10.2. Languages and Grammars

10.2.1. Formal Languages. Consider algebraic expressions written with the symbols $A = \{x, y, z, +, *, (,)\}$. The following are some of them: "$x + y * y$", "$y + (x * y + y) * x$", "$(x + y) * x + z$", etc. There are however some strings of symbols that are not legitimate algebraic expressions, because they have some sort of syntax error, e.g.: "$(x + y$", "$z + +y * x$", "$x(*y) + z$", etc. So syntactically correct algebraic expressions are a subset of the whole set A^* of possible strings over A.

In general, given a finite set A (the *alphabet*), a *(formal) language* over A is a subset of A^* (set of strings of A).

Although in principle any subset of A^* is a formal language, we are interested only in languages with certain structure. For instance: let $A = \{a, b\}$. The set of strings over A with an even number of a's is a language over A.

10.2.2. Grammars. A way to determine the structure of a language is with a *grammar*. In order to define a grammar we need two kinds of symbols: *non-terminal*, used to represent given subsets of the language, and *terminal*, the final symbols that occur in the strings of the language. For instance in the example about algebraic expressions mentioned above, the final symbols are the elements of the set $A = \{x, y, z, +, *, (,)\}$. The non-terminal symbols can be chosen to represent a complete algebraic expression (E), or terms (T) consisting of product of factors (F). Then we can say that an algebraic expression E consists of a single term

$$E \to T,$$

or the sum of an algebraic expression and a term

$$E \to E + T.$$

A term may consists of a factor or a product of a term and a factor

$$T \to F$$

$$T \to T * F$$

A factor may consists of an algebraic expression between parenthesis

$F \to (E)$,

or an isolated terminal symbol

$F \to x$,

$F \to y$,

$F \to z$.

Those expressions are called *productions*, and tell us how we can generate syntactically correct algebraic expressions by replacing successively the symbols on the left by the expressions on the right. For instance the algebraic expression "'$y + (x * y + y) * x$" can be generated like this:

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow y+T \Rightarrow y+T*F \Rightarrow y+F*F \Rightarrow$$

$$y+(E)*F \Rightarrow y+(E+T)*F \Rightarrow y+(T+T)*F \Rightarrow y+(T*F+T)*F \Rightarrow$$

$$y+(F*F+T)*F \Rightarrow y+(x*T+T)*F \Rightarrow y+(x*F+T)*F \Rightarrow$$

$$y+(x*y+T)*F \Rightarrow y+(x*y+F)*T \Rightarrow y+(x*y+y)*F \Rightarrow$$

$$y+(x*y+y)*x\,.$$

In general a *phrase-structure grammar* (or simply, *grammar*) G consists of

1. A finite set V of symbols called *vocabulary* or *alphabet*.
2. A subset $T \subseteq V$ of *terminal symbols*. The elements of $N = V - T$ are called *nonterminal symbols* or *nonterminals*.
3. A start symbol $\sigma \in N$.
4. A finite subset P of $(V^* - T^*) \times V^*$ called the set of *productions*.

We write $G = (V, T, \sigma, P)$.

A production $(A, B) \in P$ is written:

$$A \to B\,.$$

The right hand side of a production can be any combination of terminal and nonterminal symbols. The left hand side must contain at least one nonterminal symbol.

If $\alpha \to \beta$ is a production and $x\alpha y \in V^*$, we say that $x\beta y$ is *directly derivable* from $x\alpha y$, and we write

$$x\alpha y \Rightarrow x\beta y\,.$$

If we have $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ $(n \geq 0)$, we say that α_n is *derivable* from α_1, and we write $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ (by convention also $\alpha_1 \overset{*}{\Rightarrow} \alpha_1$.)

Given a grammar G, the language $L(G)$ associated to this grammar is the subset of T^* consisting of all strings derivable from σ.

10.2.3. Backus Normal Form. The *Backus Normal Form* or *BNF* is an alternative way to represent productions. The production $S \to T$ is written $S ::= T$. Productions of the form $S ::= T_1$, $S ::= T_2$, $\ldots$, $S ::= T_n$, can be combined as

$$S ::= T_1 \mid T_2 \mid \cdots \mid T_n\,.$$

So, for instance, the grammar of algebraic expressions defined above can be written in BNF as follows:

$E ::= T \mid E + T$

$T ::= F \mid T * F$

$F ::= (E) \mid x \mid y \mid z$

10.2.4. Combining Grammars. Let $G_1 = (V_1, T_1, \sigma_1, P_1)$ and $G_2 = (V_2, T_2, \sigma_2, P_2)$ be two grammars, where $N_1 = V_1 - T_1$ and $N_2 = V_2 - T_2$ are disjoint (rename nonterminal symbols if necessary). Let $L_1 = L(G_1)$ and $L_2 = L(G_2)$ be the languages associated respectively to G_1 and G_2. Also assume that σ is a new symbol not in $V_1 \cup V_2$. Then

1. *Union Rule*: the language *union of* L_1 *and* L_1

$$L_1 \cup L_2 = \{\alpha \mid \alpha \in L_1 \text{ or } \alpha \in L_1\}$$

starts with the two productions

$$\sigma \to \sigma_1\,, \quad \sigma \to \sigma_2\,.$$

2. *Product Rule*: the language *product of* L_1 *and* L_2

$$L_1 L_2 = \{\alpha\beta \mid \alpha \in L_1,\, \beta \in L_1\}$$

where $\alpha\beta$ = string concatenation of α and β, starts with the production

$$\sigma \to \sigma_1\sigma_2\,.$$

3. *Closure Rule*: the language *closure of* L_1

$$L_1^* = L_1^0 \cup L_1^1 \cup L_1^2 \cup \dots$$

were $L_1^0 = \{\lambda\}$ and $L_1^n = \{\alpha_1\alpha_2\dots\alpha_n \mid \alpha_k \in L_1,\ k = 1, 2, \dots, n\}$ $(n = 1, 2, \dots)$, starts with the two productions

$$\sigma \to \sigma_1\sigma\,, \quad \sigma \to \lambda\,.$$

10.2.5. Types of Grammars (Chomsky's Classification). Let G be a grammar and let λ denote the null string.

0. G is a *phrase-structure* (or type 0) *grammar* if every production is of the form:

$$\alpha \to \delta\,,$$

where $\alpha \in V^* - T^*$, $\delta \in V^*$.

1. G is a *context-sensitive* (or type 1) *grammar* if every production is of the form:

$$\alpha A\beta \to \alpha\delta\beta$$

(i.e.: we may replace A with δ in the context of α and β), where $\alpha, \beta \in V^*$, $A \in N$, $\delta \in V^* - \{\lambda\}$.

2. G is a *context-free* (or type 2) *grammar* if every production is of the form:

$$A \to \delta\,,$$

where $A \in N$, $\delta \in V^*$.

3. G is a *regular* (or type 3) *grammar* if every production is of the form:

$$A \to a \quad \text{or} \quad A \to aB \quad \text{or} \quad A \to \lambda\,,$$

where $A, B \in N$, $a \in T$.

A language L is *context-sensitive* (respectively *context-free*, *regular*) if there is a context-sensitive (respectively context-free, regular) grammar G such that $L = L(G)$.

The following examples show that these grammars define different kinds of languages.

Example: The following language is type 3 (regular):

$$L = \{a^n b^m \mid n = 1, 2, 3 \dots ; m = 1, 2, 3, \dots \}\,.$$

A type 3 grammar for that language is the following: $T = \{a, b\}$, $N = \{\sigma, S\}$, with start symbol σ, and productions:

$$\sigma \to a\sigma\,, \quad \sigma \to aS\,, \quad S \to bS\,, \quad S \to b\,.$$

Example: The following language is type 2 (context-free) but not type 3:

$$L = \{a^n b^n \mid n = 1, 2, 3, \dots \}\,.$$

A type 2 grammar for that language is the following:

$T = \{a, b\}$, $N = \{\sigma\}$, with start symbol σ, and productions

$$\sigma \to a\sigma b\,, \quad \sigma \to ab\,.$$

Example: The following language is type 1 (context-sensitive) but not type 2:

$$L = \{a^n b^n c^n \mid n = 1, 2, 3, \dots \}\,.$$

A type 1 grammar for that language is the following:

$T = \{a, b, c\}$, $N = \{\sigma, A, C\}$, with start symbol σ, and productions

$$\begin{array}{ll} \sigma \to abc\,, & \sigma \to aAbc\,, \\ A \to abC\,, & A \to aAbC\,, \\ Cb \to bC\,, & Cc \to cc\,. \end{array}$$

There are also type 0 languages that are not type 1, but they are harder to describe.

10.2.6. Equivalent Grammars. Two grammars G and G' are *equivalent* if $L(G) = L(G')$.

Example: The grammar of algebraic expressions defined at the beginning of the section is equivalent to the following one:

Terminal symbols $= \{x, y, z, +, *, (,)\}$, nonterminal symbols $= \{E, T, F, L\}$, with start symbol E, and productions

$E \to T, \quad E \to E + T,$

$T \to F, \quad T \to T * F$

$F \to (E), \quad F \to L,$

$L \to x, \quad L \to y, \quad L \to z.$

10.2.7. Context-Free Interactive Lindenmayer Grammar. A context-free interactive Lindenmayer grammar is similar to a usual context-free grammar with the difference that it allows productions of the form $A \to B$ where $A \in N \cup T$ (in a context free grammar A must be nonterminal). Its rules for deriving strings also are different. In a context-free interactive Lindenmayer grammar, to derive string β from string α, all symbols in α must be replaced *simultaneously*.

Example: The von Koch Snowflake. The von Koch Snowflake is a fractal curve obtained by start with a line segment and then at each stage transforming all segments of the figure into a four segment polygonal line, as shown below. The von Koch Snowflake fractal is the limit of the sequence of curves defined by that process.

FIGURE 10.1. Von Koch Snowflake, stages 1–3.

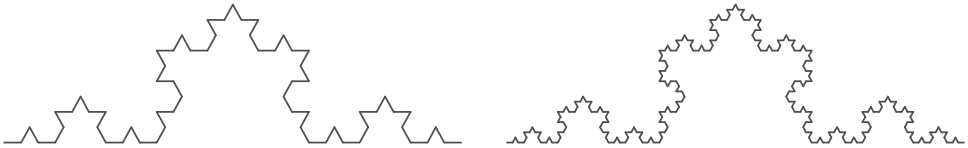

FIGURE 10.2. Von Koch Snowflake, stages 4–5

A way to represent an intermediate stage of the making of the fractal is by representing it as a sequence of movements of three kinds: 'd'= draw a straight line (of a fix length) in the current direction, 'r'= turn right by 60°, 'l'= turn left by 60°. For instance we start with a single horizontal line d, which we then transform into the polygonal $dldrrdld$, then each segment is transformed into a polygonal according to the rule $d \to dldrrdld$, so we get

$$dldrrdldldldrrdldrrdldrrdldldldrrdld$$

If we represent by D a segment that may no be final yet, then the sequences of commands used to build any intermediate stage of the curve can be defined with the following grammar:

$N = \{D\}$, $T = \{d, r, l\}$, with start symbol D, and productions:

$$D \to DlDrrDlD\,, \quad D \to d\,, \quad r \to r\,, \quad l \to l\,.$$

Example: The Peano curve. The Peano curve is a space filling curve, i.e., a function $f : [0,1] \to [0,1]^2$ such that the range of f is the whole square $[0,1]^2$, defined as the limit of the sequence of curves shown in the figures below.

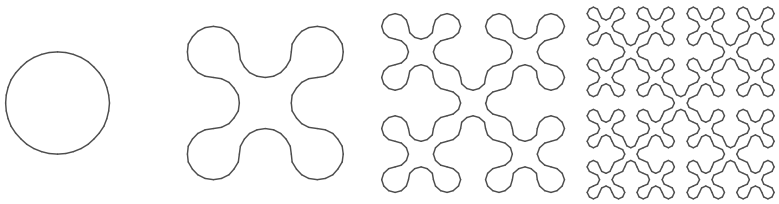

FIGURE 10.3. Peano curve, stages 1–4.

Each element of that sequence of curves can be described as a sequence of 90° arcs drawn either anticlockwise ('l') or clockwise ('r'). The corresponding grammar is as follows:

$T = \{l, r\}$, $N = \{C, L, R\}$, with and start symbol C, and productions

$$C \to LLLL\,,$$

$$L \to RLLLR\,, \quad R \to RLR\,,$$

$$L \to l\,, \quad R \to r\,, \quad l \to l\,, \quad r \to r\,.$$