Capítulo 2

Definição de Linguagens

2.1 Linguagens Formais

Definição 2.1 (Alfabeto) *Um conjunto finito e não vazio de símbolos arbitrários é designado por um* alfabeto*, e é denotado por V.*

Os elementos de V são designados por *letras*, ou *símbolos*. Exemplos de alfabetos: os alfabetos das línguas naturais, os alfabetos ASCII, EBCDIC, e UNICODE.

Definição 2.2 (Palavra) *As sequências finitas de letras são designadas por palavras sobre o alfabeto V.*

O conjunto de todas as palavras sobre V é denotado por V^*.

A palavra vazia, não contém nenhuma letra, e é denotada por ϵ. Considera-se que ϵ pertence a V^*.

O comprimento de uma palavra v, denotado por $|v|$ é dado pelo número de símbolos que contém, por exemplo $|\epsilon| = 0$, $|abcd| = 4$.

Definição 2.3 (Concatenação de Palavras) *A concatenação de duas palavras P e Q, denotado por PQ, é a palavra formada por justaposição dos símbolos de P com os símbolos de Q.*

A concatenação de palavras é: associativa, não é (em geral) comutativa, tem elemento neutro que é a palavra vazia. O conjunto V^* é fechado para a concatenação de palavras.

Duas palavras são iguais se uma é cópia exacta, letra por letra, da outra.

A palavra P é parte da palavra Q se existirem palavras P_1 e P_2 tais que $Q = P_1PP_2$. Se $P_1 \neq \epsilon \vee P_2 \neq \epsilon$, então P é uma *parte própria* de Q. Se $P_1 = \epsilon$, então P é um *prefixo* de Q. Se $P_2 = \epsilon$ então P é um sufixo de Q.

[1](Versão 1.14)

Dado um número natural n e dado um palavra P, então P^n denota a concatenação de P com ela própria n vezes. Por convenção $P^0 = \epsilon$.

A imagem inversa de P, denotada por P^{-1} é a palavra que se obtém de P por inversão da ordem das letras. Por exemplo: se $P = abcd$, então $P^{-1} = dcba$.

Temos que:

$$\begin{aligned} (P^{-1})^{-1} &= P \\ (P^{-1})^i &= (P^i)^{-1} \end{aligned}$$

Definição 2.4 (Linguagem) *Um conjunto arbitrário de palavras de V^* é designado por uma linguagem e é usualmente denotado por L.*

A linguagem vazia $\emptyset$ não contém nenhuma palavra. O conjunto $V^* \backslash \{\epsilon\}$ é denotado por V^+.

Uma linguagem $L \subset V^*$ é finita se contém um número finito de palavras. A linguagem (completa) V^* é sempre infinita enumerável. O conjunto de todas as linguagens sobre um dado alfabeto (finito) é infinito não enumerável.

Para poder definir uma dada linguagem é necessário encontrar um processo finito (e fácil) de caracterizar uma linguagem, uma forma de o fazer recorre à definição de conjuntos e utiliza as operações acima descritas como forma de caracterizar a construção de palavras.

Por exemplo, se $V = \{a, b\}$, então:

$$\begin{aligned} L_1 &= \{a, b, \epsilon\} \\ L_2 &= \{a^i b^i \mid i = 0, 1, \ldots\} \\ L_3 &= \{PP^{-1} \mid P \in V^*\} \\ L_4 &= \{a^{n^2} \mid n = 1, 2, \ldots\} \end{aligned}$$

são tudo definições possíveis de linguagens sobre V. A primeira das definições foi feita por extensão, as seguintes de forma predicativa.

Torna-se claro que é necessário uma forma mais poderosa de definir uma linguagem.

2.1.1 Operações com Linguagens

Dado que se definiu as linguagens como sendo conjuntos de palavras, podemos então definir sobre as linguagens as habituais operações com conjuntos.

$$\begin{aligned} L_1 \cup L_2 &= \{P \mid P \in L_1 \text{ ou } P \in L_2\} \\ L_1 \cap L_2 &= \{P \mid P \in L_1 \text{ e } P \in L_2\} \\ L_1 \backslash L_2 &= \{P \mid P \in L_1 \text{ e } P \notin L_2\} \end{aligned}$$

O complementar de uma linguagem é dado em relação à linguagem completa V^*.

$$\overline{L} = V^* \backslash L$$

Pode-se também estender a concatenação de palavras às linguagens de forma a definir a concatenação de duas linguagens.

$$L_1 L_2 = \{P_1 P_2 \mid P_1 \in L_1 \text{ ou } P_2 \in L_2\}$$

Podemos então definir a iteração de uma linguagem L, L^i, para um qualquer i inteiro positivo. Por convenção $L^0 = \{\epsilon\}$. Temos que $\emptyset L = L\emptyset = \emptyset$ e $\{\epsilon\}L = L\{\epsilon\} = L$ para qualquer linguagem L.

O fecho da iteração (fecho de Kleene), denotado por L^*, é definido como:

$$L^* = \bigcup_{i \geq 0} L^i$$

esta notação está de acordo com a notação V^* se se considerar o alfabeto V como uma linguagem contendo palavras só com uma letra. Se se definir:

$$L^+ = \bigcup_{i \geq 1} L$$

temos as relações $L^+ = L^*$ se $\epsilon \in L$, e $L^+ = L^* \backslash \{\epsilon\}$ se $\epsilon \notin L$.

A linguagem espelho pode-se definir como:

$$L^{-1} = \left\{P \mid P^{-1} \in L\right\}$$

para a qual se verifica $(L^{-1})^{-1} = L$ e $(L^{-1})^i = (L^i)^{-1}$, para $i = 0, 1, \ldots$.

As operações de união, concatenação, e fecho da iteração são designadas por operações regulares, tendo um conjunto de propriedade importantes. Por exemplo pode-se afirmar que as linguagens do tipo (estabeleceremos mais à frente uma classificação para as linguagens) L_i com $i = 0, 1, 2, 3$ são fechadas para as operações regulares (Révész(1986)).

2.2 Linguagens e Expressões Regulares

As linguagens regulares vão constituir o tipo de linguagem mais simples, o seu interesse deriva da possibilidade do seu tratamento computacional.

Definição 2.5 (Linguagem Regular) *Uma linguagem diz-se regular se puder ser obtida aplicando um número finito de operações regulares sobre as seguintes linguagens:*

- *linguagem vazia;*

- *linguagem cuja única palavra é a palavra vazia;*
- *linguagem com uma única palavra.*

2.2.1 Expressões Regulares

As expressões regulares permitem representar, de modo sucinto, as linguagens regulares. Uma das vantagens das expressões regulares reside na possibilidade da sua utilização como linguagem de especificação de analisadores léxicos.

Proposição 2.1 *Toda a expressão regular denota uma linguagem regular e, conversamente, toda a linguagem regular é denotada por uma expressão regular.*

Podemos então usar as expressões regulares como forma de definir/manipular as linguagens regulares.

Definição 2.6 (Expressões Regulares) *Uma expressão regular sobre um alfabeto finito V é definida da modo indutivo como se segue:*

- *ϵ é uma expressão regular;*
- *a é uma expressão regular, para todo o $a \in V$;*
- *se R é uma expressão regular sobre V, então também o é $(R)^*$ (iteração);*
- *se Q e R são expressões regulares sobre V, então também $(Q)(R)$ e $(Q)|(R)$ (concatenação e união respectivamente) o são.*

Podemos ainda, para efeito de simplificação de escrita, acrescentar o seguinte:

- $\emptyset$ é uma expressão regular (denotando a linguagem vazia);
- se R é uma expressão regular sobre V, então também o é $(R)^+$;
- $[a, b, c] \doteq a|b|c$ e $[a - z] \doteq a|b|\ldots|z$.

Duas expressões regulares podem expressar a mesma linguagem, como tal é importante estabelecer quando é que isso acontece e, quais as propriedades que gozam as expressões regulares.

Definição 2.7 (Equivalência de Expressões Regulares) *Duas expressões regulares α e β dizem-se equivalentes, $\alpha \equiv \beta$ sse definem a mesma linguagem, isto é, $L_\alpha = L_\beta$.*

Podemos também estabelecer a equivalência das expressões regulares tendo em conta as propriedade de que elas gozam:

- ϵ é o elemento neutro da concatenação;
- a concatenação é associativa;
- $\emptyset$ é o elemento neutro da união;
- a união é associativa, comutativa, e idempotente;
- a concatenação é distributiva (à esquerda e à direita) em relação à união;
- $\alpha^+ = \alpha\alpha^* = \alpha^*\alpha$; $\alpha^* = \epsilon \,|\, \alpha^+$; $\alpha^* = (\alpha \,|\, \epsilon)^+ = (\epsilon \,|\, \alpha)^+$

Exemplo 2.1 *Os identificadores em* C *podem ser definidos por:*

```
letra -> A|B|...|Z|a|b|...|z
digito -> 0|1|...|9
identificador -> letra(letra|digito)*
```

em que os nomes são introduzidos como forma de simplificar a escrita da expressão regular final.

2.2.2 Expressões Regulares no UNIX

As expressões regulares podem (são) usadas em muitos dos comandos disponíveis nos sistemas UNIX, por exemplo o comando `grep` é um desses comandos.

Tomando como referência a página do manual do `grep` ("`man` page") temos que esse comando aceita o seguinte conjunto de expressões regulares (que é equivalente ao definido anteriormente).

`[a]`	associa o símbolo a, pode-se escrever simplesmente `a`
`[^a]`	associa com qualquer símbolo que não a
`[0-9]`	sequência de símbolos $0, 1, 2, \ldots, 9$
`[[:digit:]]`	`[:digit:]` é uma expressão regular que sobre o alfabeto ASCII é equivalente à expressão regular `[0-9]`, tem no entanto sobre esta a vantagem de ser independente do alfabeto computacional usado. O UNIX tem definidas as seguintes classes `[:alnum:]`, `[:alpha:]`, `[:cntrl:]`, `[:digit:]`, `[:graph:]`, `[:lower:]`, `[:print:]`, `[:punct:]`, `[:space:]`, `[:upper:]`, e `[:xdigit:]`
`[.]`	associa com um só símbolo (qualquer)

Uma expressão regular pode ser seguida de um de vários operadores de repetição.

`?`	opcional e associa com no máximo um símbolo
`*`	associa zero ou mais vezes
`+`	associa uma ou mais vezes
`{n}`	associa n vezes
`{n,}`	associa n ou mais vezes
`{,m}`	associa no máximo m vezes, e é opcional
`{n,m}`	associa entre n e m vezes

Os símbolos `^` e `$` são meta-símbolos que associam à palavra vazia respectivamente no princípio e no fim de uma linha. Os símbolos `\>` e `\<` associam à palavra vazia respectivamente no princípio e no fim de uma palavra.

Podemos ainda fazer a concatenação de expressões regulares `[a][b]`, assim como a união `[a]|[b]`.

A repetição tem precedência sobre a concatenação, a qual tem precedência sobre a união. Pode-se usar parêntisis para nos sobrepormos a estas regras.

Para se usarem os símbolos `?`, `+`, `{`, `|`, `(`, `)` é necessário usar as versões com *símbolo de escape* "\".

As expressões regulares (e consequentemente as linguagens regulares) são no entanto insuficientes para poder expressar completamente a maior parte das linguagens de programação, por exemplo as expressões regulares são incapazes de denotar uma linguagem com um número igual de parêntesis a abrir e a fechar.

2.3 Gramáticas

Torna-se claro que é necessário uma forma mais poderosa de definir uma linguagem.

Definição 2.8 (Gramática Generativa) *Uma gramática generativa G é um quádruplo (V_N, V_T, I, P), com V_N e V_T alfabetos finitos tais que $V_N \cap V_T = \emptyset$, I é um símbolo único em V_N, e P é um conjunto finito de pares ordenados (A, B) tais que A e B estão em $(V_N \cup V_T)^*$ e A contém pelo menos um símbolo de V_N.*

Os símbolos de V_N são designados por *símbolos não-terminais*, ou variáveis e serão denotados por letras minúsculas.

Os símbolos de V_T são designados por *terminais*, e serão denotados por letras maiúsculas.

O símbolo I é designado por *símbolo inicial* e é usado como ponto de partida da formação das palavras da linguagem que se pretende definir.

Os pares ordenados em P, são designados por *regras de re-escrita* ou *produções*, e serão denotados por $A \to B$, sendo que o símbolo "$\to$" é um meta-símbolo e como tal não pertence a $V_N \cup V_T$. As produções vão ser usadas para formar as palavras da linguagem que se pretende definir.

Definição 2.9 (Derivação num Passo) *Dado uma gramática* $G = (V_N, V_T, I, P)$ *e duas palavras* $X, Y \in (V_N, V_T)^*$*, diz-se que* Y *é derivável de* X *num passo,* $X \underset{G}{\Rightarrow} Y$*, sse existem palavras* P_1 *e* P_2 *em* $(V_N, V_T)^*$ *e uma produção* $A \rightarrow B$ *em* P*, tal que* $X = P_1 A P_2$ *e* $Y = P_1 B P_2$*.*

Definição 2.10 (Derivação) *Dado uma gramática* $G = (V_N, V_T, I, P)$ *e duas palavras* $X, Y \in (V_N, V_T)^*$*, diz-se que* Y *é derivável de* X*,* $X \overset{*}{\underset{G}{\Rightarrow}} Y$*, sse* $X = Y$*, ou existe uma palavra* Z *em* $(V_N, V_T)^*$ *tal que* $X \overset{*}{\underset{G}{\Rightarrow}} Z$*, e* $Z \underset{G}{\Rightarrow} Y$

Ou seja $\overset{*}{\underset{G}{\Rightarrow}}$ é o fecho reflexivo e transitivo de $\underset{G}{\Rightarrow}$. É também usual usar-se o fecho transitivo de $\underset{G}{\Rightarrow}$, o qual se denota por $\overset{+}{\underset{G}{\Rightarrow}}$, isto é efectua-se pelo menos uma redução.

Definição 2.11 (Linguagem Gerada por G) *A linguagem gerada pela gramática* G *é definida como sendo:*

$$L(G) = \left\{ Q \mid I \overset{*}{\underset{G}{\Rightarrow}} Q \text{ e } Q \in V_T^* \right\}$$

Ou seja a linguagem gerada por G contém exactamente as palavras que são deriváveis do símbolo inicial I e que contêm só símbolos terminais.

Uma *derivação termina* quando já não restam símbolos não-terminais na palavra.

Uma *derivação falha* quando já não é possível aplicar nenhuma regra de re-escrita de P, e ainda existem símbolos não-terminais na palavra.

Existem duas estratégias distintas para efectuar a derivação de uma frase: derivações pela esquerda e derivações pela direita.

Definição 2.12 (Derivação pela Direita (Esquerda)) *Uma derivação pela direita (esquerda) é toda a derivação*

$$I = f_0 \Rightarrow f_1 \ldots \Rightarrow f_{n-1} \Rightarrow f_n = f$$

em que f_i *obtém-se de* f_{i-1} *por substituição do não-terminal mais à esquerda (direita) em* f_{i-1}*.*

2.3.1 Hierarquia de Linguagens

Como se depreende da definição *toda a gramática gera uma linguagem única*, mas *a mesma linguagem pode ser gerada por muitas gramáticas diferentes.* Põe-se então a questão de classificar as gramáticas de acordo com o tipo de linguagens que são geradas por elas.

Definição 2.13 (Equivalência fraca de Gramáticas) *Duas gramáticas são designadas (fracamente) equivalentes se elas geram a mesma linguagem.*

As gramáticas vão ser classificadas em diferentes tipos consoante as restrições impostas às regras de derivação, o esquema apresentado toma o nome do Matemático que a propôs, N. Chomsky.

Definição 2.14 (Classificação de Gramáticas) *Uma gramática generativa $G = (V_N, V_T, I, P)$ é dita do tipo i se satisfaz as seguintes restrições impostas às regras de derivação:*

$i = 0$ **(gramáticas de frases estruturadas)** *Sem restrições.*

$i = 1$ **(gramáticas sensíveis ao contexto)** *Toda a regra em P tem a forma $Q_1AQ_2 \rightarrow Q_1BQ_2$, com Q_1, Q_2, e B em $(V_N \cup V_T)^*$, A em V_N, e $B \neq \epsilon$, excepto para a regra $I \rightarrow \epsilon$ que pode ocorrer em P, caso em que I não ocorre no lado direito das produções.*

$i = 2$ **(gramáticas livres do contexto)** *Toda a regra em P tem a forma $A \rightarrow B$ com A em V_N e B em $(V_N \cup V_T)^*$.*

$i = 3$ **(gramáticas regulares)** *Toda a regra em P tem a forma $A \rightarrow BQ$ ou $A \rightarrow B$, com A e Q em V_N e B em V_T^*.*

Uma Linguagem é dita do tipo i se é gerada por uma gramática do tipo i. A classe das linguagens do tipo i é designada por $\mathcal{L}_i$. Prova-se que $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$ com todas as inclusões próprias.

As linguagens do tipo 2 são as mais apropriadas para descrever linguagens artificiais, nomeadamente as linguagens de programação, as linguagens do tipo 1 são também usadas nesse contexto mas sempre que seja necessário ter em conta o contexto. As linguagens do tipo 3 estão relacionadas com a teoria dos autómatos finitos. As linguagens do tipo 0 são as linguagens que podem ser usadas para descrever as linguagens naturais.

2.3.2 Representação de Gramáticas

As duas representações mais usuais para gramáticas do tipo 2 e 3 são dadas pela notação BNF (Backus-Naur Form), e pelos diagramas de sintaxe. Vejamos de seguida as características de cada uma delas.

Backus-Naur Form

- Os símbolos terminais são escritos com maiúsculas.
- Os símbolos não-terminais são escritos com minúsculas.
- Numa produção os lados esquerdo e direito são separados pelo meta-símbolo $\rightarrow$, ou pelo meta-símbolo ::=.

- Produções alternativas são escritas numa só produção através do meta-símbolo "|" com o significado de alternativa. Por exemplo as regras $A ::= b_1$, $A ::= b_2, \ldots, A ::= b_n$ podem ser escritas através da seguinte produção $A ::= b_1|b_2|\ldots|b_n$.

- Se o lado direito de uma produção não contém nenhum símbolo então escreve-se $A ::= \epsilon$.

EBNF (Extended Backus-Naur Form) A notação EBNF extende a notação BNF com mais duas regras com vista à simplificação da escrita das gramáticas.

- A repetição de um dado símbolo n vezes ($n \geq 0$) é representada através dos meta-símbolos "{" e "}". Por exemplo $A ::= a\{b\}c$ que nos descreve uma produção em que o símbolo "b" pode ser repetido $n \geq 0$ vezes, é uma simplificação das produções $A ::= aBc$ e $B ::= \epsilon|Bb$.

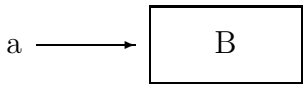
- A representação de elementos opcionais pode ser feita através da utilização dos meta-símbolos "[" e "]". Por exemplo $A ::= a[b]c$ que nos descreve uma produção em que o símbolo "b" ser omitido, é uma simplificação das produções $A ::= aBc$ e $B ::= \epsilon|b$.

Diagramas de Sintaxe Outra representação também muito usada, dada a sua fácil interpretação, é a dos diagramas de sintaxe.

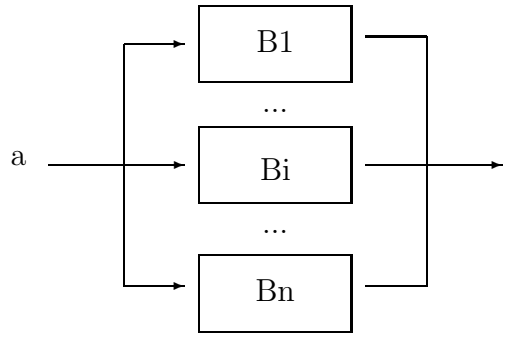
- Símbolos terminais são representados circunscritos em rectângulos ovalados (ou mesmo em círculos). Por exemplo:

- Símbolos não-terminais são representados circunscritos em rectângulos. Por exemplo:

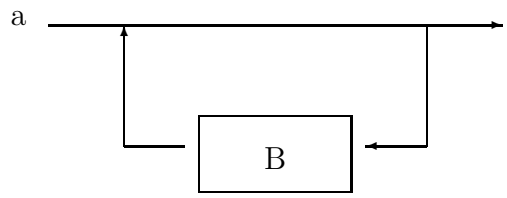
- As produções na forma $a ::= B$ são representadas pelos diagrama:



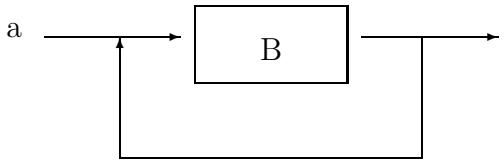
- As produções na forma $a ::= B_1|B_2|\ldots|B_n$ são representadas pelos diagrama:



- As produções na forma $a ::= \{B\}$ são representadas pelos diagrama:



Com base neste conjunto de regras é possível criar algumas variantes, as quais são facilmente interpretadas "visualmente". Por exemplo o diagrama



dá-nos a representação gráfica da produção $a ::= B|\{B\}$.

2.3.3 Gramáticas Reduzidas

Duas gramáticas independentes do contexto distintas podem gerar a mesma linguagem, dentro destas é importante determinar quais a que são mínimas, isto é, contêm o menor números de regras sintácticas.

Definição 2.15 (Gramática Reduzida) *Uma gramática independente do contexto diz-se reduzida se verificar as seguintes situações:*

- *não existem regras na forma $X ::= X$;*
- *cada símbolo não-terminal deriva numa sub-frase da linguagem;*

- *todos os símbolos terminais são resultado de derivações.*

Antes de ver a forma de transformação de uma gramática numa gramática reduzida vejamos algumas definições auxiliares.

Definição 2.16 (Símbolo Anulável) *Um símbolo não-terminal X de uma gramática diz-se anulável se existe*

$$X \stackrel{*}{\Rightarrow} \epsilon$$

i.e. a partir de X consegue-se derivar a frase nula.

Os símbolos anuláveis podem, e devem, ser eliminados da definição da gramática.

Definição 2.17 (Símbolo Activo) *Um símbolo não-terminal X de uma gramática diz-se activo, se verificar a seguinte condição, $\exists_{a \in V_T} : X \underset{G}{\stackrel{*}{\Rightarrow}} a$.*

A forma de obter uma gramática sem símbolos não activos passa pela identificação sistemática dos símbolos activos, o que pode ser feito pelo seguinte algoritmo.

$w_1 \leftarrow \{A \mid \{A ::= \alpha\} \subseteq P, \alpha \in V_T^*\}$
$k \leftarrow 1$
repetir
    $w_k \leftarrow w_{k-1} \cup \{A \mid \{A ::= \alpha\} \subseteq P, \alpha \in (V_T \cup w_{k-1})^*\}$
até $w_k = w_{k-1}$

Exercício 2.1 *Encontre os símbolos não activos da seguinte gramática.*

```
I ::= aA
A ::= Ib|bBB
B ::= abb|aC
C ::= aCA
```

Definição 2.18 (Símbolo Acessível) *Um símbolo α de uma gramática diz-se acessível, se for o resultado de uma derivação, isto é, $I \underset{G}{\stackrel{*}{\Rightarrow}} \alpha$.*

A forma de obter uma gramática sem símbolos inacessíveis é semelhante à obtenção de uma gramática sem símbolos não activos.

$w_1 \leftarrow I$
$k \leftarrow 1$
repetir
    $w_k \leftarrow w_{k-1} \cup \{A \mid \{B ::= \alpha A \beta\} \subseteq P, B \in w_{k-1}, \{\alpha, \beta\} \subseteq (V_T \cup w_{k-1})\}$
até $w_k = w_{k-1}$

Exercício 2.2 *Determine os símbolos inacessíveis da seguinte gramática:*

```
I ::= cAb
A ::= a|Aa
B ::= b|Bb
```

A transformação de uma gramática numa gramática reduzida passa então por:

- eliminação da regras $X ::= X$;
- eliminação dos símbolos não activos;
- eliminação dos símbolos não acessíveis.

Outro problema que pode surgir na definição de gramáticas é a ambiguidade.

2.3.4 Gramáticas Ambíguas

Definição 2.19 (Gramáticas Ambíguas) *Uma gramática independente do contexto $G = (V_T, V_N, I, P)$ diz-se ambígua se existirem duas sequências de derivações que, partindo de I, derivam a mesma frase $F \in V_T^*$.*

Por exemplo a seguinte gramática de expressões aritméticas

$$G = (\{\text{Exp}, \text{Num}\}, \{+, *, 0, 1, \ldots, 9\}, I, P)$$

com P o conjunto formado pelas duas seguintes produções:

```
Exp ::= Exp + Exp | Exp * Exp | Num
Num ::= 0 | 1 | ... | 9
```

Vejamos se a frase "3+5*7" pertence à linguagem gerada por G.

Primeira possibilidade (a colocação dos parêntesis é unicamente para aumentar a legibilidade)

```
Exp => Exp + Exp => (Exp + Exp) * Exp => (Num + Exp) * Exp =>
 => (Num + Num) * Exp => (Num + Num) * Num => ... => 3+5*7
```

Segunda possibilidade (a colocação dos parêntesis é unicamente para aumentar a legibilidade)

```
Exp => Exp * Exp => Exp + (Exp * Exp) => ... =>
 => Num + (Num * Num) => ... => 3+5*7
```

Temos assim duas sequências de derivações distintas mas que, partindo do símbolo inicial, derivam a mesma frase.

Dado que em geral (como acontece neste caso particular) duas sequências de derivação distintas estão associadas a dois significados distintos $(3 + (5 * 7) \neq (3 + 5) * 7)$, é então importante construir gramáticas não ambíguas.

Em alguns casos é possível eliminar de uma forma sistemática a ambiguidade na gramática, é esse o caso quando se pretende introduzir vários níveis de precedência entre alguns dos símbolos da gramática, isso é feito gerando símbolos não-terminais para cada nível de prioridade.

Em cada nível de prioridade o lado direito da produção deve conter os operadores desse nível de prioridade, e os símbolos não-terminais do nível de prioridade superior. No nível de prioridade mais elevado, o lado direito da produção contém os símbolos terminais das folhas da árvore de derivação.

Para o exemplo que estamos a tratar teríamos:

```
Exp ::= Exp + Termo | Termo
Termo ::= Termos * Factor | Factor
Factor ::= 0 | 1 | ... | 9
```

Podemos agora definir outro tipo de gramáticas.

2.3.5 Gramáticas Recursivas

Definição 2.20 (Gramáticas Recursivas) *Uma gramática $G = (V_N, V_T, I, P)$ diz-se recursiva à esquerda (direita) se existir um $A \in V_N$ tal que que existe uma derivação $A \stackrel{*}{\Rightarrow} A\alpha$ ($A \stackrel{*}{\Rightarrow} \alpha A$) para qualquer frase α.*

A conversão de gramáticas regulares em expressões regulares nem sempre é possível, o contrário é sempre possível, veremos mais à frente como o fazer

2.4 Formas Normais

As formas normais (e o processo de normalização) são formas de construção de gramáticas simplificadas mas sem perda de poder expressivo.

Definição 2.21 (Forma Normal de Chomsky) *Uma gramática livre do contexto $G = (V_T, V_N, I, P)$ é expressa na forma normal de Chomsky se cada produção possuir uma das seguintes formas:*

- $I ::= \alpha$;
- $A ::= BC$, *na qual* $B, C \in V_N$ *e se* $A \neq I$, *então* $BC \neq I$;
- $A ::= a$, *na qual* $a \in V_T$.

Podemos obter a forma normal de Chomsky de uma gramática independente do contexto através do seguinte algoritmo (explicado através da sua aplicação a um exemplo).

Considere-se o seguinte conjunto de produções:

```
I ::= aAA | bB
A ::= B | c
B ::= bB | b
```

1. Identificar uma gramática equivalente para a qual os símbolos terminais no lado direito da produção surgem isolados (são os únicos símbolos do lado direito da produção).

   ```
   I ::= aAA
   ```

   criar um novo símbolo, introduzir uma nova produção do tipo "novo símbolo deriva símbolo repetido", substituir o símbolo repetido pelo novo símbolo.

   ```
   I ::= CAA
   C ::= a
   B ::= DB
   D ::= b
   ```

2. Identificar uma gramática equivalente na qual o lado direito das produções com símbolos não-terminais é formado apenas por dois símbolos. O processo vai ser semelhante ao anterior com a introdução de novos símbolos e novas produções.

   A produção:

   ```
   I ::= CAA
   ```

   é substituída por:

   ```
   I ::= CE
   E ::= AA
   ```

3. Substituir as produções do tipo $A ::= E$ com $E \in V_N$ por produções em que E é substituído pelo resultado das produções que ele encabeça.

   A produção:

   ```
   A ::= B
   ```

é substituída por:

```
A ::= DB | b
```

Teríamos então o seguinte conjunto de produções.

```
I ::= CE | DB
A ::= DB | b | c
B ::= DB | b
C ::= a
D ::= b
E ::= AA
```

As gramáticas na forma normal tendem a ter um maior número de produções, estas são no entanto mais simples (na sua forma) do que as produções numa gramática equivalente que não esteja na forma normal.