = <expr>

<var>::= a | b | c | d

<expr>::=<term> + <term> | <term> - <term>

<term>::=<var> | const
```

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

E-BNF - Extended Backus-Naur Form

- Notação que acrescenta meta-símbolos adicionais à notação BNF
  - [ ] → opcionalidade
  - { } → repetição

Exemplo:
E-BNF Pascal (simplificado)

```
<programa> ::= PROGRAM <identificador> ; <bloco> .
<bloco> ::= <declarações> BEGIN <comando>  END | BEGIN  <comando> END
<comando> ::=  <variável> := <expressão> ; <comando>
            | READ <variável> ; <comando>
            | WRITE <expressão> ; <comando>
            | IF <expressão> THEN <comando> ELSE <comando>
            | FOR <identificador> := <expressão> TO <expressão> DO <comando>
            | BEGIN <comando> END
 <declarações>   ::= VAR <lista_identif.> : <tipo>;
                   | TYPE <identificador> = <tipo> ;
<expressão>      ::= <variável>
                   | <identificador>
                   | <constante>
                   | <expressão> <operador binário> <expressão>
<variável>       ::= <identificador>
<operador_bin>::=  +| - | * | / | = | > | < | <> | <= | >=
<tipo>           ::= INTEGER
                   | REAL
                   | STRING
<constante>      ::= <identificador> | <número>
<lista_identif.>  ::= <identificador> { , <identificador> }
<identificador>  ::= <letra> { <letra> | <dígito> }
<número>         ::= <dígito> { <dígito> }
<dígito>         ::= 0|1|2|3|4|5|6|7|8|9
<letra>          ::= a|b|c|d|e|f ... x|y|z|A|B|C|D ... X|Y|Z
```

OBS:
As partes marcadas em vermelho geralmente vem prontas do analisador léxico. Não precisaria aparecer aqui na GLC.

BNF x E-BNF

BNF

```
<expr> → <expr> + <term>
<expr> → <expr> - <term>
<expr> → <term>
<term> → <term> * <factor>
<term> → <term> / <factor>
<term> → <factor>
<factor> → <exp> ** <factor>
<factor> → <exp>
<exp> → ( <expr> )
<exp> → id
```

EBNF

```
<expr> → <term> { ( + | - ) <term> }

<term> → <factor> { ( * | / ) <factor> }

<factor> → <exp> [ ** <factor> ]

<exp> → ( <expr> ) | id
```

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

Problemas:

- Não há como impor restrição de distribuição no código fonte
  - Exemplo, uma variável deve ser declarada antes de ser usada
  - Descreve apenas a sintaxe, não descreve a semântica
  - Nada melhor foi proposto

Especificação da sintaxe

Gramáticas Livres de Contexto - BNF

Sugestões de consulta:

https://tomassetti.me/ebnf/

https://www.cs.cmu.edu/~pattis/misc/ebnf.pdf

https://www.cl.cam.ac.uk/~mgk25/iso-14977-paper.pdf

Dar uma olhada também nos livros disponíveis na seção de arquivos no grupo do Teams.

Derivações

- Existem várias formas de enxergar o processo pelo qual uma gramática define uma linguagem
- A visão derivacional descreve a construção *top-down* (de cima para baixo) de uma árvore gramatical (árvore de derivação)
- A ideia é que uma produção seja tratada como uma regra de reescrita, no qual o não-terminal à esquerda (antes de ‘→’) é substituído por uma cadeia do lado direito (depois de ‘→’)

Derivações

Exemplo

- Vejamos algumas derivações na gramática de expressões
- Começando por uma expressão simples:

142 + 17

- Sequência de tokens: <num, 142> <op, +> <num, 17>
- Derivação:

$$\mathbf{E \Rightarrow E + E \Rightarrow num + E \Rightarrow num + num}$$

‘⇒’ significa *deriva em*

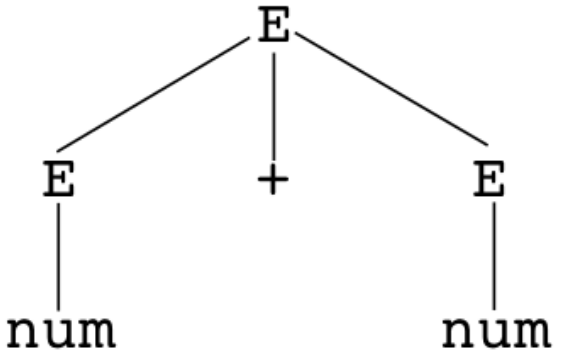
Derivações

Árvore de derivação

Exemplo

- A árvore de derivação correspondente para

$$\mathbf{E \Rightarrow E + E \Rightarrow num + E \Rightarrow num + num}$$



Derivações

Derivações mais à esquerda e mais à direita

- É importante termos formas sistemáticas de aplicações das regras de derivação
- Essa forma sistemática de aplicação das regras de uma gramática é estabelecida através das *derivações canônicas*
- Duas formas de derivação canônica são estabelecidas:
  - Derivações mais à esquerda
  - Derivações mais à direita

Derivações

Derivações mais à esquerda e mais à direita

- Na derivação mais à esquerda (*leftmost derivation*), a opção é aplicar uma regra da gramática ao símbolo não-terminal **mais à esquerda** da forma sentencial sendo analisada
  - No exemplo anterior, utilizamos uma derivação mais à esquerda
- Similarmente, na derivação mais à direita (*rightmost derivation*) o símbolo não-terminal **mais à direita** é sempre selecionado para ser substituído usando alguma regra da gramática

Análise Sintática – Conceitos preliminares

Big-Example

- Considere a gramática abaixo, para a linguagem LS (E-BNF)

```
program    ::= cmd-seq
cmd-seq    ::= cmd {';' cmd}
cmd        ::= if-cmd | repeat-cmd | assign-cmd | read-cmd | write-cmd
if-cmd     ::= IF exp THEN cmd-seq [ELSE cmd-seq] END
repeat-cmd ::= REPEAT cmd-seq UNTIL exp
assign-cmd ::= ID ':=' exp
read-cmd   ::= READ ID
write-cmd  ::= WRITE exp
exp        ::= simple-exp [rel-op simple-exp]
rel-op     ::= '<' | '='
simple-exp ::= term {add-op term}
add-op     ::= '+' | '-'
term       ::= factor {mul-op factor}
mul-op     ::= '*' | '/'
factor     ::= '(' exp ')' | NUMBER | ID
```

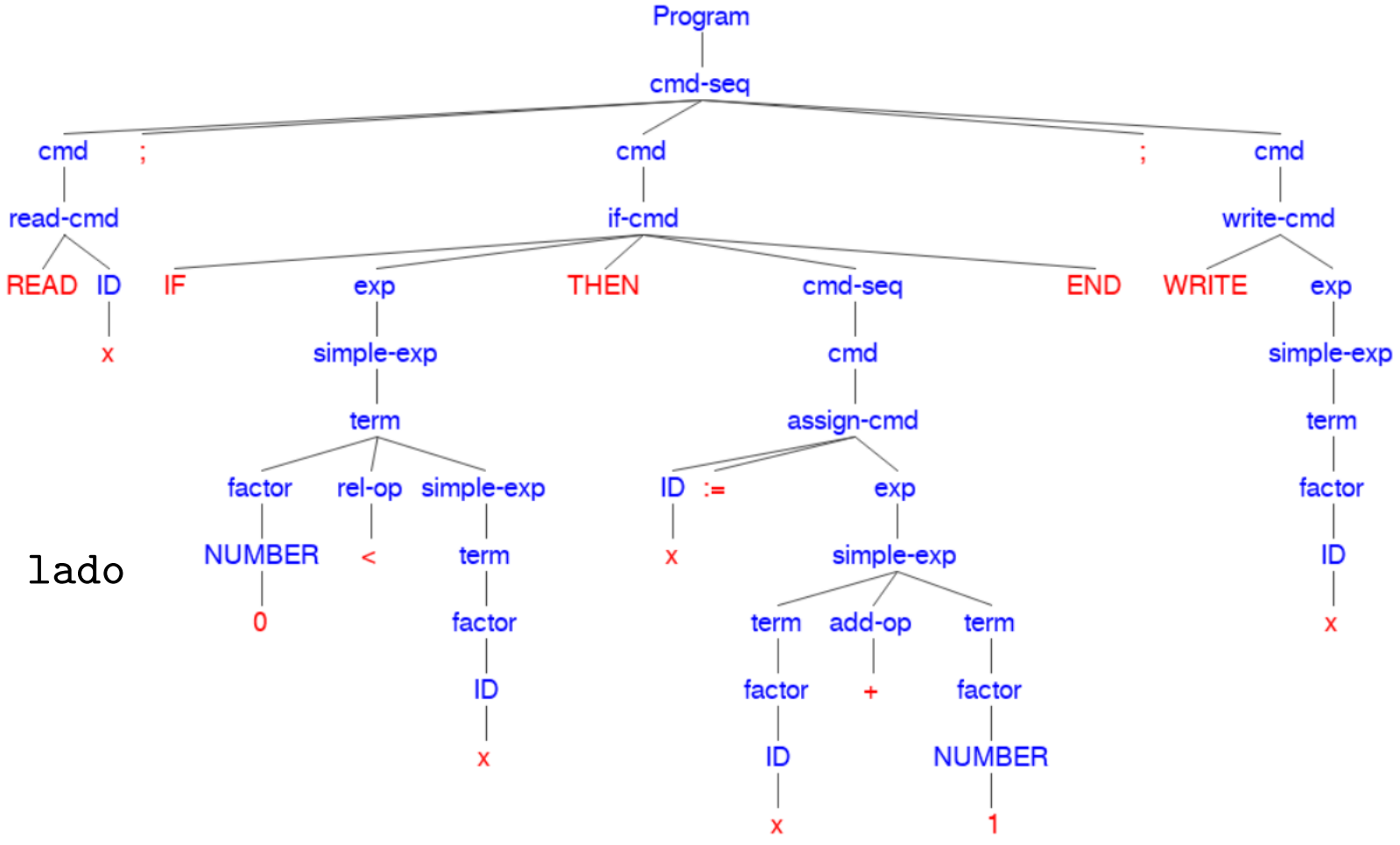
Análise Sintática – Conceitos preliminares

Big-Example

O código:

```
READ x;
IF 0 < x then
   x := x-1
END;
WRITE x
```

gera a árvore ao lado

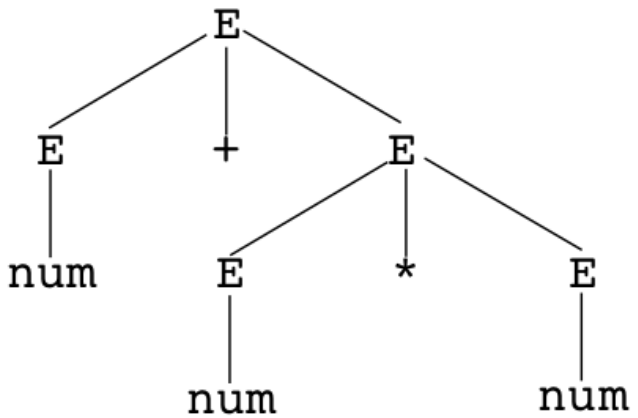


Ambiguidade

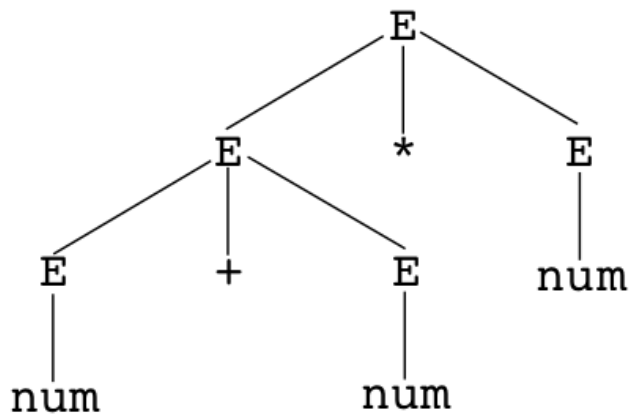
- Certas gramáticas permitem que uma mesma sentença tenha mais de uma árvore de derivação
- Isso torna a gramática inadequada para a linguagem de programação, pois o compilador não pode determinar a estrutura desse programa fonte
- Duas derivações (esquerda e direita) podem gerar uma unica árvore sintática, mas duas árvores sintáticas não podem ser geradas por uma derivação

Ambiguidade

- Exemplo:
  - Gramática de expressões
  - Na expressão 6 + 5 * 12, deve ser efetuada primeiro a soma ou a multiplicação?
  - Duas árvores de derivação podem ser construídas



derivação mais à esquerda



derivação mais à direita

Ambiguidade

- Uma ambiguidade por ser evitada de duas formas:

1) **Reescrever a gramática para remover a ambiguidade** (isso pode tornar a gramática mais complexa).

   Entretanto, dada uma gramática qualquer não existe um algoritmo que indique se a gramática é ambígua ou não; ou

2) **Definir ordens de prioridade durante a derivação** (durante o processamento de uma sentença).

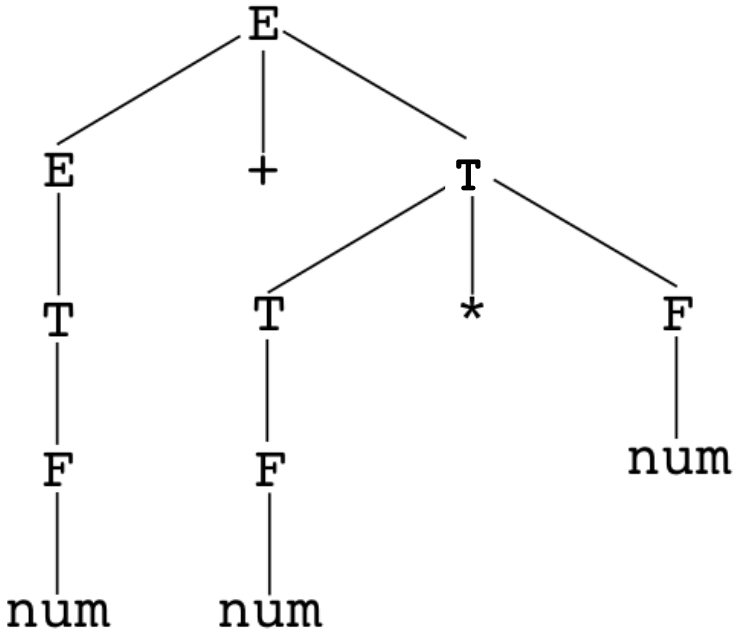
Ambiguidade

- Reescrita da gramática (Exemplo)
- Expressões:
  - multiplicações devem ser efetuadas antes das somas
  - Nova gramática

$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow (E) \mid num \mid id$$

- Árvore de derivação única →

  6 + 5 * 12



Ambiguidade

- Outro exemplo clássico em Lps
- *Problema do Else Pendente*:
  - Dada a produção em uma gramática que representa um comando condicional de uma linguagem de programação, onde a cláusula else é opcional

```
decl    → if_decl |
          outra
if_decl → if ( exp ) decl |
          if ( exp ) decl else decl
exp     → 0 |
          1
```

Ambiguidade

- Outro exemplo clássico em Lps
- *Problema do Else Pendente*:
  - Dada a produção em uma gramática que representa um comando condicional de uma linguagem de programação, onde a cláusula else é opcional
- A sentença:

if (0) if (1) outra else outra

  - poderia dar margem a duas árvores gramaticais distintas
  - A qual *if* o *else* pertence?
    - Mais comum: associa o *else* ao último *if* possível

Ambiguidade

Essa ambiguidade pode ser removida de três formas:

1) Inserir delimitadores de blocos, indicando o início e o fim de cada if
2) Reescrever a gramática
3) Usar a chamada regra do aninhamento, que implica que o else sempre será associado à declaração if mais próxima que não tiver um else associado a ele

- Forma 1 é a mais fácil quando se está na fase do projeto da linguagem
  - De qualquer forma, a gramática deverá ser reescrita
- Usar a forma 3 caso já esteja na etapa de programação do compilador

Estratégias para Análise Sintática

Duas técnicas distintas e opostas para gerar as derivações são sugeridas:

1) *Top-Down* ou Descendente

2) *Bottom-Up* ou Ascendente

Estratégias para Análise Sintática

- Os parsers **top-down** começam com a raiz e fazem a árvore de **derivação** crescer em direção às folhas
- A cada etapa, ele seleciona um nó para algum não-terminal na borda inferior da árvore
  - Mais a direita ou mais a esquerda
  - Estende como uma subárvore que representa o lado direito de uma produção que substitui o não-terminal

Estratégias para Análise Sintática

- Os parsers **bottom-up** começam com as folhas e fazem a **redução** com as folhas e fazem a árvore crescer em direção à raiz
- Em cada etapa, ele identifica uma substring contígua na borda superior da árvore de derivação, que corresponde ao lado direito de alguma produção
- Depois, constrói um nó para o lado esquerdo da regra e o conecta à árvore

Estratégias para Análise Sintática

Exemplo:

Gramática:

1. **E → E + T**
2. **E → T**
3. **T → T*F**
4. **T → F**
5. **F → (E)**
6. **F → a**

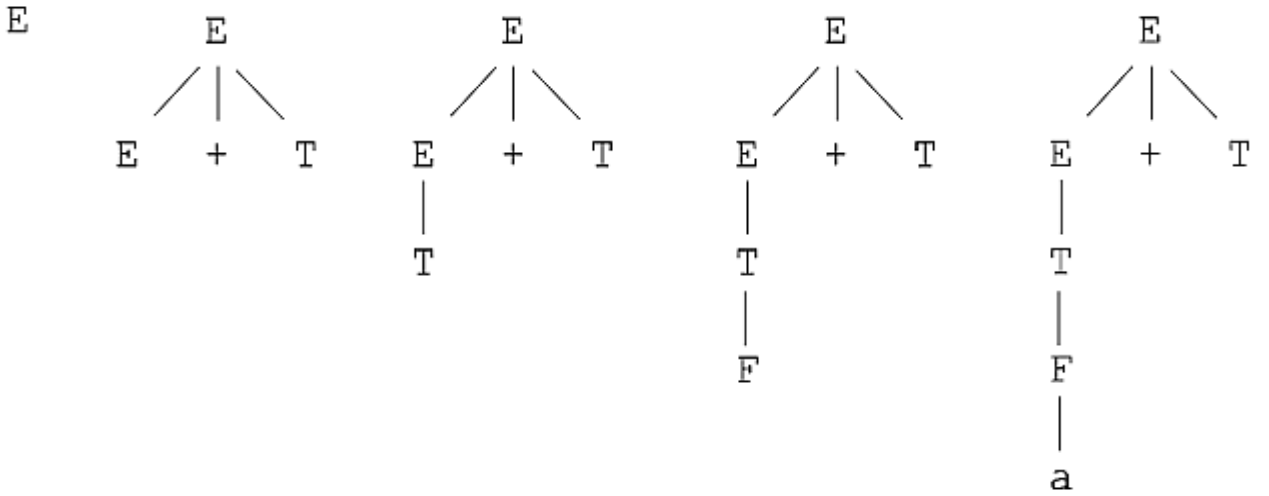
Sentença: a + a * a

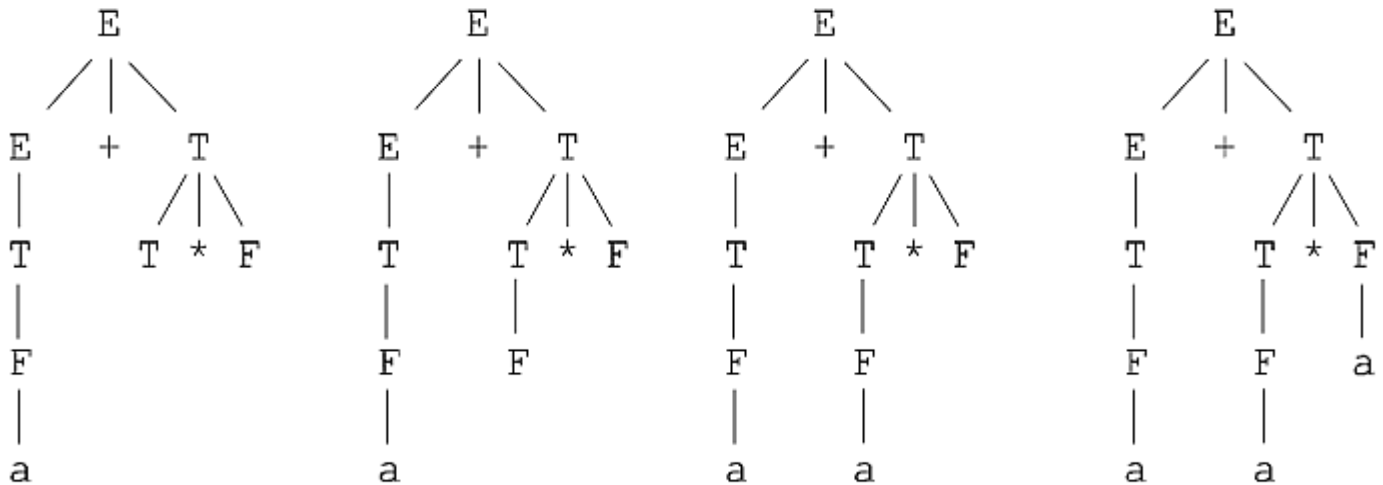
Gramática:

1. E → E + T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

Sentença: a + a * a

Top-Down:





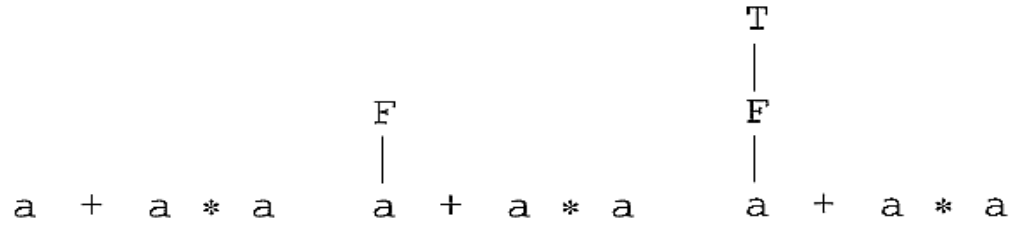
E ⇒ E+T ⇒ T+T⇒ a+T ⇒ a+T*F ⇒ a+F*F ⇒ a+a*F ⇒ a+a*a

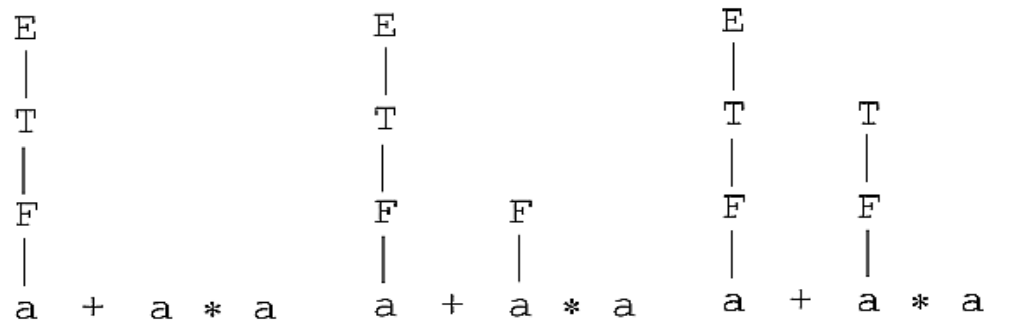
Gramática:

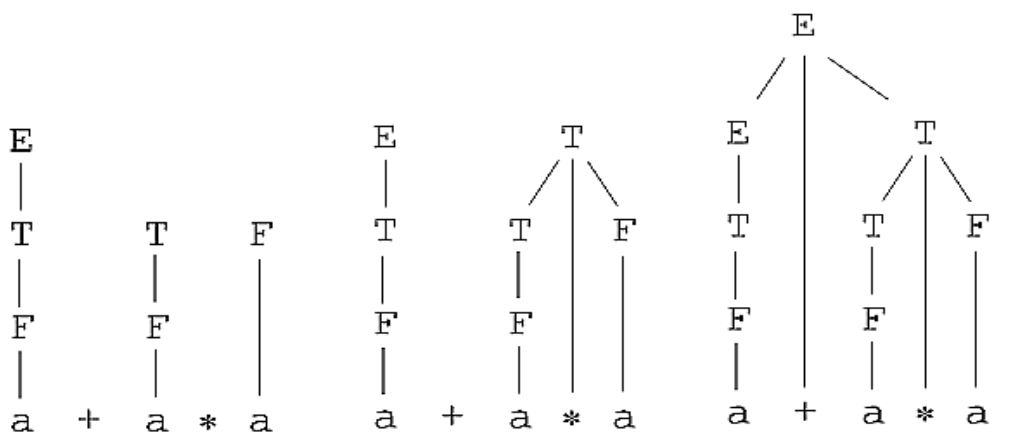
1. E → E + T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

Sentença: a + a * a

Bottom-up:







a+a*a ⇐ F+a*a ⇐ T+a*a ⇐ E+a*a ⇐ E+F*a ⇐ E+T*a

Análise Sintática:
Exercícios

- **Apostila: Capítulo 3**
  - Todos

- **Livro Dragão (1ª Ed.): Capítulo 3**
  - 4.1
  - 4.2
  - 4.3