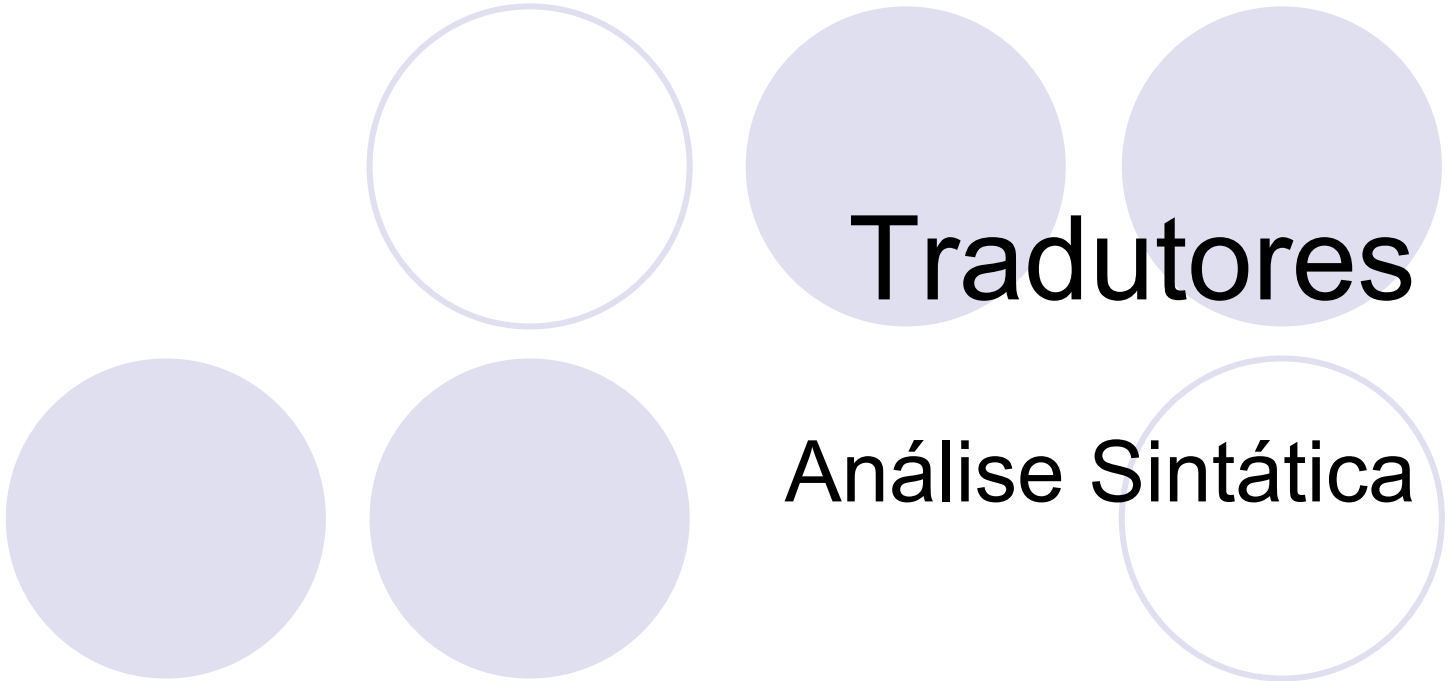


Tradutores

Análise Sintática

Sumário

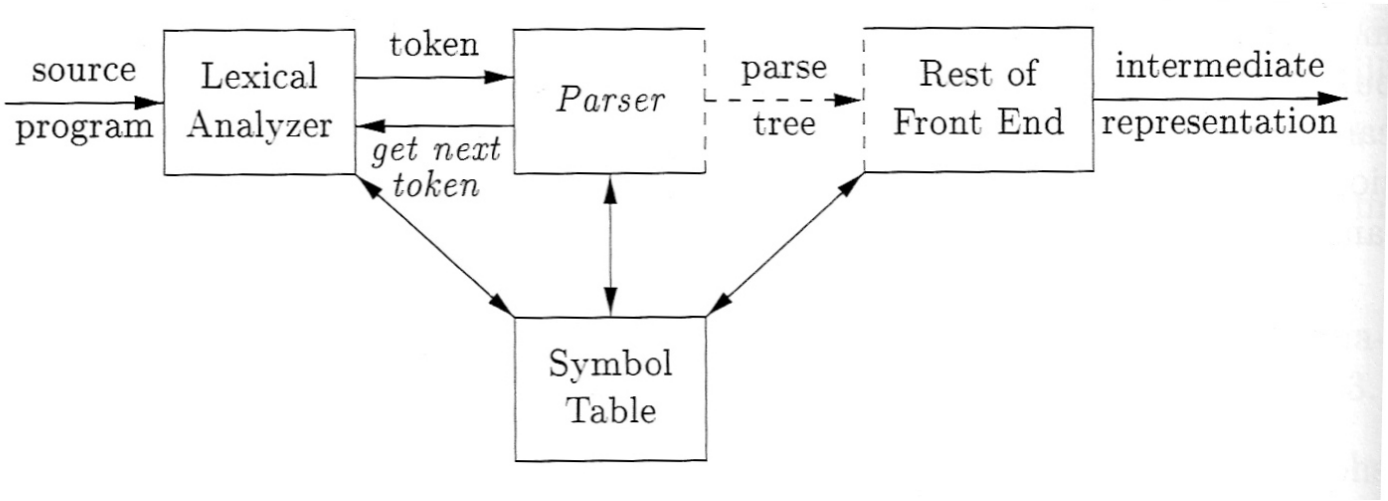
- Introdução
- O papel do analisador sintático
- Tratamento de erros
- Recuperação de erros
- Gramáticas Livres de Contexto

Análise Sintática

- A especificação de uma linguagem através de uma gramática oferece as seguintes vantagens:
  - Definição precisa e de fácil entendimento da especificação sintática da linguagem
  - Possibilidade da construção automática de analisadores sintáticos a partir da gramática
  - Estrutura imposta pela gramática facilita o processo de tradução
  - Possibilita que a linguagem seja construída de maneira incremental através da adição de novas construções

Análise Sintática

- Papel do analisador sintático
  - Construir, a partir de uma seqüência de tokens uma estrutura intermediária: árvore sintática
  - Reporte de erros sintáticos de maneira clara e precisa



Análise Sintática

- Métodos mais utilizados
  - Analisador Sintático Descendente
    - Top Down
      - Constrói a árvore sintática a partir da raiz em direção as folhas
  - Analisador Sintático Ascendente
    - Bottom Up
      - Constrói a árvore sintática a partir das folhas em direção a raiz

Análise Sintática

- Na prática diferentes tarefas podem ser realizadas pelo analisador sintático:
  - Inserção de símbolos na tabela de símbolos
  - Verificação de tipos
  - Geração de código
    - Tradução dirigida por sintaxe

Análise Sintática

- Tratamento de erros sintáticos
  - Analisador sintático deve saber lidar com erros
  - Diferentes níveis de erros:
    - Erros léxicos: indentificadores, palavras reservadas ou operadores mal escritos
    - Erros sintáticos: falta de “:” ou “;”. Em linguagens como C ou Java – uso de um comando case for a de um switch
    - Erros semânticos: incompatibilidade de tipos entre operadores. Em Java um método void retornando um valor
    - Erros lógicos: tipo de erro mais difícil de ser detectado. Ex.: utilização de “=“ no lugar de “==“. Programa é considerado “bem formado”

Análise Sintática

- O tratamento de erros em um analisador sintático possui objetivos simples, embora difíceis de serem implementados:
  - Reportar a presença de erros de maneira clara e precisa
    - Lugar onde o erro ocorreu (mínimo esperado)
  - Recuperar-se de cada erro encontrado de modo a encontrar erros subsequentes
  - Adicionar o mínimo possível de *overhead* ao processo

Análise Sintática

- Estratégias de recuperação de erros
  - Uma vez encontrado o erro, diferentes métodos podem ser utilizados de modo a tratá-lo.
  - Métodos possíveis
    - Parar o processo de análise ao encontrar um erro
    - Retornar a um estado consistente e continuar a análise
    - Estipular um limíte máximo de erros. Ao extrapolar este limite a análise para (cascateamento de erros)

Análise Sintática

- Estratégias mais utilizadas:
  - *Panic Mode*
  - *Phrase Level*
  - *Error-productions*

Análise Sintática

- *Panic Mode*
  - Ao detectar um erro o analisador sintático descarta tokens de sua entrada, um a um, até que um token de sincronismo seja encontrado.
  - Tokens de sincronismo: tipicamente delimitadores: “;” ou “}”
  - Tokens delimitadores devem ter um papel claro e não ambíguo na especificação da linguagem
  - Vantagem: simplicidade, não gera laços infinitos
  - Desvantagem: grande quantidade da entrada pode ser descartada

Análise Sintática

- *Phrase Level* – Baseado em expressões
  - Ao detectar o erro o analisador executa uma correção local no restante da entrada.
  - Um prefixo do restante da entrada é substituído por uma seqüência de símbolos que possibilite ao analisador continuar a análise
  - Exemplos de substituições: substituição de um “.” por um “;”
  - Dificuldade em tratar erros que tenham ocorrido em algum ponto anterior ao ponto de detecção

Análise Sintática

- *Error Productions* – Produção de erros
  - Tenta antecipar erros possíveis
  - Gramática da linguagem é “aumentada” de modo a conter produções de erros
  - Erro é detectado no momento em que a produção é utilizada durante a análise sintática
  - Associado a esta produção pode existir uma ação semântica que trata o erro

Análise Sintática

- Gramáticas livres de contexto
  - Conjunto de símbolos não terminais
  - Conjunto de símbolos terminais
  - Símbolo inicial (não terminal)
  - Produções

$G = (V, \Sigma, R, S)$ where

1. V is a finite set of *non-terminal* characters or variables. They represent different types of phrase or clause in the sentence.
2. $\sum$ is a finite set of *terminals*, disjoint with V, which make up the actual content of the sentence.
3. S is the start variable, used to represent the whole sentence (or program). It must be an element of V.
4. R is a relation from V to $(V \cup \Sigma)^*$ such that $\exists\, w \in (V \cup \Sigma)^* : (S, w) \in R$.

The members of R are called the *rules* or *productions* of the grammar.

Análise Sintática

- Gramáticas Livres de Contexto
- Terminais
  - Símbolos básicos a partir dos quais as sentenças são formadas. Neste sentido, do ponto de vista do analisador sintático, um token é um terminal. A partir disso assume-se que o analisador léxico envia ao analisador sintático uma seqüência de símbolos terminais.

Análise Sintática

- Gramáticas Livres de Contexto
- Não terminais
  - Variáveis sintáticas que denotam um conjunto de sentenças. O conjunto de sentenças definido por um não terminal ajuda a definir a linguagem gerada por uma gramática. Símbolos não terminais impõem uma estrutura hierárquica na linguagem que é chave para a analise sintática

Análise Sintática

- Gramáticas Livres de Contexto
- Símbolo Inicial
  - O conjunto de sentenças gerado por este símbolo inicial é a linguagem definida pela gramática. Convencionalmente o símbolo inicial é listado primeiro na definição da gramática

Análise Sintática

- Gramáticas Livres de Contexto
- Produções
  - Especificam a maneira como símbolos terminais e não terminais podem ser combinados para formar sentenças. Cada produção consiste de:
    - Um não terminal – também referenciado como lado esquerdo da produção
    - O símbolo →. As vezes também é utilizado o símbolo ::=
    - O corpo – também referenciado como lado direito da produção. Consiste de zero ou mais símbolos terminais e não terminais. O corpo da produção define como sentenças do lado esquerdo da produção são construídas.

Análise Sintática

- Gramáticas Livres de Contexto
  - Gramática para expressões aritméticas

expression → *expression* + *term*
expression → *expression* – *term*
expression → *term*
term → *term* * *factor*
term → *term* / *factor*
term → *factor*
factor → *(expression)*
factor → id

Terminais: + - / * ( ) e id

Não terminais: *expression term factor*

Símbolo inicial: *expression*

Análise Sintática

- Gramáticas livres de contexto
  - Derivações
    - Utilizadas na construção da árvore sintática
    - Produções da gramática são vistas como regras de reescrita
    - Iniciando-se com o símbolo inicial da gramática, cada passo de reescrita substitui um não terminal pelo corpo de uma de suas produções.

Análise Sintática

- Derivações

Seja a seguinte gramática:

$$E \rightarrow E + E \mid E * E \mid -E \mid (E) \mid id$$

A produção E → -E implica que se E denota uma expressão, então –E também denota uma expressão. A substituição de um único não terminal E por –E pode ser descrita da seguinte maneira:
E => -E
Lê-se E deriva –E. A produção E → (E) pode ser aplicada para substituir qualquer instancia de E pertencente a qualquer sentença de símbolos da gramática por (E). Ex.: E => (E) * E ou E => E * (E). A partir de um único não terminal E, pode-se repetidamente aplicar produções em qualquer ordem de modo a obter uma seqüência de substituições. Por exemplo:
E => -E => -(E) => -(id)
Esta seqüência de substituições é chamada de "derivação de –(id) a partir de E" Esta derivação é uma prova de que a seqüência –(id) é uma instância particular de uma expressão.

Análise Sintática

Gramáticas Livres de Contexto

O símbolo => significa "deriva em um passo". Quando uma seqüência de derivações *a1* => *a2* => *an* reescreve *a1* em *an* diz-se que *a1* deriva *an*. O símbolo =>* pode ser usado para indicar a derivação em zero ou mais passos. Neste sentido tem-se que:

a =>**a* para qualquer seqüência *a*

Se *a* =>* *B* e *B* => *y*, então *a*=>**y*

Da mesma maneira tem-se que =>+ significa "deriva em um ou mais passos"

Se *S* =>* *a*, onde *S* é o símbolo inicial de uma gramática *G*, diz-se que *a* é uma forma sentencial de *G*. Uma forma sentencial pode conter tanto terminais quanto não terminais e também pode ser vazia. Uma sentença de *G* é uma forma sentencial que não contenha nenhum símbolo não terminal.

A linguagem gerada por uma gramática é um conjunto de sentenças. Neste sentido uma seqüência de terminais *w* está em *L(G)*, a linguagem gerada pela gramática *G*, se e somente se *w* é uma sentença de *G*. Em outras palavras *S* =>* *w*. A linguagem gerada por uma gramática é dita uma linguagem livre de contexto. Se duas gramáticas geram a mesma linguagem elas são equivalentes.

A sentença –(id+id) é uma sentença da gramática citada acima pois existe uma derivação:

E => -E => -(E) => -(E+E) => -(id + E) => -(id+id)

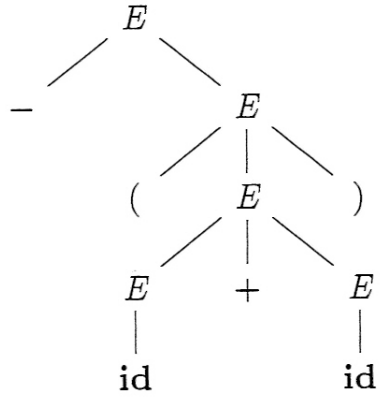
As seqüências E, -E, -(E),...., -(id+id) são todas formas sentencias de *G*. Desta maneira podemos escrever que E =>* -(id+id).

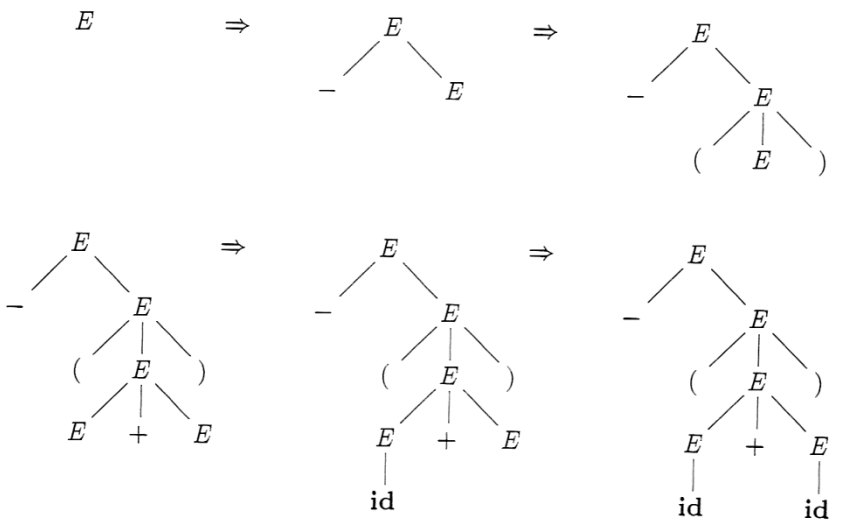
Análise Sintática

- Derivações
  - Derivação mais à esquerda
    - O não terminal mais à esquerda de cada sentença é sempre escolhido para substituição.
  - Derivação mais à direita
    - O não terminal mais à direita de cada sentença é sempre escolhido para substituição.

Análise Sintática

- Árvores sintáticas e derivações
  - Uma árvore sintática é uma representação gráfica de uma derivação que elimina a ordem na qual as produções são aplicadas





Análise Sintática

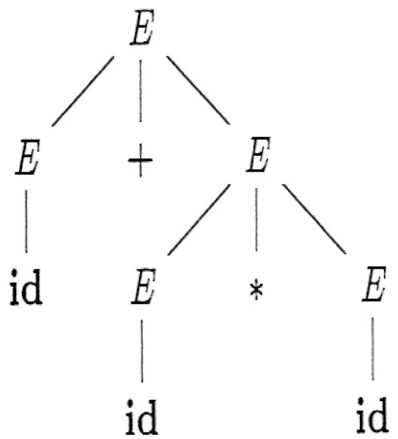
- **Ambigüidade**
  - Uma gramática que produz mais de uma árvore sintática para a mesma sentença é dita ambígua. Em outras palavras, uma gramática é ambígua se existe mais de uma derivação mais à direita ou mais de uma derivação mais à esquerda para a mesma sentença.

Análise Sintática

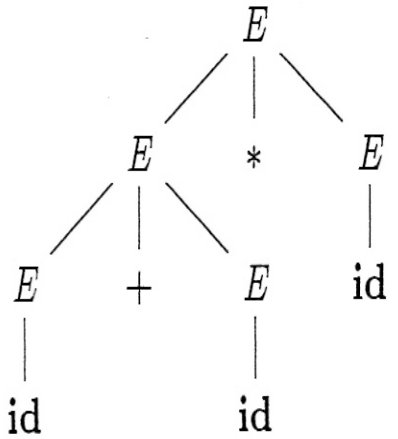
- **Ambigüidade**

E → E + E | E * E | -E | (E) | id

$$
\begin{aligned}
&\Rightarrow E+E \\
&\Rightarrow \mathrm{id}+E \\
&\Rightarrow \mathrm{id}+E*E \\
&\Rightarrow \mathrm{id}+\mathrm{id}*E \\
&\Rightarrow \mathrm{id}+\mathrm{id}*\mathrm{id}
\end{aligned}
\qquad
\begin{aligned}
E &\Rightarrow E*E \\
&\Rightarrow E+E*E \\
&\Rightarrow \mathrm{id}+E*E \\
&\Rightarrow \mathrm{id}+\mathrm{id}*E \\
&\Rightarrow \mathrm{id}+\mathrm{id}*\mathrm{id}
\end{aligned}
$$



(a)



(b)

Análise Sintática

- Precedência de operadores em uma gramática
  - Considere a expressão 9+5*2. Levando-se em conta apenas as regras de associatividade, existem duas maneiras desta expressão ser avaliada: (9+5)*2 ou 9+(5*2)
  - Informação de precedência deve ser acrescentada na gramática

Análise Sintática

- Precedência de operadores em uma gramática
  - Tabela é construída onde operadores com a mesma precedência e associatividade são colocados na mesma linha. Operadores com mesma associatividade, porém com precedência maior são colocados em linhas subseqüentes. Desta maneira a tabela é organizada em ordem crescente de precedência. A tabela abaixo mostra como ficaria uma tabela para as operações aritméticas de *, /, + e -.

Associatividade	Operadores
Associativo à esquerda	+ -
Associativo à esquerda	* /

Análise Sintática

- Precedência de operares em uma gramática
  - Um não terminal é criado para cada nível de precedência (linha da tabela)
  - Um terminal é criado para unidades básicas
  - Ex: gramática de expressões aritméticas
    - Um não terminal para dígitos e expressões entre parênteses (alteram a precedência)

`fator → digito | (expr)`

Análise Sintática

- Exemplo (cont.)
  - Para os operadores * e / cria-se um não terminal com as seguintes produções:

```
termo → termo * fator
termo → termo / fator
termo → fator
```

  - Para os operadores + e – cria-se um não terminal com as seguintes produções:

```
expr → expr + termo
expr → expr - termo
expr → termo
```

Análise Sintática

- Exemplo (cont.)
  - Tem-se então a seguinte gramática resultante

$$
\begin{aligned}
\text{expr} &\rightarrow \text{expr} + \text{termo} \mid \text{expr} - \text{termo} \mid \text{termo} \\
\text{termo} &\rightarrow \text{termo} * \text{fator} \mid \text{termo} \ / \ \text{fator} \mid \text{fator} \\
\text{fator} &\rightarrow \text{digito} \mid (\text{expr})
\end{aligned}
$$

•Nesta gramática uma expressão é uma lista de termos separada por operadores + ou –, e um termo é uma lista de fatores separados por * ou /. Uma expressão entre parênteses é considerada um fator.

•Esta técnica pode ser generalizada para n níveis deprecedência

Análise Sintática

- Exercício
  - Considerar a seguinte gramática livre de contexto:
    S → SS+ | SS* | a
    E a seguinte sentença: **aa+a***
  - Dê a derivação mais a esquerda para esta sentença
  - Dê a derivação mais a direita para esta sentença
  - Desenhe a árvore sintática para esta sentença.

Análise Léxica x Análise Sintática

- Porque utilizar expressões regulares para a definição léxica de uma linguagem?
  - Modularização - a estrutura sintática de uma linguagem, quando separada em partes léxicas e não léxicas facilita a implementação de um *front-end* mais modular.
  - As regras léxicas são normalmente mais simples que as regras sintáticas. Neste sentido não é necessário à utilização de uma notação tão poderosa como uma gramática para descrevê-las.
  - Expressões regulares normalmente provêem uma notação mais concisa e de fácil entendimento para tokens do que uma gramática.
  - Analisadores léxicos mais eficientes podem ser construídos a partir de expressões regulares.

Análise Léxica x Análise Sintática

- Não existem regras claras do que deve ser atribuído a regras léxicas ou a regras sintáticas.
- Expressões regulares : melhores para descrever a estrutura de construções como identificadores, constantes, palavras reservadas e espaços em branco.
- Gramáticas: melhores para descrição de estruturas aninhadas, como por exemplo, parênteses balanceados, construções do tipo “begin” “end”, “if then else”. Estas estruturas não podem ser descritas por expressões regulares.

Eliminando a ambiguidade

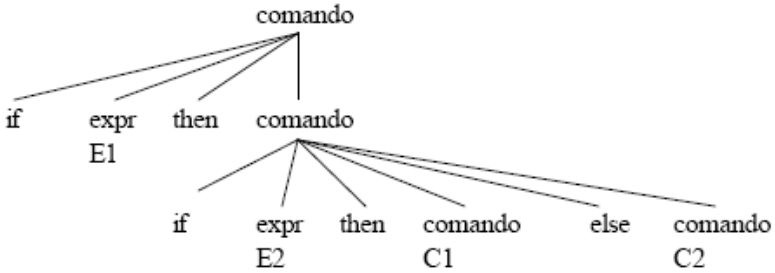
- Existem casos onde uma gramática ambígua pode ser reescrita de modo a eliminar a ambigüidade

comando → *if expr then comando*
| *if expr then comando else comando*
| *outro*

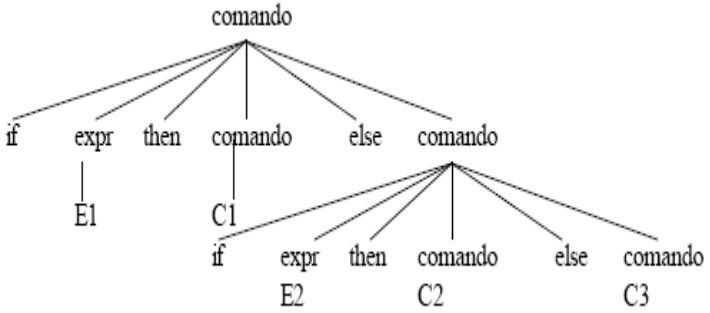
Eliminando a ambiguidade

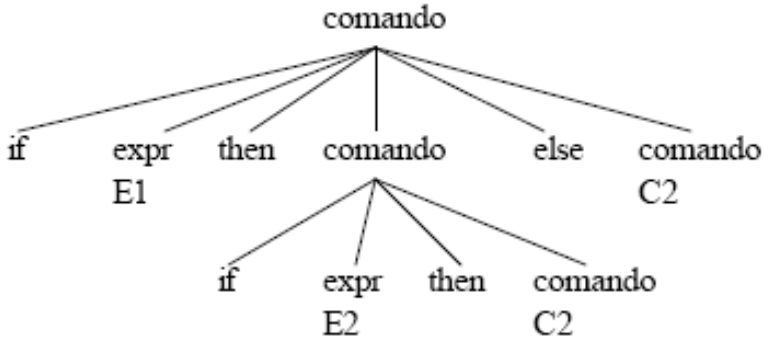
comando → *if expr then comando*
| *if expr then comando else comando*
| *outro*

if E1 then if E2 then C1 else C2



if E1 then C1 else if E2 then C2 else C3





Eliminando a ambiguidade

comando → *comando_não_marcado*
| *comando_marcado*

comando_marcado → *if expr then comando_marcado else comando_marcado*
| *outro*

comando_não_marcado → *if expr then comando*
| *if expr then comando_marcado else comando_não_marcado*

- Nesta gramática modificada um comando que aparece entre um “then” e um “else” deve ser marcado, isto é, ele não pode terminar com um “then” não associado a um “if” ou “then” aberto. Um comando marcado é um comando if-then-else que não contenha nenhuma abertura de comando condicional. Ele pode conter qualquer outro comando não condicional.

Eliminando a recursão à esquerda

- Uma gramática é considerada recursiva à esquerda quando possui produções com o seguinte formato geral: $A \rightarrow Aa$

A recursão a esquerda pode ser eliminada com a utilização da seguinte técnica: inicialmente agrupam-se as produções da seguinte maneira:

$$A \rightarrow Aa_1 \mid Aa_2 \mid \ldots\ldots \mid Aa_m \mid B_1 \mid B_2 \mid \ldots\ldots \mid B_n$$

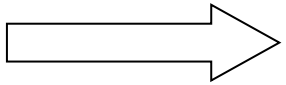
Onde nenhuma produção B_i inicie com A.
Após isto estas produções são substituídas por:

$$A \rightarrow B_1A' \mid B_2A' \mid \ldots\ldots \mid B_nA'$$
$$A' \rightarrow a_1A' \mid a_2A' \mid \ldots\ldots \mid a_mA'$$

Este procedimento elimina a recursão à esquerda dos não terminais A e A' (dado que nenhum a_i seja ε. Este procedimento, entretanto, não elimina uma recursão indireta à esquerda.

Eliminando a recursão à esquerda

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



$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \varepsilon$$
$$F \rightarrow (E) \mid id$$

Fatoração à esquerda

- Uma gramática fatorada a esquerda deve ser utilizada quando se deseja utilizar um analisador preditivo.

comando → *if expr then comando else comando*
| *if expr then comando*

- Nesta regra o terminal if inicia ambas as produções, podendo ocasionar uma dúvida a respeito de qual deve ser aplicada.

Fatoração à esquerda

Algoritmo para fatoração a esquerda de uma gramática:

ENTRADA: Uma gramática G
SAÍDA: Uma gramática equivalente fatorada à esquerda
MÉTODO: Para cada não terminal A, encontra-se o maior prefixo *a* comum a duas ou mais produções. Se *a* é diferente de ε (existe um prefixo comum), substituir todas as produções $A \rightarrow aB_1 \mid aB_2 \mid .... \mid aB_n \mid y$ por:

$$A \rightarrow aA' \mid y$$
$$A' \rightarrow B_1 \mid B_2 \mid .... \mid B_n$$

Fatoração à esquerda

- Exercício
  - Fatorar a esquerda a seguinte gramática:

comando → *if expr then comando*
| *if expr then comando else comando*
| *outro*

Fatoração à esquerda

- Solução

comando → if *expr* then *comando resto*
resto → else *comando* | ε

Exercício

- Considere a seguinte gramática para expressões boleanas.
  - E → E **or** E | E **and** E | **not** E | (E) | id
  - Mostre que esta gramática é ambigua mostrando duas derivações diferentes para a seguinte expressão:

    id **and** (**not** id **or** id)

Exercício

- Costrua uma gramática não ambígua que descreva a mesma linguagem do exercício anterior. Esta gramática deve também lidar com a precedência dos operadores boleanos.
  - **not** possui a maior precedência, seguido pelo operador **and** que é seguido pelo operador **or**.

Exercício

Ao fatorar uma gramática à esquerda podem haver diferentes prefixos comuns em diferentes produções para o mesmo não terminal. Para estes casos o algoritmo visto em aula escolhe o maior prefixo comum para iniciar o processo. Em seguida, caso ainda existam prefixos comuns, o segundo maior prefixo é escolhido. Este processo é repetido até que não existam mais prefixos comuns.
Considerando a seguinte gramática:

S → aSSbs
S → aSaSb
S → abb
S → b

Mostre os passos envolvidos na fatoração para cada um dos prefixos comuns.