TALN 2001, Tours, 2-5 juillet 2001

Formal Languages for Linguists: Classical and Nonclassical Models

Extended version of the chapter 8: C. Martín-Vide, Formal grammars and languages, in R. Mitkov, ed., *Oxford Handbook of Computational Linguistics.* Oxford University Press, Oxford, 2001.

Carlos Martín-Vide
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco, 1
43005 Tarragona, Spain
E-mail: cmv@correu.urv.es, cmv@nil.fut.es
Web: http://www.urv.es/centres/Grups/grlmc/grlmc.html

Abstract

The basics of classical formal language theory are introduced, as well as a wide coverage is given of some new nonstandard devices motivated in molecular biology, which are challenging traditional conceptions, are making the theory revived and could have some linguistic relevance. Only definitions and a few results are presented, without including any proof. The chapter can be profitably read without any special previous mathematical background. A long list of references completes the chapter, which intends to give a flavour of the field and to encourage young researchers to go deeper into it.

1 Languages

1.1 Basic Notions

An **alphabet** or **vocabulary** V is a finite set of letters. By concatenating the letters from V again and again, one obtains V^*, an infinite set of **strings** or **words**. The **empty string** is denoted by λ and contains no letter: it is the unit element of V^* under the concatenation operation. The **concatenation** of strings is an associative and noncommutative operation, which closes V^*, i.e.:

$$\text{for every } w, v \in V^* : wv \in V^*.$$

The **length** of a string, denoted by $|w|$, is the number of letters the string consists of. It is clear that:

- $|\lambda| = 0$,

- $|wv| = |w| + |v|$.

w is a **substring** or **subword** of v if and only if there exist u_1, u_2 such that $v = u_1wu_2$. Special cases of a substring include:

- if $w \neq \lambda$ and $w \neq v$, then w is a **proper substring** of v,
- if $u_1 = \lambda$, then w is a **prefix** or a **head**,
- if $u_2 = \lambda$, then w is a **suffix** or a **tail**.

The i-times **iterated concatenation** of w is practically showed in the following:

Example 1.1 *If $w = ab$, then $w^3 = (ab)^3 = ababab$. $(w^0 = \lambda$.)*

If $w = a_1a_2 \ldots a_n$, then its **mirror image** $w^{-1} = a_na_{n-1} \ldots a_1$. It is clear that:

$$(w^{-1})^{-1} = w,\ (w^{-1})^i = (w^i)^{-1} \text{ (for every } i \geq 0).$$

Any subset $L \subseteq V^*$ (including both $\emptyset$ and $\{\lambda\}$) is a **language**. One denotes $V^+ = V^* - \{\lambda\}$.

Regarding **cardinality** (generally, the number of elements a set contains):

- V^* is denumerably infinite, i.e. $|V^*| = \aleph_0$ (the smallest transfinite number),
- $\mathcal{P}(V^*)$ is nondenumerably infinite, i.e. $|\mathcal{P}(V^*)| = 2^{\aleph_0}$ (also called $\aleph_1$).

We do not go here deeper into the details of infinite sets, which would require an extensive presentation.

Examples of languages include:

Example 1.2 $L = \{a, b, \lambda\}$, $L = \{a^i, b^i : i \geq 0\}$, $L = \{ww^{-1} : w \in V^*\}$, $L = \{a^{n^2} : n \geq 1\}$, $L = \{w : w \in \{a, b\}^+$ *and* $|w|_a = |w|_b\}$ *($|w|_x$ denotes the number of occurrences of x in w).*

1.2 Chomsky Grammars. The Chomsky Hierarchy

A (formal) **grammar** is a construct $G = (N, T, S, P)$, where N, T are alphabets, with $N \cap T = \emptyset$, $S \in N$ and P is a finite set of pairs (w, v) such that $w, v \in (N \cup T)^*$ and w contains at least one letter from N. ((w, v) uses to be written $w \rightarrow v$.) N is the **nonterminal alphabet**, T the **terminal alphabet**, S the **initial letter** or **axiom**, and P the set of **rewriting rules** or **productions**.

Given $G = (N, T, S, P)$ and $w, v \in (N \cup T)^*$, an **immediate** or **direct derivation** (in 1 step) $w \Longrightarrow_G v$ holds if and only if:

(i) there exist $u_1, u_2 \in (N \cup T)^*$ such that $w = u_1\alpha u_2$ and $v = u_1\beta u_2$, and

(ii) there exists $\alpha \longrightarrow \beta \in P$.

Given $G = (N, T, S, P)$ and $w, v \in (N \cup T)^*$, a **derivation** $w \Longrightarrow_G^* v$ holds if and only if either $w = v$ or there exists $z \in (N \cup T)^*$ such that $w \Longrightarrow_G^* z$ and $z \Longrightarrow_G v$.

$\Longrightarrow_G^*$ denotes the reflexive transitive closure and $\Longrightarrow_G^+$ the transitive closure, respectively, of $\Longrightarrow_G$.

The **language** generated by a grammar is defined by:

$$L(G) = \{w : S \Longrightarrow_G^* w \text{and} w \in T^*\}.$$

Example 1.3 *Let $G = (N, T, S, P)$ be a grammar such that:*

$N = \{S, A, B\}$,

$T = \{a, b\}$,

$P = \{S \longrightarrow aB, S \longrightarrow bA, A \longrightarrow a, A \longrightarrow aS, A \longrightarrow bAA, B \longrightarrow b, B \longrightarrow bS, B \longrightarrow aBB\}$.

The language generated by G is the following:

$$L(G) = \{w : w \in \{a, b\}^+ and |w|_a = |w|_b\}.$$

Example 1.4 *Let $G = (N, T, S, P)$ be a grammar such that:*

$N = \{S, A, B\}$,

$T = \{a, b, c\}$,

$P = \{S \longrightarrow abc, S \longrightarrow aAbc, Ab \longrightarrow bA, Ac \longrightarrow Bbcc, bB \longrightarrow Bb, aB \longrightarrow aaA, aB \longrightarrow aa\}$.

The language generated by G is the following:

$$L(G) = \{a^n b^n c^n : n \geq 1\}.$$

Grammars can be classified according to several criteria. The most spread one is the form of their productions. According to it, a grammar is said to be of type:

- 0 (**phrase-structure grammar, RE**) if and only if there are no restrictions on the form of the productions: everything at both the left-hand side and the right-hand side of the rules is allowed.

- 1 (**context-sensitive grammar, CS**) if and only if every production is of the form:

$$u_1Au_2 \longrightarrow u_1wu_2,$$

  with $u_1, u_2, w \in (N \cup T)^*$, $A \in N$ and $w \neq \lambda$ (except possibly for the rule $S \longrightarrow \lambda$, in which case S does not occur on any right-hand side of a rule).

- 2 (**context-free grammar, CF**) if and only if every production is of the form:

$$A \longrightarrow w,$$

  with $A \in N$, $w \in (N \cup T)^*$.

- 3 (**regular or finite-state grammar, REG**) if and only if every production is of any of the forms:

$$\begin{gathered} A \longrightarrow wB, \\ A \longrightarrow w, \end{gathered}$$

  with $A, B \in N$, $w \in T^*$.

A language is said to be of type i ($i = 0, 1, 2, 3$) if it is generated by a type i grammar. The family of type i languages is denoted by $\mathcal{L}_i$.

One of the most important and early results in formal language theory is the so-called **Chomsky hierarchy** of languages: $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$.

Note that every grammar generates a unique language. However, one language can be generated by several different grammars.

Two grammars are said to be:

- **(weakly) equivalent** if they generate the same string language,

- **strongly equivalent** if they generate both the same string language and the same tree language.

(We shall see later that a context-free grammar generates not only a set of strings, but a set of trees too: each one of the trees is associated with one string and pictures the way how this latter is derived in the grammar.)

1.3 Operations on Languages

Usual set-theoretic operations on languages include:

- **Union**: $L_1 \cup L_2 = \{w : w \in L_1$ or $w \in L_2\}$.
- **Intersection**: $L_1 \cap L_2 = \{w : w \in L_1$ and $w \in L_2\}$.
- **Difference**: $L_1 - L_2 = \{w : w \in L_1$ and $w \notin L_2\}$.
- **Complement** of $L \subseteq V^*$ with respect to $V^* : \bar{L} = V^* - L$.

  Specific language-theoretic operations on languages include:

- **Concatenation**: $L_1 L_2 = \{w_1 w_2 : w_1 \in L_1$ and $w_2 \in L_2\}$.
- **Iteration**:

$$
\begin{aligned}
&L^0 = \{\lambda\},\\
&L^1 = L,\\
&L^2 = LL,\\
&\ldots\\
&L^* = \bigcup_{i \geq 0} L^i \text{ (closure of the iteration: \textbf{Kleene star})},\\
&L^+ = \bigcup_{i \geq 1} L^i \text{ (positive closure of the iteration: \textbf{Kleene plus})}.
\end{aligned}
$$

  Note that L^+ equals L^* if $\lambda \in L$, and equals $L^* - \{\lambda\}$ if $\lambda \notin L$.

- **Mirror image**: $L^{-1} = \{w : w^{-1} \in L\}$.
  Note that $(L^{-1})^{-1} = L$ and $(L^{-1})^i = (L^i)^{-1}$, for every $i \geq 0$.
- **Right quotient** of L_1 over L_2: $L_1/L_2 = \{w :$ there exists $v \in L_2$ such that $wv \in L_1\}$.
- **Right derivative** of L over v: $\partial_v^r L = L/\{v\} = \{w : wv \in L\}$.
- **Head** of $L \subseteq V^*$: $HEAD(L) = \{w \in V^* :$ there exists $v \in V^*$ such that $wv \in L\}$.
  Note that for every $L : L \subseteq HEAD(L)$.
- **Left quotient** of L_1 over L_2: $L_2 \backslash L_1 = \{w :$ there exists $v \in L_2$ such that $vw \in L_1\}$.
- **Left derivative** of L over v: $\partial_v^l L = \{v\} \backslash L = \{w : vw \in L\}$.
- **Tail** of $L \subseteq V^*$: $TAIL(L) = \{w \in V^* :$ there exists $v \in V^*$ such that $vw \in L\}$.
  Note that for every $L : L \subseteq TAIL(L)$.
- **Morphism**: Given two alphabets V_1, V_2, a mapping $h : V_1^* \longrightarrow V_2^*$ is a morphism if and only if:

  (i) for every $w \in V_1^*$, there exists $v \in V_2^*$ such that $v = h(w)$ and v is unique,
  (ii) for every $w, u \in V_1^* : h(wu) = h(w)h(u)$.

A morphism is called λ-**free** if, for every $w \in V_1^*$, if $w \neq \lambda$ then $h(w) \neq \lambda$.

- **Morphic image**: $h(L) = \{v \in V_2^* : v = h(w),\ \text{for some } w \in L\}$.
- A morphism is called an **isomorphism** if, for every $w, u \in V_1^*$, if $h(w) = h(u)$ then $w = u$.

Example 1.5 *An isomorphism between $V_1 = \{0, 1, 2, \ldots, 9\}$ and $V_2 = \{0, 1\}$ is the binary coded decimal representation of the integers:*

$$h(0) = 0000, h(1) = 0001, \ldots, h(9) = 1001.$$

Union, concatenation and Kleene star are called **regular operations**.

Theorem 1.6 *Each of the language families $\mathcal{L}_i$, for every $i \in \{0, 1, 2, 3\}$, is closed under regular operations.*

A systematic study of the common properties of language families has led to the theory of abstract families of languages (AFL's). An **abstract family of languages** is a class of those languages that satisfy certain specified closure axioms. If one fixes an AFL, one can prove general theorems about all languages in the family.

A few simple closure properties are depicted next:

	REG	CF	CS	RE
union	+	+	+	+
intersection	+	–	+	+
complement	+	–	+	–
concatenation	+	+	+	+
Kleene star	+	+	+	+
intersection with regular languages	+	+	+	+
morphisms	+	+	–	+
left/right quotient	+	–	–	+
left/right quotient with regular languages	+	+	–	+
left/right derivative	+	+	+	+
mirror image	+	+	+	+

The table must be interpreted in the following way. Using the first $+$ to the left as an example, it means that the union of two regular languages is always a regular language. And so on.

2 Grammars

2.1 Context-Free Grammars

Theorem 2.1 *For every CF grammar G, one can find an equivalent CF grammar G' such that the right-hand sides of its productions are all different from λ except when $\lambda \in L(G)$. In this* *latter case, $S \longrightarrow \lambda$ is the only rule with the right-hand side λ, but then S does not occur on any right-hand side of the rules. (This is also true for REG grammars.)*

A grammar is said to be **λ-free** if none of its rules has the right-hand side λ.

An CF grammar is said to be in **Chomsky normal form** (CNF) if each of its rules has either of the two following forms:

- $X \longrightarrow a, X \in N, a \in T,$
- $X \longrightarrow YZ, X, Y, Z \in N.$

Theorem 2.2 *For every λ-free CF grammar, one can effectively (algorithmically) find an equivalent grammar in CNF.*

Theorem 2.3 *For every CF grammar G, it is decidable whether or not an arbitrary strings w belongs to $L(G)$.*

Corollary 2.4 *Given an CF grammar G and a finite language L, both $L \subseteq L(G)$ and $L \cap L(G) = \emptyset$ are solvable.*

2.2 Derivation Trees

A very common and practical representation of the derivation process in a grammar (particularly, in an CF grammar) is a tree.

A **derivation tree** is defined as $T = (V, D)$, where V is a set of **nodes** or **vertices** and D is a **dominance** relation, which is a binary relation in V that satisfies:

(i) D is a weak order:

  (i.a) reflexive: for every $a \in V : aDa$,

  (i.b) antisymmetric: for every $a, b \in V$, if aDb and bDa, then $a = b$,

  (i.c) transitive: for every $a, b, c \in V$, if aDb and bDc, then aDc.

(ii) **root condition**: there exists $r \in V$ such that for every $b \in V : rDb$,

(iii) **nonbranching condition**: for every $a, a', b \in V$, if aDb and $a'Db$, then aDa' or $a'Da$.

Special cases of the dominance relation include, for every $a, b \in V$:

- a **strictly dominates** b ($aSDb$) if and only if aDb and $a \neq b$; so SD is a strict order in V:

  (i) irreflexive: it is not the case that $aSDa$,

(ii) asymmetric: if $aSDb$, then it is not the case that $bSDa$,

(iii) transitive: if $aSDb$ and $bSDc$, then $aSDc$.

- a **immediately dominates** b ($aIDb$) if and only if $aSDb$ and there does not exist any c such that $aSDc$ and $cSDb$.

The **degree** of a node $deg(b) = |\{a \in V : bIDa\}|$. Consequences of this definition are:

- b is a **terminal node** or a **leaf** if and only if $deg(b) = 0$,
- b is a **unary node** if and only if $deg(b) = 1$,
- b is a **branching node** if and only if $deg(b) > 1$,
- T is an **n-ary derivation tree** if and only if all its nonterminal nodes are of degree n.

Two nodes a, b are **independent** of each other: $aINDb$ if and only if neither aDb nor bDa.

Some family relations among nodes include:

- a is a **mother node** of b: aMb if and only if $aIDb$,
- a is a **sister node** of b: aSb if and only if there exists d such that dMa and dMb.

The mother relation has the following features:

(i) there does not exist any $a \in V$ such that aMr,

(ii) if $b \neq r$, then it has just one mother node.

Given $T = (V, D)$, for every $b \in V$, a **derivation subtree** or a **constituent** is $T_b = (V_b, D_b)$, where $V_b = \{c \in V : bDc\}$ and xD_by if and only if $x \in V_b$ and $y \in V_b$ and xDy.

Given $T = (V, D)$, for every $a, b \in V$: a **c-commands** b (aCCb) if and only if:

(i) $aINDb$,

(ii) there exists a branching node that strictly dominates a,

(iii) every branching node that strictly dominates a dominates b.

a **asymmetrically c-commands** b if and only if $aCCb$ and it is not the case that $bCCa$.

Given two derivation trees $T = (V, D)$, $T' = (V', D')$ and $h : V \to V'$:

- h **preserves** D if and only if for every $a, b \in V : aDb \to h(a)D'h(b)$.

- h is an **isomorphism** of T in T' ($T \approx T'$) if and only if h is a bijection and preserves D. (Note that a mapping $f : A \rightarrow B$ is a bijection if and only if:
  - (i) f is one-to-one or injective: for every $x, y \in A$, if $x \neq y$ then $f(x) \neq f(y)$ or, equivalently, if $f(x) = f(y)$ then $x = y$,
  - (ii) f is onto or exhaustive: for every $z \in B$: there exists $x \in A$ such that $f(x) = z$.)

Any two isomorphic derivation trees share all their properties:

- $aSDb$ if and only if $h(a)SD'h(b)$,
- $aIDb$ if and only if $h(a)ID'h(b)$,
- $deg(a) = deg(h(a))$,
- $aCCb$ if and only if $h(a)CCh(b)$,
- a is the root of T if and only if $h(a)$ is the root of T',
- $depth(a) = depth(h(a))$,

  ($depth(a) = |\{b \in V : bDa\}| - 1$.)
- $height(T) = height(T')$.

  ($height(T) = max\{depth(a) : a \in V\}$.]

Once one has an $T = (V, D)$, one may enrich its definition to get a **labelled derivation tree** $T = (V, D, L)$, where (V, D) is a derivation tree and L is a mapping from V to a specified set of labels.

Given $T = (V, D, L)$ and $T' = (V', D', L')$, one says $T \approx T'$ if and only if:

(i) $h : V \rightarrow V'$ is a bijection,

(ii) h preserves D,

(iii) for every $a, b \in V$: $L(a) = L(b)$ if and only if $L'(h(a)) = L'(h(b))$.

A **terminally ordered derivation tree** is $T = (V, D, <)$, where (V, D) is a derivation tree and $<$ is a strict total (or linear) order on the terminal nodes of V, i.e. a relation that is:

(i) irreflexive: for every a terminal: it is not the case that $a < a$,

(ii) asymmetric: if $a < b$, then it is not the case that $b < a$,

(iii) transitive: if $a < b$ and $b < c$, then $a < c$,

(iv) connected: either $a < b$ or $b < a$.

Given $T = (V, D, <)$, for every $b, c, d, e \in V : b <' c$ (b **precedes** c) if and only if:

$$\text{if } bDd,\ d \text{ is terminal, } cDe \text{ and } e \text{ is terminal, then } d < e.$$

The following **exclusivity condition** completely orders a tree. Given $T = (V, D, <)$, for every $b, d \in V$, if $bINDd$, then either $b <' d$ or $d <' b$). Consequently, every two nodes of the tree must hold one, and only one, of the dominance and precedence relations.

2.3 More about Context-Free Languages

An CF grammar is called **redundant** if it contains useless nonterminal letters. A nonterminal letter is **useless** if:

(i) either no terminal string is derivable from it: **inactive** or **dead** letter,

(ii) or it does not occur in any string derivable from S: **unreachable** letter.

Theorem 2.5 *For any CF grammar $G = (N, T, S, P)$:*

- *$A \in N$ is inactive if and only if the language generated by $G_A = (N, T, A, P)$ is empty.*
- *A is unreachable if and only if the language generated by $G_A^\lambda = ((N - \{A\}) \cup T, \{A\}, S, P_1 \cup \{X \to \lambda : X \in (N - \{A\}) \cup T\})$ (P_1 being the set of rules remaining after having removed from P the productions that have A on their left-hand sides) is $\{\lambda\}$.*

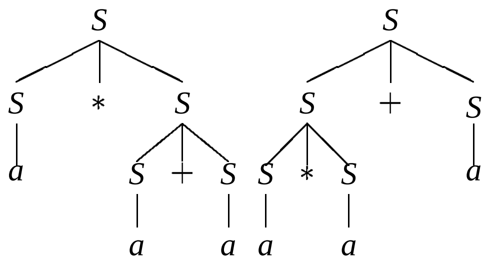
An CF grammar is **nonredundant** or **reduced** if each of its nonterminal letters is both active and reachable.

Theorem 2.6 *For every CF grammar, one can effectively (algorithmically) find an equivalent nonredundant grammar.*

Given an CF grammar $G, w \in L(G)$ is an **ambiguous** string if and only if w has at least two derivation trees in G. G is said to be an **ambiguous** grammar if and only if there exists some string in $L(G)$ that is ambiguous. L is an **inherently ambiguous** context-free language if and only if every CF grammar generating L is ambiguous.

An example of ambiguity is the following:

Example 2.7 *Given $G = (\{S\}, \{a, +, *\}, S, \{S \to S + S, S \to S * S, S \to a\}), w = a * a + a$ has two different derivation trees in G:*



An example of inherent ambiguity is the following:

Example 2.8 *Let* $L = \{a^m b^m c^n : m, n \geq 1\} \cup \{a^m b^n c^n : m, n \geq 1\}$*. Every CF grammar generating it will produce two different derivation trees for the strings of the form* $a^n b^n c^n$ *(which belong to both components of the language).*

The **Bar-Hillel pumping lemma** for CF languages allows one to prove that a language is not CF just looking at the structure of the strings:

for every $L \in CF$ there exist $p, q \in N$ such that for every $z \in L,$ if $|z| > p,$ then $z = uvwxy,$ where $u, v, w, x, y \in T^*$, $|vwx| \leq q, vx \neq \lambda,$ and $uv^i wx^i y \in L,$ for every $i \geq 0$.

Since not all languages satisfy the pumping lemma above, the following corollary is obvious.

Corollary 2.9 *There are noncontext-free languages.*

Examples of this include:

$\{a^n b^n c^n : n \geq 1\}$,

$\{a^{n^2} : n \geq 1\}$,

$\{a^n b^m c^n d^m : m, n \geq 1\}$.

2.4 Linear and Regular Languages

An CF grammar is called **linear** (LIN) if each production has either of the forms:

- $A \longrightarrow w, A \in N, w \in T^*,$
- $A \longrightarrow w_1 B w_2, A, B \in N, w_1, w_2 \in T^*.$

Further, it is called **left-linear** (LLIN) (respectively, **right-linear** (RLIN)) if w_1 (respectively, w_2) is λ in every rule of the second form.

Thus, the class of right-linear grammars exactly coincides with REG. Also:

Theorem 2.10 *Every LLIN grammar generates an REG language.*

And now, one can pose the following question: do all linear grammars generate languages in $\mathcal{L}_3$? The answer is: no!, and a counterexample is next.

Example 2.11 *Consider the languages:*

$$L_1 = \{a^n b^n c^k : n \geq 1, k \geq 1\},$$
$$L_2 = \{a^k b^n c^n : n \geq 1, k \geq 1\}.$$

Both are generated by linear grammars. For instance, the first one by:

$$G = (\{S, A\}, \{a, b, c\}, S, P\}, P = \{S \longrightarrow Sc, S \longrightarrow Ac, A \longrightarrow ab, A \longrightarrow aAb\}.$$

However, could one build a regular grammar generating any of them? Since:

$$L_1 \cap L_2 = \{a^n b^n c^n : n \geq 1\} \notin CF$$

and it is known that CF ∩ REG = CF, one gets that neither L_1 nor L_2 can be REG.

Theorem 2.12 *Every language in $\mathcal{L}_3$ can be generated by a grammar having the following two types of productions: $X \longrightarrow aY, X \longrightarrow a$, with $X, Y \in N, a \in T$.*

An CF grammar $G = (N, T, S, P)$ is said to be **self-embedding** if there exists an $A \in N$ such that $A \Longrightarrow^* XAY$, for some $X, Y \in (N \cup T)^+$.

Theorem 2.13 *If an CF grammar is reduced and not self-embedding, then $L(G) \in \mathcal{L}_3$.*

Thus, self-embedding is the very characteristic feature of CF languages which separates them from smaller language classes. Since that feature (it is the case with relative sentences, for instance) does appear in natural languages (being the source for its recursiveness leading to the infinite set of sentences which a natural language is), it is obvious that a natural language cannot be smaller than an CF language. This discussion will continue later.

An CF grammar is said to be in **Greibach normal form** (GNF) if each rule is of the form:

$$A \longrightarrow aX, A \in N, a \in T, X \in N^*.$$

Theorem 2.14 *For every λ-free CF grammar, one can find an equivalent grammar in GNF.*

Given a finite alphabet V, a **regular expression** is inductively defined as follows:

(i) λ is a regular expression,

(ii) for every $a \in V$, a is a regular expression,

(iii) if R is a regular expression, then so is R^*,

(iv) if Q, R are regular expressions, then so are QR and $Q \cup R$.

Every regular expression denotes an REG language. For example:

λ denotes $\{\lambda\}$,

a denotes $\{a\}$,

$a \cup b$ denotes $\{a, b\}$,

ab denotes $\{ab\}$,

a^* denotes $\{a\}^*$,

$(a \cup b)^*$ denotes $\{a, b\}^*$,

$(a \cup b)a^*$ denotes $\{a, b\}a^* = aa^* \cup ba^* = \{aa^*, ba^*\}$.

The following question now arises: is every REG language describable by means of a regular expression? The answer is simply yes!

Theorem 2.15 *Every regular expression denotes a language in $\mathcal{L}_3$ and, conversely, every language in $\mathcal{L}_3$ is denoted by a regular expression.*

A short list of valid equations for all regular expressions P, Q, R includes:

$P \cup (Q \cup R) = (P \cup Q) \cup R$,

$P(QR) = (PQ)R$,

$P \cup Q = Q \cup P$,

$P(Q \cup R) = PQ \cup PR$,

$(P \cup Q)R = PR \cup QR$,

$P^* = \lambda \cup PP^*$,

$\lambda P = P\lambda = P$,

$P^* = (\lambda \cup P)^*$.

The concept of a regular expression suggests the operation on languages called substitution.

Given a finite alphabet V, let V_a denote an alphabet and $s(a) \subseteq V_a^*$ a language for each $a \in V$. For each string $w = a_1a_2 \ldots a_n \in V^*$, one defines the **substitution**:

$$s(w) = s(a_1)s(a_2) \ldots s(a_n)$$

as the concatenation of the languages corresponding to the letters of w. This is extended to any $L \subseteq V^*$ by:

$$s(L) = \{v : v \in s(w), \text{ for some } w \in L\}.$$

The family $\mathcal{L}_3$ is closed under substitution, i.e. the set of regular expressions is closed under substitution of a regular expression for each of its letters. Substitution can be regarded as the generalization of the notion of morphism.

As for CF languages, for both families LIN and REG there are necessary conditions in the form of pumping lemmata.

Pumping lemma for linear languages:

for every $L \in LIN$, there exist $p, q \in N$ such that for every $z \in L$, if $|z| > p$ then $z = uvwxy$, where $u, v, w, x, y \in T^*$, $|uvxy| \leq q, vx \neq \lambda$, and $uv^iwx^iy \in L$, for every $i \geq 0$.

Pumping lemma for regular languages:

for every $L \in REG$, there exist $p, q \in N$ such that for every $z \in L$, if $|z| > p$ then $z = uvw$, where $u, v, w \in T^*$, $|uv| \leq q, v \neq \lambda$, and $uv^iw \in L$, for every $i \geq 0$.

2.5 Semilinear, Context-Sensitive and Mildly Context-Sensitive Languages

Whether or not natural languages are context-free sets of sentences was a much discussed issue in the eighties. To enter the discussion's core, cfr. the following papers: Gazdar (1981), Bresnan, Kaplan, Peters and Zaenen (1982), Pullum and Gazdar (1982), Culy (1985), and Shieber (1985) (all of them were collected in Savitch, Bach, Marsh and Safran-Naveh 1987).

Today, there is a relative agreement that natural languages are not context-free. However, how large they are continues to be a not so simple matter. There are two main noncompatible options. A natural language:

(i) either forms a class of sentences that includes the context-free family but is larger than it (so still comfortably placed within the Chomsky hierarchy),

(ii) or occupies a position eccentric with respect to that hierarchy, in such a way that it does not contain any whole family in the hierarchy but is spread along all of them.

Following the first alternative gave origin to a new family of languages, which is of a clear linguistic interest.

A family of **mildly context-sensitive languages** (MCS) is a family $\mathcal{L}$ such that:

(i) each language in $\mathcal{L}$ is semilinear,

(ii) for each language in $\mathcal{L}$, the membership problem (whether or not a string belongs to the language) is solvable in deterministic polynomial time,

(iii) $\mathcal{L}$ contains the following three nonCF languages:

- $L = \{a^n b^n c^n : n \geq 0\}$: multiple agreements,
- $L = \{a^n b^m c^n d^m : n, m \geq 0\}$: crossed dependencies,
- $L = \{ww : w \in \{a, b\}^*\}$: duplications.

MCS is a linguistically-motivated family, as both it contains the above three languages, which are more or less agreed to represent structures that exist in natural languages, and enjoys good complexity conditions (i.e. fast processing), as stated by the deterministic polynomial time requirement.

In order to see what a semilinear language is, let us assume $V = \{a_1, a_2, \ldots, a_k\}$. Being $\mathbb{N}$ the set of integers, the **Parikh mapping** of a string is:

$$\Psi : V^* \longrightarrow \mathbb{N}^k$$
$$\Psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_k}), w \in V^*.$$

Given a language, its **Parikh set** is:

$$\Psi(L) = \{\Psi(w) : w \in L\}.$$

A **linear set** is a set $M \subseteq \mathbb{N}^k$ such that:

$$M = \{v_0 + \sum_{i=1}^{m} v_i x_i : x_i \in \mathbb{N}, \text{ for some } v_0, v_1, \ldots, v_m \in \mathbb{N}^k\}.$$

A **semilinear set** is a finite union of linear sets. A **semilinear language** is an L such that $\Psi(L)$ is a semilinear set.

A phrase-structure grammar is called **length-increasing** if, for every production $w \longrightarrow v \in P$, one has $|w| \leq |v|$. This is clear for every CS grammar. Moreover:

Theorem 2.16 *Every length-increasing grammar generates an CS language.*

The length-increasing property is, therefore, equivalent to context-sensitivity with the sole exception of the rule $S \longrightarrow \lambda$, which is needed only to derive λ.

A length-increasing grammar is said to be in **Kuroda normal form** (KNF) if each of its productions is of any of the following forms:

- $A \longrightarrow a$,
- $A \longrightarrow B$,
- $A \longrightarrow BC$,
- $AB \longrightarrow CD$,

  for A, B, C, D nonterminals and a terminal.

Theorem 2.17 *For every length-increasing grammar, one can find an equivalent grammar in KNF.*

Corollary 2.18 *Every λ-free CS language can be generated by a grammar in KNF.*

A λ-free CS grammar is said to be in **Penttonen** or **one-sided normal form** (PNF) if each of its productions is of any of the following forms:

- $A \longrightarrow a$,
- $A \longrightarrow B$,
- $A \longrightarrow BC$,
- $AB \longrightarrow AC$,
- $AB \longrightarrow BA$.

An example of an CS grammar generating a nonCF language is the following:

Example 2.19 *Let G = (N, T, S, P) be a grammar such that:*

$N = \{S, A_1, A_2, B_1, B_2, C_1, C_2\}$,

$T = \{a, b, c\}$,

$P = \{S \longrightarrow A_1B_1C_1, {}^*A_1 \longrightarrow aA_2B_2, B_2B_1 \longrightarrow B_2B_2, B_2C_1 \longrightarrow B_2C_2c,$
${}^*A_2 \longrightarrow aA_1B_1, B_1B_2 \longrightarrow B_1B_1, B_1C_2 \longrightarrow B_1C_1c, A_1 \longrightarrow a, B_1 \longrightarrow b, C_1 \longrightarrow$
$c, A_2 \longrightarrow a, B_2 \longrightarrow b, C_2 \longrightarrow c\}$.

The generated language is:

$$L(G) = \{a^n b^n c^n : n \geq 1\} \notin CF.$$

*As is easily seen, every application of the rules marked with * sends a signal through the B's to C_1 or C_2 on their right, which may be killed on its way or reach the C's, where it deposits a c.*

3 Automata

3.1 Finite Automata

Grammars are generating devices which may simulate the productive (i.e. speaking) behaviour of speakers/hearers. Automata are recognizing devices which may simulate the receptive (i.e. hearing) behaviour of them. Each class of mechanisms serves to model one of the two aspects of human linguistic capacity. As well, there are surprising, strong formal connections between grammar theory and automata theory. Let us see.

A **finite automaton** (FA) is a construct:

$$A = (Q, T, M, q_0, F),$$

with:

- Q a finite nonempty set of states,
- T a finite alphabet of input letters,
- M a transition function: $Q \times T \to Q$,
- $q_0 \in Q$ the initial state,
- $F \subseteq Q$ the set of final (accepting) states.

A **accepts** or **recognizes** a string if it reads until the last letter of it and enters a final state.

Example 3.1

$$A = (Q, T, M, q_0, F) :$$

$Q = \{q_0, q_1, q_2, q_3\},$

$T = \{a, b\},$

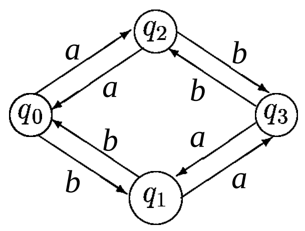
$F = \{q_0\},$

$M(q_0, a) = q_2 \quad , \quad M(q_0, b) = q_1,$

$$M(q_1, a) = q_3 \quad , \quad M(q_1, b) = q_0,$$
$$M(q_2, a) = q_0 \quad , \quad M(q_2, b) = q_3,$$
$$M(q_3, a) = q_1 \quad , \quad M(q_3, b) = q_2.$$

The **transition table** *and the* **transition graph** *for A are, respectively:*

M	a	b
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2



One can check that $L(A) = \{w \in \{a, b\}^* : |w|_a$ *is even,* $|w|_b$ *is even* $\}$.

If M is a one-valued function, the finite automaton is called **deterministic** (DFA). Otherwise, it is called **nondeterministic** (NFA). In the first case, M contains exactly one transition with the same left-hand side. Notice that the definition of M does not require M to be a total function, i.e. M may well be not defined for some combinations of a state and a letter.

The symbols $\vdash$ and $\vdash^*$ for transitions are, respectively, equivalent to the symbols $\Longrightarrow$ and $\Longrightarrow^*$ for derivations in grammars.

The **language** accepted by a finite automaton is:

$$L(A) = \{w \in T^* : q_0 w \vdash^* p, p \in F\}.$$

Take notice that $\lambda \in L(A)$ if and only if $q_0 \cap F \neq \emptyset$.

Theorem 3.2 *For every NFA, one can find an equivalent REG grammar.*

Theorem 3.3 *For every REG grammar, one can find an equivalent NFA.*

Corollary 3.4 $\mathcal{L}_3$ *coincides with the family of languages accepted by NFA's.*

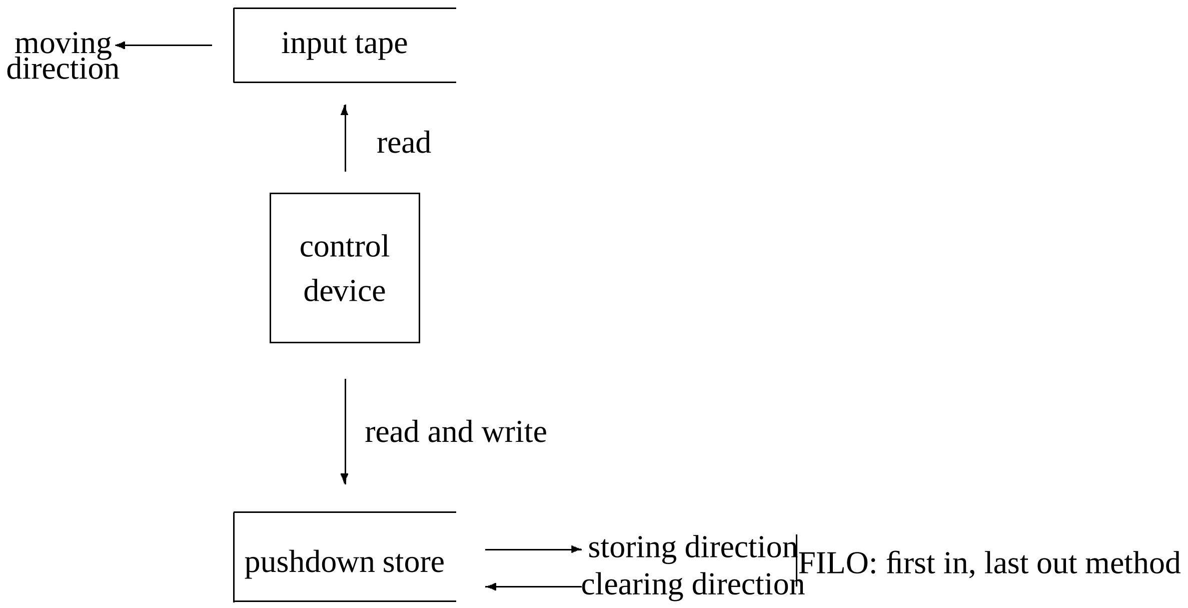
The following question arises now: is there some language in $\mathcal{L}_3$ that cannot be accepted by any DFA? The answer is simply no! Consequently, one can always simulate the behaviour of an NFA by means of an DFA (with more states).

A number of important consequences for the languages in $\mathcal{L}_3$ follow from the concept of an FA, among others:

- $\mathcal{L}_3$ is a Boolean algebra,
- it is decidable whether two REG grammars are equivalent, etc.

3.2 Pushdown Automata

Let us introduce now a new element in the definition of an automaton: memory. Pushdown automata result.



A **pushdown automaton** (PDA) is a construct $A = (Z, Q, T, M, z_0, q_0, F)$, with:

- Z a finite alphabet of pushdown letters,
- Q a finite set of internal states,
- T a finite set of input letters,
- M the transition function:

$$Z \times Q \times (T \cup \#) \longrightarrow \mathcal{P}_{fin}(Z^* \times Q),$$

- $z_0 \in Z$ the initial letter,
- $q_0 \in Q$ the initial state,
- $F \subseteq Q$ a set of final or accepting states.

A **configuration** of an PDA is a string zq, where $z \in Z^*$ is the current contents of the pushdown store and $q \in Q$ is the present state of the control device.

A **nondeterministic pushdown automaton** (NPDA) may reach a finite number of different new configurations from one configuration in one move:

$$M(z, q, a) = \{(w_1, p_1), (w_2, p_2), \ldots, (w_m, p_m)\}, a \in T, w_i \in Z^*, p_i \in Q, 1 \leq i \leq m.$$

There may be λ-moves too, which make it possible for the PDA to change its configuration without reading any input.

For a string to be accepted, the three following conditions must hold:

(i) the control device read the whole string,

(ii) the PDA reached a final state,

(iii) the pushdown store is empty.

Note that only the existence of at least one sequence of moves leading to an accepting configuration is required, while others may lead to nonaccepting ones.

Example 3.5 *To accept $L = \{a^n b^n : n \geq 1\}$, the following PDA is adequate:*

$$A = (\{z_0, a\}, \{q_0, q_1, q_2\}, \{a, b\}, M, z_0, q_0, \{q_2\}),$$

with:

M	a	b	$\#$
(z_0, q_0)	$(z_0 a, q_0)$	$\emptyset$	$\emptyset$
(a, q_0)	(aa, q_0)	(λ, q_1)	$\emptyset$
(z_0, q_1)	$\emptyset$	$\emptyset$	(λ, q_2)
(a, q_1)	$\emptyset$	(λ, q_1)	$\emptyset$
(z_0, q_2)	$\emptyset$	$\emptyset$	$\emptyset$
(a, q_2)	$\emptyset$	$\emptyset$	$\emptyset$

An PDA $A = (Z, Q, T, M, z_0, q_0, F)$ is said to be **deterministic** (DPDA) if and only if for every $(z, q) \in Z \times Q$:

(i) either $M(z, q, a)$ contains exactly one element, for every $a \in T$, while $M(z, q, \lambda) = \emptyset$,

(ii) or $M(z, q, \lambda)$ contains exactly one element, while $M(z, q, a) = \emptyset$, for every $a \in T$.

Theorem 3.6 *The family of languages accepted by DPDA's is strictly contained in the family of languages accepted by NPDA's.*

To illustrate this, let us observe the following:

Example 3.7

$$L_1 = \{wcw^{-1} : w \in \{a, b\}^*\},$$
$$L_2 = \{ww^{-1} : w \in \{a, b\}^*\}.$$

While L_1 belongs to both NPDA and DPDA, L_2 belongs only to NPDA.

The following results establish the relationship of $NPDA's$ to CF languages.

Theorem 3.8 *NPDA = CF.*

Theorem 3.9 *For every CF grammar, an algorithm exists to transform it into an equivalent NPDA.*

Theorem 3.10 *For every PDA, an algorithm exists to transform it into an equivalent CF grammar.*

Turing machines are the most powerful recognizing devices, and are able to recognize any RE language. They are the foundation of computation theory and involve a lot of complexities which cannot be addressed here.

4 Regulated and Parallel Rewriting

4.1 A Sample of Regulated Rewriting

Next, a few important types of regulated grammars are presented, without being exhaustive at all. In order to have a regulated (i.e. controlled) grammar, one more or less modifies/restricts the notion of a grammar, and as a consequence one usually gets either a different (often greater) generative capacity or a simpler method of generation.

A **matrix grammar** is:

$$G_M = (N, T, S, M),$$

where M is a finite set of finite nonempty sequences (**matrices**) of the form:

$$m : [r_1, r_2, \ldots, r_n], n \geq 1,$$

with:

$$r_i : \alpha_i \longrightarrow \beta_i,\ \alpha_i \in (N \cup T)^* N (N \cup T)^*,\ \beta_i \in (N \cup T)^*.$$

A derivation in a matrix grammar is as follows:

for every $x, y \in (N \cup T)^*, x \Longrightarrow_{G_M} y$ if and only if there exist $x_0, x_1, \ldots, x_n \in (N \cup T)^*$ and there exist $[r_1, r_2, \ldots, r_n] \in M, r_i : \alpha_i \longrightarrow \beta_i, 1 \leq i \leq n,$ such that $x_{i-1} = x'_{i-1}\alpha_i x''_{i-1}$ and $x_i = x'_{i-1}\beta_i x''_{i-1},$ for some $x'_{i-1}, x''_{i-1} \in (N \cup T)^*, 0 \leq i \leq n-1.$

Example 4.1 *The grammar G_M with the following matrices:*

$$m_1 : [S \longrightarrow ABC],$$

$$m_2 : [A \longrightarrow aA, B \longrightarrow bB, C \longrightarrow cC],$$

$$m_3 : [A \longrightarrow a, B \longrightarrow b, C \longrightarrow c],$$

yield $L(G_M) = \{a^n b^n c^n : n \geq 1\}$.

A **programmed grammar** is:

$$G_P = (N, T, S, P),$$

where P is a finite set of triples $(r : \alpha \longrightarrow \beta, \sigma(r), \phi(r))$, r is a label, $r : \alpha \longrightarrow \beta \in (N \cup T)^*$ and $\sigma(r)$, $\phi(r)$ are two sets of labels of rules.

An immediate derivation in a programmed grammar is as follows:

$$\text{for every}(x, r_1), (y, r_2) \in (N \cup T)^* \times Lab(P) : (x, r_1) \Longrightarrow_{G_P} (y, r_2) \text{ if and only if } :$$

(i) either $x = x_1 \alpha x_2$ and $y = x_1 \beta x_2$ $(x_1, x_2 \in (N \cup T)^*)$ and $(r_1 : \alpha \longrightarrow \beta, \sigma(r_1), \phi(r_1)) \in P)$ and $r_2 \in \sigma(r_1)$,

(ii) or $x = y$ and $r_1 : \alpha \longrightarrow \beta$ (for $(r_1 : \alpha \longrightarrow \beta, \sigma(r_1), \phi(r_1)) \in P)$ cannot be applied to x and $r_2 \in \phi(r_1)$.

The language generated by a programmed grammar is:

$$L(G_P) = \{x : x \in T^* \text{and there exist} r_1, r_2 \in Lab(P) \text{such that} (S, r_1) \Longrightarrow^*_{G_P} (x, r_2)\}.$$

Example 4.2 *The grammar G_P with the following productions:*

$$(r_1 : S \longrightarrow AA, \{r_1\}, \{r_2, r_3\}),$$

$$(r_2 : A \longrightarrow S, \{r_2\}, \{r_1\}),$$

$$(r_3 : A \longrightarrow a, \{r_3\}, \emptyset),$$

yield $L(G_P) = \{a^{2^n} : n \geq 1\}$.

A **random context grammar** is:

$$G_{RC} = (N, T, S, P),$$

where P is a finite set of rules of the form $(\alpha \longrightarrow \beta, Q, R)$, with $\alpha \longrightarrow \beta \in (N \cup T)^*$, $Q \subseteq N$, $R \subseteq N$.

An immediate derivation in a random context grammar is as follows:

for every $x, y \in (N \cup T)^* : x \Longrightarrow_{G_{RC}} y$ if and only if $x = x'\alpha x''$ and $y = x'\beta x''(x', x'' \in (N \cup T)^*)$ and $(\alpha \longrightarrow \beta, Q, R) \in P$ and for every $u \in Q : u \in x'x''$ and for every $v \in R : v \notin x'x''$.

Example 4.3 *The grammar G_{RC} with the following productions:*

$$(S \longrightarrow AA, \emptyset, \{B, D\}),$$
$$(A \longrightarrow B, \emptyset, \{S, D\}),$$
$$(B \longrightarrow S, \emptyset, \{A, D\}),$$
$$(A \longrightarrow D, \emptyset, \{S, B\}),$$
$$(D \longrightarrow a, \emptyset, \{S, A, B\}),$$

yield $L(G_{RC}) = \{a^{2^n} : n \geq 1\}$.

A **grammar with regular control** is:

$$G_{REG} = (N, T, S, P, R),$$

where R is a regular language over P.

The language generated by an G_{REG} consists of the strings resulting from a derivation:

$$S \Longrightarrow^{p_1}_{G_{REG}} w_1 \Longrightarrow^{p_2}_{G_{REG}} w_2 \Longrightarrow_{G_{REG}} \ldots \Longrightarrow^{p_n}_{G_{REG}} w_n = w \in T^*$$

such that $p_1 p_2 \ldots p_n \in R$.

Example 4.4 *The grammar G_{REG} consisting of:*

$$N = \{S, A, B\},$$
$$T = \{a, b, c\},$$
$$P = \{p_1 : S \longrightarrow AB, p_2 : A \longrightarrow aAb, p_3 : B \longrightarrow Bc, p_4 : A \longrightarrow ab, p_5 : B \longrightarrow c\},$$

$R = \{p_1\}\{p_2p_3\}^*\{p_4p_5\}$,

generates:

$$L(G_{REG}) = \{a^nb^nc^n : n \geq 1\}.$$

An **additive valence grammar** is:

$$G_{AV} = (N, T, S, P, v),$$

where $v : P \longrightarrow \mathbb{Z}(\mathbb{Z}$ is the set of integers).

The language generated by an G_{AV} is:

$$L(G_{AV}) = \{w : w \in T^* \text{ and } S \Longrightarrow^{p_1}_{G_{AV}} w_1 \Longrightarrow^{p_2}_{G_{AV}} w_2 \Longrightarrow_{G_{AV}} \ldots \Longrightarrow^{p_n}_{G_{AV}} w_n = w \text{ and } v(p_1) + v(p_2) + \ldots + v(p_n) = 0\}.$$

Example 4.5 *The grammar G_{AV} consisting of:*

$N = \{S, A, B\}$,

$T = \{a, b, c\}$,

$P = \{p_1 : S \longrightarrow AB, p_2 : A \longrightarrow aA, p_3 : B \longrightarrow bBc, p_4 : A \longrightarrow a, p_5 : B \longrightarrow bc\}$,

$v(p_1) = v(p_4) = v(p_5) = 0$, $v(p_2) = 1$, $v(p_3) = -1$,

generates:

$$L(G_{AV}) = \{a^nb^nc^n : n \geq 1\}.$$

Notice that the rules p_2 and p_3 must be applied the same number of times.

A **multiplicative valence grammar** is:

$$G_{MV} = (N, T, S, P, v),$$

where $v : P \longrightarrow Q^+$ (Q is the set of rational numbers).

The language generated by an G_{MV} is:

$$L(G_{MV}) = \{w : w \in T^* \text{ and } S \Longrightarrow^{p_1}_{G_{MV}} w_1 \Longrightarrow^{p_2}_{G_{MV}} w_2 \Longrightarrow_{G_{MV}} \ldots \Longrightarrow^{p_n}_{G_{MV}} w_n = w \text{ and } v(p_1)v(p_2) \ldots v(p_n) = 1\}.$$

Example 4.6 *As G_{MV}, take G_{AV} in the last example, with the following specific valence mapping:*

$$v(p_1) = v(p_4) = v(p_5) = 1,$$

$$v(p_2) = 2,$$

$$v(p_3) = 1/2.$$

It generates:

$$L(G_{MV}) = \{a^n b^n c^n : n \geq 1\}.$$

An **ordered grammar** is:

$$G_O = (N, T, S, P, <),$$

where $<$ is a strict partial order (i.e. irreflexive, asymmetric and transitive) over P.

An immediate derivation in an ordered grammar $x \Longrightarrow_{G_O} y$ holds if and only if:

(i) there exist $x_1, x_2 \in (N \cup T)^*$ such that $x = x_1 w x_2$ and $y = x_1 z x_2$,

(ii) $w \longrightarrow z \in P$ and there does not exist any substring w' of x such that there exists z' such that $w' \longrightarrow z' \in P$ and $w' \longrightarrow z' > w \longrightarrow z$.

Thus, the production that is utilized at each step is maximal in the ordered set of rules.

Example 4.7 *The ordered grammar:*

$$G_O = (\{S, A, B, A', B', A'', M\}, \{a, b, c\}, S, P, <),$$

where $P = \{(1)A'' \longrightarrow M, (2)B \longrightarrow bc, (3)A' \longrightarrow M, (4)B' \longrightarrow B, (5)A \longrightarrow M, (6)B \longrightarrow bB'c, (7)B' \longrightarrow M, (8)A \longrightarrow A'', (9)A \longrightarrow aA', (10)B \longrightarrow M, (11)A'' \longrightarrow a, (12)A' \longrightarrow A, (13)S \longrightarrow AB\}$ *and the order relation is satisfied by the following pairs:*

$$(2) < (3), (2) < (5), (4) < (3), (6) < (1), (6) < (5), (8) < (7), (9) < (7), (11) < (10),$$
$$(12) < (10),$$

generates:

$$L(G_O) = \{a^n b^n c^n : n \geq 1\}.$$

Generally speaking, there exist two main types of grammar derivations:

- sequential: as is the case with grammars in the Chomsky hierarchy as well as all other ones presented in this chapter so far;
- parallel: which appear in several mechanisms, particularly in:
  - (i) Indian parallel grammars: at each step of the derivation, every occurrence of one letter is rewritten (by using the same production),
  - (ii) Lindenmayer systems: at each step of the derivation, all occurrences of all letters are rewritten (using different productions for different occurrences of one letter is allowed: see below for details).

An **Indian parallel grammar** is a construct:

$$G_{IP} = (N, T, S, P).$$

The immediate derivation runs as follows:

for every $x \in (N \cup T)^+$, for every $y \in (N \cup T)^*$: $x \Longrightarrow_{G_{IP}} y$ if and only if:

(i) $x = x_1 A x_2 A \ldots x_n A x_{n+1}$, $A \in N$, $x_i \in ((N \cup T) - \{A\})^*$, $1 \leq i \leq n+1$,

(ii) $y = x_1 w x_2 w \ldots x_n w x_{n+1}$,

(iii) $A \longrightarrow w \in P$.

Example 4.8 *The Indian parallel grammar:*

$$G_{IP} = (\{S\}, \{a\}, S, \{S \longrightarrow SS, S \longrightarrow a\})$$

yields $L(G_{IP}) = \{a^{2^n} : n \geq 0\}$.

A **Russian parallel grammar** is a construct:

$$G_{RP} = (N, T, S, P),$$

where $P = P_1 \cup P_2$, $P_1 \cap P_2 = \emptyset$.

An immediate derivation in a Russian parallel grammar is:

$$x \Longrightarrow_{G_{RP}} y \text{ if and only if:}$$

(i) either $x = x_1 A x_2$ and $y = x_1 w x_2$ $(x_1, x_2 \in (N \cup T)^*)$ and $A \longrightarrow w \in P_1$,

(ii) or $x = x_1 A x_2 A \ldots x_n A x_{n+1}$ and $y = x_1 w x_2 w \ldots x_n w x_{n+1}$ $(x_i \in ((N \cup T) - \{A\})^*$, $1 \leq i \leq n+1)$ and $A \longrightarrow w \in P_2$.

If $P_1 = \emptyset$, one gets an G_{IP}. If $P_2 = \emptyset$, one gets an CF.

4.2 Lindenmayer Systems

The motivation behind Lindenmayer systems (L systems for short) is biological. They are intended to model the (parallel) growth of living beings.

An **interactionless Lindenmayer system** ($0L$) is a context-free pure (without a nonterminal alphabet) grammar with parallel derivations:

$$G = (V, w, P),$$

where V is an alphabet, $w \in V^*$ is an axiom and P is a finite set of rules of the form $a \to v$, with $a \in V$ and $v \in V^*$ such that for each $a \in V$ there is at least one rule $a \to v$ in P (P is said to be complete).

Given $w_1, w_2 \in V^*$, one writes $w_1 \Longrightarrow w_2$ if and only if $w_1 = a_1a_2 \ldots a_n$ and $w_2 = v_1v_2 \ldots v_n$, for $a_i \to v_i \in P, 1 \leq i \leq n$. The generated language is:

$$L(G) = \{x \in V^* : w \Longrightarrow^* x\}.$$

There are several important variants of L systems:

- if for each rule $a \to v \in P$ one has $v \neq \lambda$, then G is **propagating** (nonerasing);
- if for each $a \in V$ there is only one rule $a \to v \in P$, then G is **deterministic**;
- if a subset T of V is distinguished and $L(G)$ is defined as the set $\{x \in T^* : w \Longrightarrow^* x\}$, then G is **extended**.

Regarding the generative power of L systems, many results are known, among others the following ones:

- The family of deterministic $0L$ languages is incomparable with FIN, REG, LIN, CF (FIN is the family of finite languages).
- CF is a strict subset of the family of extended $0L$ languages.
- All L languages are contained in CS.

A remarkable feature of a deterministic $0L$ system G is that it generates its language in a sequence $L(G) = w = w_0, w_1, w_2, \ldots$ such that $w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots$ Thus, one can define the **growth function** of G, denoted $growth_G : \mathbb{N} \longrightarrow$, by:

$$growth_G(n) = |w_n|, n \geq 0.$$

5 Nonstandard Generative Mechanisms

5.1 Contextual Grammars

A **contextual grammar** is a construct:

$$G = (V, B, (S_1, C_1), (S_2, C_2), \ldots, (S_n, C_n)),$$

consisting of:

- V an alphabet,
- $B \subseteq V^*$ a finite set: the base or the set of axioms,
- $S_i \subseteq V^*$ selectors,
- $C_i \subseteq V^* \times V^*$ contexts,
- (S_i, C_i) productions, $1 \leq i \leq n$.

In a contextual grammar, one considers two main types of immediate derivation:

- **external derivation**:
  for every $x, y \in V^* : x \Longrightarrow_{ex} y$ if and only if $y = uxw, x \in S_i$, $(u, v) \in C_i, 1 \leq i \leq n$.
- **internal derivation**:

$$\text{for every } x, y \in V^* : x \Longrightarrow_{in} y \text{ if and only if } x = x_1x_2x_3 \text{ and}$$
$$y = x_1ux_2vx_3, x_2 \in S_i, (u, v) \in C_i, 1 \leq i \leq n.$$

The language generated by an internal contextual grammar is:

$$L_{in}(G) = \{z \in V^* \text{such that there exists} w \in B \text{such that} w \Longrightarrow_{in}^* z\}.$$

Another way of defining $L_{in}(G)$ is to say that it is the smallest language L such that:

(i) $B \subseteq L$,

(ii) for every $x \in L$: if $x = x_1x_2x_3$ and $x_2 \in S_i$, for some i, $1 \leq i \leq n$, then $x_1ux_2vx_3 \in L$, for every $(u, v) \in C_i$.

If one introduces several different restrictions, the following natural variants of derivation arise:

- **minimal local derivation**:

$$\text{for every} x, y \in V^* : x \Longrightarrow_{ml} y \text{ if and only if } :$$

  (i) $x = x_1x_2x_3$,
  (ii) $y = x_1ux_2vx_3$ $(x_2 \in S_i, (u, v) \in C_i, 1 \le i \le n)$, and
  (iii) there do not exist $x'_1, x'_2, x'_3 \in V^*$ such that $x = x'_1x'_2x'_3$ and $x'_2 \in S_i$ and $|x'_1| \ge |x_1|$ and $|x'_3| \ge |x_3|$ and $|x'_2| < |x_2|$.

  A context is adjoined to a selector provided this is minimal (i.e. the shortest one) in such a position.

- **maximal local derivation**:

$$\text{for every} x, y \in V^* : x \Longrightarrow_{Ml} y \text{ if and only if } :$$

  (i) $x = x_1x_2x_3$,
  (ii) $y = x_1ux_2vx_3$ $(x_2 \in S_i, (u, v) \in C_i, 1 \le i \le n)$, and
  (iii) there do not exist $x'_1, x'_2, x'_3 \in V^*$ such that $x = x'_1x'_2x'_3$ and $x'_2 \in S_i$ and $|x'_1| \le |x_1|$ and $|x'_3| \le |x_3|$ and $|x'_2| > |x_2|$.

  A context is adjoined to a selector provided this is maximal (i.e. the longest one) in such a position.

- **minimal global derivation** ($\Longrightarrow_{mg}$): In the definition of $\Longrightarrow_{ml}$, one replaces $x'_2 \in S_i$ with $x'_2 \in S_j$, for every $j : 1 \le j \le n$. Note that the chosen selector has to be the shortest one among all the selectors.

- **maximal global derivation** ($\Longrightarrow_{Mg}$): In the definition of $\Longrightarrow_{Ml}$, the same substitution as above is introduced. Note that the chosen selector has to be the longest one among all the selectors.

Given $\alpha \in \{ml, mg, Ml, Mg\}$, the language generated by the contextual grammar is:

$$L_\alpha(G) = \{z \in V^* \text{such that there exist} w \in B \text{such that} w \Longrightarrow^*_\alpha z\}.$$

If all the languages in S_i belong to the same family F of languages in the Chomsky hierarchy, G is said to be a **contextual grammar with selection** of type F.

Example 5.1 *The contextual grammar:*

$$G = (\{a, b\}, \{abab\}, (ab^+a, \{(a, a)\}), (ba^+b, \{(b, b)\}))$$

generates:

$$L_{in}(G) = L_{mg}(G) = \{a^nb^ma^nb^m : n, m \ge 1\}.$$

Example 5.2 *The language:*

$$L = \{a^n cb^n a^m cb^m : n, m \geq 1\}$$

is generated by:

$$G_{in} = (\{a, b, c\}, \{cc\}, (c, \{(a, b)\})).$$

Example 5.3 *The language:*

$$L = \{a^+\} \cup \{a^n b^n : n \geq 1\}$$

is generated by:

$$G_{Mg} = (\{a, b\}, \{a, ab\}, (a, \{(\lambda, a)\}), (a^+ b, \{(a, b)\})).$$

Contextual grammars allow one to produce families of languages that are eccentric with respect to the Chomsky hierarchy, as shall be seen below. This seems to be a very relevant feature from a linguistic viewpoint, as natural languages could possibly occupy an eccentric position with regard to that hierarchy.

5.2 Grammar Systems

Grammar systems are complex, modular generating architectures intended for either increasing the generative power or decreasing the complexity of the generative strategy of the mechanism. Several types can be distinguished.

A **cooperating distributed grammar system** (CDGS) is a construct:

$$\Gamma = (N, T, S, P_1, P_2, \ldots, P_n),$$

where:

- $N \cap T = \emptyset$,
- $S \in N$,
- $P_1, P_2, \ldots, P_n$ are finite sets of rewriting rules: the components of the system.

Several modes of derivation can be considered (being $\mathbb{N}$ the set of integers):

- **in exactly k steps:** $\Longrightarrow_{p_i}^{=k}$ $(k \in \mathbb{N})$,

- **in at most k steps:** $\Longrightarrow_{p_i}^{\leq k}$,
- **in at least k steps:** $\Longrightarrow_{p_i}^{\geq k}$,
- **arbitrary derivation**: $\Longrightarrow_{p_i}^{*}$,
- **terminal** or **maximal derivation**: $\Longrightarrow_{p_i}^{t}$:

$x \Longrightarrow_{p_i}^{t} y$ if and only if $x \Longrightarrow_{p_i}^{\geq 1} y$ and there does not exist any $z \in (N \cup T)^*$ such that $y \Longrightarrow_{p_i} z$.

Given a mode of derivation $f \in D = \{*, t\} \cup \{\leq k, = k, \geq k : k \geq 1\}$, the language generated by an CDGS is:

$$L_f(\Gamma) = \{w \in T^* : S \Longrightarrow_{P_{i_1}}^{f} w_1 \Longrightarrow_{P_{i_2}}^{f} w_2 \Longrightarrow \ldots \Longrightarrow_{P_{i_m}}^{f} w_m = w, m \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq m\}.$$

Thus, five languages are associated with Γ.

Example 5.4 *The CDGS:*

$$\Gamma = (\{S, A, A', B, B'\}, \{a, b, c\}, S, P_1, P_2)$$

consisting of:

$$P_1 = \{S \longrightarrow S, S \longrightarrow AB, A' \longrightarrow A, B' \longrightarrow B\},$$
$$P_2 = \{A \longrightarrow aA'b, B \longrightarrow cB', A \longrightarrow ab, B \longrightarrow c\},$$

generates:

$$L_{=2}(\Gamma) = \{a^n b^n c^n, n \geq 1\},$$
$$L_{=k}(\Gamma) = L_{\geq k}(\Gamma) = \emptyset, \text{with} k \geq 3.$$

Example 5.5 *The CDGS:*

$$\Gamma = (\{S, A, A'\}, \{a, b\}, S, P_1, P_2, P_3)$$

consisting of:

$$P_1 = \{S \longrightarrow S, S \longrightarrow AA, A' \longrightarrow A\},$$
$$P_2 = \{A \longrightarrow aA', A \longrightarrow a\},$$

$P_3 = \{A \longrightarrow bA', A \longrightarrow b\}$,

generates:

$$L_{=2}(\Gamma) = L_{\geq 2}(\Gamma) = \{ww : w \in \{a, b\}^+\}.$$

Let $CD_n(f)$ denote the family of languages generated by CDGS's of degree (the number of components) at most n ($n \geq 1$) working in the mode f. If the number of components is not limited, one writes ∞ instead of n. The union of all families $CD_\infty(= k)$ (respectively, $CD_\infty(\geq k)$, $CD_\infty(\leq k)$), $k \geq 1$, is denoted by $CD_\infty(=)$ (respectively, $CD_\infty(\geq)$, $CD_\infty(\leq)$). If λ-rules are accepted, one writes $CD_n^\lambda(f)$, $CD_\infty^\lambda(f)$, etc. Many results on CDGS's' generative capacity are known:

- $CD_\infty(f) = CF$, for every $f \in \{*, t, = 1, \geq 1\} \cup \{\leq k : k \geq 1\}$.
- $CF = CD_1(f) \subset CD_2(f) \subseteq CD_r(f) \subseteq CD_\infty(f)$, for every $f \in \{= k, \geq k : k \geq 2\}$, $r \geq 3$.
- $CD_r(= k) \subseteq CD_r(= sk)$, for every $k, r, s \geq 1$.
- $CD_r(\geq k) \subseteq CD_r(\geq k + 1)$, for every $k, r \geq 1$.
- $CD_\infty(\geq) \subseteq CD_\infty(=)$.
- $CF = CD_1(t) = CD_2(t) \subset CD_3(t) = CD_\infty(t)$.
- All the six relations above are also true for CD^λ systems.

A **parallel communicating grammar system** (PCGS) is a construct:

$$\Gamma = (N, K, T, (P_1, S_1), (P_2, S_2), \ldots, (P_n, S_n)),$$

where:

- N, T, K are pairwise disjoint alphabets,
- $K = \{Q_1, Q_2, \ldots, Q_n\}$ are query letters (the subindex associates the letter to the corresponding component),
- $S_i \in N$,
- P_i are finite sets of productions over $N \cup K \cup T$, $1 \leq i \leq n$.

Given $V_\Gamma = (N \cup K \cup T)$, every n-uple $(x_1, x_2, \ldots, x_n)$, $x_i \in V_\Gamma^*$, is a **configuration** of the system.

Given two configurations $(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_n)$, $x_i, y_i \in V_\Gamma^*$, $1 \leq i \leq n$, one defines the **immediate derivation** $(x_1, x_2, \ldots, x_n) \Longrightarrow_\Gamma (y_1, y_2, \ldots, y_n)$ if and only if either of the following situations occur:

(i) either ($|x_i|_K = 0$) and (($x_i \Longrightarrow_{P_i} y_i$) or ($x_i \in T^*$ and $x_i = y_i$)), $1 \leq i \leq n$,

(ii) or there exists $i, 1 \leq i \leq n$ such that $|x_i|_K > 0$; in such a case, for every i one writes $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, $t \geq 1$, $z_j \in V_\Gamma^*$, $|z_j|_K = 0$, $1 \leq j \leq t+1$; if $|x_{i_j}|_K = 0$, $1 \leq j \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ [and $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$]; if, for some j, $1 \leq j \leq t$, $|x_{i_j}|_K \neq 0$, then $y_i = x_i$; for every i, $1 \leq i \leq n$, for which y_i is not specified above, one has $y_i = x_i$.

(i) is a rewriting step, (ii) is a communication step: the latter has a priority over the former (i.e. if both are possible at a certain stage of the derivation process, communication must be applied before).

The **blocking** of an PCGS may happen in either of the following cases:

- when one component x_i in $(x_1, ..., x_n)$ is not terminal but cannot be rewritten according to P_i, or
- when a circular query occurs: P_{i_1} introduces Q_{i_2}, P_{i_2} introduces $Q_{i_3}, \ldots, P_{i_{k-1}}$ introduces Q_{i_k}, and P_{i_k} introduces Q_{i_1} (communication has priority but it cannot take place, because strings to be communicated must contain no query letters at all).

The language generated by an PCGS is:

$$L(\Gamma) = \{x \in T^* : (S_1, S_2, \ldots, S_n) \Longrightarrow^* (x, \alpha_2, \ldots, \alpha_n), \alpha_i \in V_\Gamma^*, 2 \leq i \leq n\}.$$

Thus, the language of the system is the language of the **master** component, which is the first component of the system: S_1.

An PCGS is said to be **centralized** if and only if $P_i \subseteq (N \cup T)^* \times (N \cup T)^*$, $2 \leq i \leq n$ (i.e. only the master is allowed to introduce query letters). Otherwise, it is said to be **noncentralized**.

An PCGS is called **returning** if and only if, after communication, each component that has communicated goes back to its axiom and starts again. Otherwise, it is called **nonreturning**. An PCGS Γ will produce two languages, $L_r(\Gamma)$ and $L_{nr}(\Gamma)$, according to the returning or non-returning mode of working, respectively.

By default, one understands that an PCGS works in a noncentralized, returning mode. Some conventional notations include:

PC_nX: family of languages generated by noncentralized, returning PCGS's with at most n components of type X, $X \in \{REG, LIN, CF, CS, RE\}$,

CPC_nX: PC_nX+ centralized mode,

NPC_nX: PC_nX+ nonreturning mode,

$NCPC_nX$: CPC_nX+ nonreturning mode,

$PC_\infty X = \bigcup_{n\geq 1} PC_n X$ (analogously for $CPC_\infty X$, $NPC_\infty X$, $NCPC_\infty X$).

Example 5.6 *The PCGS:*

$$\Gamma = (\{S_1, S_2, S_3\}, K, \{a, b\}, (P_1, S_1), (P_2, S_2), (P_3, S_3))$$

consisting of:

$$P_1 = \{S_1 \longrightarrow aS_1, S_1 \longrightarrow a^3Q_2, S_2 \longrightarrow b^2Q_3, S_3 \longrightarrow c\},$$
$$P_2 = \{S_2 \longrightarrow bS_2\},$$
$$P_3 = \{S_3 \longrightarrow cS_3\},$$

generates:

$$L_r(\Gamma) = L_{nr}(\Gamma) = \{a^n b^n c^n : n \geq 1\}.$$

Example 5.7 *The PCGS:*

$$\Gamma = (\{S_1, S_2\}, K, \{a, b\}, (P_1, S_1), (P_2, S_2))$$

consisting of:

$$P_1 = \{S_1 \longrightarrow S_1, S_1 \longrightarrow Q_2Q_2\},$$
$$P_2 = \{S_2 \longrightarrow aS_2, S_2 \longrightarrow bS_2, S_2 \longrightarrow a, S_2 \longrightarrow b\},$$

generates:

$$L_r(\Gamma) = L_{nr}(\Gamma) = \{ww : w \in \{a, b\}^+\}.$$

Let $\Pi = \{PC, CPC, NPC, NCPC\}$. The following results about PCGS's' generative power are known:

- $Y_nCS^\lambda = RE$, for every n, for every $Y \in \Pi$, where CS^λ indicates that productions of arbitrary type are utilized.
- $Y_nREG - LIN \neq \emptyset$, $Y_nLIN - CF \neq \emptyset$ $(n \geq 2)$, and $Y_nREG - CF \neq \emptyset$ $(n \geq 3)$, for every $Y \in \Pi$.
- $Y_nREG - CF \neq \emptyset$, $n \geq 2$, for every $Y \in \{NPC, NCPC\}$.

- $LIN - (CPC_\infty REG \cup NCPC_\infty REG) \neq \emptyset$.
- $CPC_2REG \subset CF$, $PC_2REG \subseteq CF$.
- $CPC_\infty REG$ contains only semilinear languages.
- CPC_2CF contains nonsemilinear languages.
- If $L \subseteq V^*$, $L \in CPC_nREG$, then there exists a constant q such that for every $z \in L$, if $|z| > q$ then $z = x_1y_1x_2y_2 \dots x_my_mx_{m+1}$, $1 \leq m \leq n$, $y_i \neq \lambda$, $1 \leq i \leq m$, and for every $k \geq 1 : x_1y_1^kx_2y_2^k \dots x_my_m^kx_{m+1} \in L$.
- $CPC_nREG \subset CPC_{n+1}REG$, $n \geq 1$.
- $PC_nREG \subset PC_{n+1}REG$, $n \geq 1$.
- $CPC_nREG \subset CPC_nLIN \subset CPC_nCF$, $n \geq 1$.
- $NCPC_\infty CF \subseteq PC_\infty CF$ (i.e. a centralized + nonreturning working mode is weaker than a noncentralized + returning one).
- $CS = Y_nCS = Y_\infty CS$, $n \geq 1$, for every $Y \in \{CPC, NCPC\}$ (i.e. by using centralized CS PCGS's, one does not go beyond the family CS).
- $CS = PC_1CS = PC_2CS \subset PC_3CS = PC_\infty CS = RE$ (noncentralized returning PCGS's with three CS components suffice to generate every RE).
- $CS = NPC_1CS \subset NPC_2CS = NPC_\infty CS = RE$ (noncentralized nonreturning PCGS's with just two CS components are enough to generate every RE).
- Given X_nCS^λ, $n \geq 1$, $X \in \{PC, CPC, NPC, NCPC\} : CS = X_1CS^\lambda \subset X_2CS^\lambda = X_nCS^\lambda = RE$, for every $n \geq 2$.

5.3 Grammar Ecosystems

The postulates behind the notion of a grammar ecosystem are the following:

a) An ecosystem consists of an **environment** and some **agents**. The state of the environment as well as the states of the agents are all described by means of strings of letters from certain alphabets.

b) There exists one universal clock for the evolution of both the environment and the agents.

c) Both the environment and the agents have specific evolution rules, which are Lindenmayer rules (i.e. they are applied in parallel to all the letters describing the states of the environment and of the agents).

d) The rules for the evolution of the environment are independent from the agents and from the state of the environment. The rules for the evolution of the agents depend on the state of the environment in such a way that, at each step of the derivation, a specific appropriate subset of rules is selected from the set of rules of each agent.

e) The agents act on the environment through action rules, which are sequential (i.e. non-parallel) rewriting rules. The selection of the specific action rule to be applied at each step depends on the current state of the agent.

f) Agents' action on the environment has a priority over the environment's evolution. At each step, the rules for the evolution of the environment rewrite in a parallel way only the environment letters that are not affected by agent's actions.

A **grammar ecosystem** is a construct:

$$\Sigma = (E, A_1, A_2, \ldots, A_n)$$

consisting of:

- $E = (V_E, P_E)$, with:
  (i) V_E a finite alphabet,
  (ii) P_E a finite set of 0L rules over V_E (evolution rules of E).
- $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i)$, $1 \leq i \leq n$, with:
  (i) V_i a finite alphabet,
  (ii) P_i a finite set of 0L rules over V_i (evolution rules of the agent i),
  (iii) R_i a finite set of rules $x \to y$, with $x \in V_E^+$, $y \in V_E^*$ (action rules of the agent i),
  (iv) $\varphi_i : V_E^* \longrightarrow \mathcal{P}(P_i)$ (it selects the rules for the current evolution of the agent i),
  (v) $\psi_i : V_i^+ \longrightarrow \mathcal{P}(R_i)$, $1 \leq i \leq n$ (it selects the rules for the current action of A_i on the environment).

A grammar ecosystem works by modifying the strings representing the agents as well as the environment.

A **state** of a grammar ecosystem is:

$$\sigma = (w_E, w_1, w_2, \ldots, w_n),$$

where:

- $w_E \in V_E^*$,
- $w_i \in V_i^*$, $1 \leq i \leq n$.

Given $\sigma = (w_E, w_1, w_2, \ldots, w_n)$, A_i is an **active agent** in σ if and only if $\psi_i(w_i) \neq \emptyset$. An **action** of A_i in σ is an application of $r \in \psi_i(w_i)$ on w_E. A **simultaneous action** of the agents $A_{i_1}, A_{i_2}, \ldots, A_{i_r}$, $\{i_1, i_2, \ldots, i_r\} \subseteq \{1, 2, \ldots, n\}$, on the environment is a 1-step parallel derivation $w_E \Longrightarrow_\Sigma w'_E$ such that:

(i) $w_E = x_1u_1x_2u_2 \dots u_rx_{r+1}$,

(ii) $w'_E = x_1v_1x_2v_2 \dots v_rx_{r+1}$, and

(iii) $u_j \to v_j \in \psi_{i_j}(w_{i_j})$, $1 \leq j \leq r$, $x_i \in V_E^*$, $1 \leq i \leq r+1$.

A **state change** of a grammar ecosystem is an 0L evolution of the states of all the agents (i.e. $w_i \Longrightarrow_\Sigma w'_i$, according to the productions in $\varphi_i(w_E)$, $1 \leq i \leq n$) together with an 0L evolution of the environment in all its points (i.e. $w_E \Longrightarrow_\Sigma w'_E$, according to the productions in P_E), except those ones that are currently affected by the agents' actions (according to the productions in ψ_i).

Given two states in Σ, $\sigma = (w_E, w_1, w_2, \dots, w_n)$, $\sigma' = (w'_E, w'_1, w'_2, \dots, w'_n)$, one says that σ changes to σ' (or σ' directly derives from σ: $\sigma \Longrightarrow_\Sigma \sigma'$) if and only if:

(i) $w_E = z_1x_1z_2x_2 \dots z_mx_mz_{m+1}$ and $w'_E = z'_1y_1z'_2y_2 \dots z'_my_mz'_{m+1}$ and $z_1x_1z_2x_2 \dots z_mx_mz_{m+1} \Longrightarrow_\Sigma z_1y_1z_2y_2 \dots z_my_mz_{m+1}$ is a simultaneous action of all the agents $A_{i_1}, A_{i_2}, \dots, A_{i_m}$, $\{i_1, i_2, \dots, i_m\} \subseteq \{1, 2, \dots, n\}$, that are active in σ and $z'_1z'_2 \dots z'_mz'_{m+1}$ is an evolution from $z_1z_2 \dots z_mz_{m+1}$,

(ii) w'_i is an evolution of A_i from w_i, $1 \leq i \leq n$.

The sequences of states characterize the evolutionary behaviour of Σ. Let σ_0 be an initial state. One may define:

- The set of sequences of states of Σ:

  $$Seq(\Sigma, \sigma_0) = \{\{\sigma_i\}_{i=0}^\infty : \sigma_0 \Longrightarrow_\Sigma \sigma_1 \Longrightarrow_\Sigma \sigma_2 \Longrightarrow_\Sigma \dots\}.$$

- The set of sequences of states of E:

  $$Seq_E(\Sigma, \sigma_0) = \{\{w_{E_i}\}_{i=0}^\infty : \{\sigma_j\}_{j=0}^\infty \in Seq(\Sigma, \sigma_0) \text{ and } \sigma_j = (w_{E_j}, w_{1_j}, w_{2_j}, \dots, w_{n_j})\}.$$

- The set of sequences of states of A_j ($1 \leq j \leq n$):

  $$Seq_j(\Sigma, \sigma_0) = \{\{w_{j_k}\}_{k=0}^\infty : \{\sigma_k\}_{k=0}^\infty \in Seq(\Sigma, \sigma_0) \text{ and } \sigma_k = (w_{E_k}, w_{1_k}, \dots, w_{j_k}, \dots, w_{n_k})\}.$$

- The language of the environment:

  $$L_E(\Sigma, \sigma_0) = \{w_E \in V_E^* : \{\sigma_j\}_{j=0}^\infty \in Seq(\Sigma, \sigma_0) \text{ and } \sigma_j = (w_E, w_1, w_2, \dots, w_n)\}.$$

- The language of the agent A_i ($1 \leq i \leq n$):

  $$L_i(\Sigma, \sigma_0) = \{w_i \in V_i^* : \{\sigma_j\}_{j=0}^\infty \in Seq(\Sigma, \sigma_0) \text{ and } \sigma_j = (w_E, w_1, \dots, w_i, \dots, w_n)\}.$$

Example 5.8 *The grammar ecosystem:*

$$\Sigma = (E, A_1)$$

with:

$E = (V_E, P_E)$, *where:*

$V_E = \{a\}$,
$P_E = \{a \longrightarrow a\}$.

$A_1 = (V_1, P_1, R_1, \varphi_1, \psi_1)$, *where:*

$V_1 = \{b\}$,
$P_1 = \{b \longrightarrow b\}$,
$R_1 = \{a \longrightarrow a^4\}$,
$\varphi_1(w) = P_1$, *for every* $w \in V_E^*$,
$\psi_1(u) = R_1$, *for every* $u \in V_1^+$.

produces:

$Seq(\Sigma, \sigma_0) = \{(a, b), (a^4, b), (a^7, b), \ldots, (a^{3i+1}, b), \ldots\}$,

$L_E(\Sigma, \sigma_0) = \{a^{3i+1}, i \geq 0\}$,

$L_1(\Sigma, \sigma_0) = \{b\}$.

If P_1 *above is replaced by* $P_1' = \{b \longrightarrow \lambda\}$, *the agent becomes inactive after the first step and therefore does not act on the environment anymore:*

$Seq(\Sigma, \sigma_0) = \{(a, b), (a^4, \lambda)\}$,

$L_E(\Sigma, \sigma_0) = \{a, a^4\}$.

Example 5.9 *The grammar ecosystem:*

$$\Sigma = (E, A_1)$$

with:

$E = (V_E, P_E)$, *where:*

$V_E = \{e, f\}$,

$$P_E = \{e \longrightarrow e, f \longrightarrow f\}.$$

$A_1 = (V_1, P_1, R_1, \varphi_1, \psi_1)$, *where:*

$$\begin{aligned}
&V_1 = \{a\},\\
&P_1 = \{a \longrightarrow a, a \longrightarrow a^3\},\\
&\varphi_1(e) = \{a \longrightarrow a^3\},\\
&\varphi_1(f) = \{a \longrightarrow a\},\\
&R_1 = \{e \longrightarrow f, f \longrightarrow f\},\\
&\psi_1(a) = \{e \longrightarrow f\},\\
&\psi_1(a^3) = \{f \longrightarrow f\}.
\end{aligned}$$

produces:

$$L_1(\Sigma, (e, a)) = \{a, a^3\}.$$

5.4 Why Nonstandard Generative Mechanisms

5.4.1 Adjoining

Let $ICL(F)$, $F \in \{FIN, REG\}$, denote the family of languages generated by contextual grammars with selection of type F. Some major results concerning the generative capacity of contextual grammars are the following:

- $ICL(FIN) \subset ICL(REG) \subset ICL(CF) \subset ICL(CS) \subset ICL(RE)$.
- $REG \subset ICL(FIN)$.
- LIN and CF are incomparable with all families $ICL(F)$, $F \in \{REG, CF, CS, RE\}$, but $ICL(CS) \subset CS$. (Incomparability, meaning that the respective differences are nonempty, leads to eccentricity, which seems to be a linguistically valuable point to take into account.)
- $ICL(FIN)$ is an abstract anti-family of languages, i.e. a set closed to none of the six AFL operations: union, concatenation, Kleene star, morphisms, inverse morphisms and intersection with regular languages.

The latter result makes natural the question of finding the smallest AFL containing $ICL(FIN)$. The result is surprising.

Theorem 5.10 *For every* $L \in RE : L = L_1 \backslash L_2$, $L_1 \in REG$, $L_2 \in ICL(FIN)$.

So, iterated adjoining of contexts (i.e. paste), as selected by finite sets of strings, plus a left quotient (i.e. cut) by a regular language are able to simulate any Turing machine. What is needed is both: context-sensing and erasing abilities.

The same could be got by using a one-sided internal contextual grammar, all whose contexts are of the form (λ, v). The family of languages generated by one-sided internal contextual grammars is denoted by $1ICL(F)$ (with selection of type F).

Theorem 5.11 *For every* $L \in RE : L = (L_1 \backslash L_2) \cap V^*$, $L_1 \in REG$, $L_2 \in 1ICL(CF)$.

5.4.2 Inserting

An **insertion grammar** is a construct:

$$G = (V, A, P),$$

where:

- V is an alphabet,
- A is a finite set of axioms,
- P is a finite set of triples (u, x, v), $u, x, v \in V^*$ (insertion rules).

The immediate derivation works as follows:

foreevery$w, z \in V^* : w \Longrightarrow z$ if and only if $w = w_1 uv w_2, z = w_1 uxv w_2, w_1, w_2 \in V^*, (u, x, v) \in P$.

For an insertion grammar G, one defines its weight w:

$$w(G) = max\{|u| : (u, x, v) \in P \text{or} (v, x, u) \in P\}.$$

The family of languages generated by insertion grammars of weight at most n, $n \geq 0$, is denoted by INS_n. The union of all those families is INS_∞. The following results are known:

- $FIN \subset INS_0 \subset INS_1 \subset \ldots \subset INS_\infty \subset CS$.
- REG is incomparable with all families INS_n, $n \geq 1$, and $REG \subset INS_\infty$.
- $INS_1 \subset CF$, but CF is incomparable with all families INS_n, $n \geq 2$, and INS_∞.
- LIN is incomparable with all families INS_n, $n \geq 1$, and INS_∞.

- All families INS_n, $n \geq 0$, are anti-AFL's.

Theorem 5.12 *For every* $L \in RE : L = L_1 \backslash L_2$, $L_1 \in REG$, $L_2 \in INS_n$, $n \geq 7$.

What one does in this case is first to make use of insertion operations, and then to cut a prefix of the string by means of a quotient with respect to a regular language. If the cutting operation is introduced into the grammar, the so-called insertion-deletion grammars are obtained.

An **insertion-deletion grammar** is a construct:

$$G = (V, A, P_I, P_D),$$

where:

- V is an alphabet,
- A is a set of axioms,
- P_I is a set of insertion rules (a finite subset of $V^* \times V^* \times V^*$),
- P_D is a set of deletion rules (a finite subset of $V^* \times V^* \times V^*$).

The immediate derivation works as follows:

$$\text{for every} w, z \in V^* : w \Longrightarrow z \text{ if and only if } :$$

(i) either $w = w_1uvw_2, z = w_1uxvw_2, (u, x, v) \in P_I, w_1, w_2 \in V^*$,

(ii) or $w = w_1uxvw_2, z = w_1uvw_2, (u, x, v) \in P_D, w_1, w_2 \in V^*$.

The family of languages generated by insertion-deletion grammars is denoted by $INSDEL$.

Theorem 5.13 *For every* $L \in RE$, $L = L' \cap V^*$, *for some* $L' \in INSDEL$.

If the length of the strings that are inserted/deleted is taken into consideration, one can be more precise: $L' \in INS_1^2DEL_1^1$ (i.e. strings of length at most one are inserted in contexts of weight at most two, and strings of length at most one are deleted from contexts of weight at most one). As well, L' belongs to both $INS_2^1DEL_2^0$ and $INS_1^2DEL_2^0$.

5.4.3 Splicing

A **splicing rule** over an alphabet V is a string $r = u_1\#u_2\$u_3\#u_4$, $u_i \in V^*$, $1 \leq i \leq 4$, $\#, \$ \notin V$.

The immediate derivation relation runs as follows:

$$\text{for every} x, y, z \in V^* : (x, y) \vdash_r z \text{ if and only if } :$$

(i) $x = x_1u_1u_2x_2$,

(ii) $y = y_1u_3u_4y_2$,

(iii) $z = x_1u_1u_4y_2$,

with $x_1, x_2, y_1, y_2 \in V^*$.

An H **scheme** is a pair $\sigma = (V, R)$, $R \subseteq V^*\#V^*\$V^*\#V^*$.

Given $\sigma = (V, R)$ and a language $L \subseteq V^*$, one defines:

- $\sigma(L) = \{z \in V^* : (x, y) \vdash_r z, x, y \in L, r \in R\}$,
- $\sigma^0(L) = L$,
- $\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)), i \geq 0$,
- $\sigma^*(L) = \bigcup_{i \geq o} \sigma^i(L)$.

This allows the introduction of a new generative mechanism. An **extended** H **system** is a construct:

$$\gamma = (V, T, A, R),$$

where:

- V is an alphabet,
- $T \subseteq V$ is a set of terminal letters,
- $A \subseteq V^*$ is a set of axioms,
- $R \subseteq V^*\#V^*\$V^*\#V^*$.

($\sigma = (V, R)$ is the underlying H scheme of γ.)

The language generated by γ is:

$$L(\gamma) = \sigma^*(A) \cap T^*.$$

Given two families of languages F_1, F_2, $EH(F_1, F_2)$ denotes the family of languages $L(\gamma)$, $\gamma = (V, T, A, R)$, with $A \in F_1$, $R \in F_2$. A surprising result is obtained again.

Theorem 5.14 $EH(F_1, F_2) = RE$, *for every* $F_1, F_2 : FIN \subseteq F_1$, $REG \subseteq F_2$.

Thus, by using nonstandard resources the greatest generative power is achieved, while keeping the complexity of the mechanism at a very low level.

6 Papers Cited

- Bresnan, J., R.M. Kaplan, S. Peters and A. Zaenen. 1982. ”Cross-serial dependencies in Dutch”. *Linguistic Inquiry*, 13(4), 613–635.
- Culy, C. 1985. ”The complexity of the vocabulary of Bambara”. *Linguistics and Philosophy*, 8, 345–351.
- Gazdar, G. 1981. ”Unbounded dependencies and coordinate structure”. *Linguistic Inquiry*, 12(2), 155–184.
- Pullum, G.K. and G. Gazdar. 1982. ”Natural languages and context-free languages”. *Linguistics and Philosophy*, 4, 471–504.
- Shieber, S.M. 1985. ”Evidence against the context-freeness of natural language”. *Linguistics and Philosophy*, 8, 333–343.

7 Further Reading and Relevant Resources

Generally speaking, for a linguist wishing to be introduced into the field of mathematical methods in linguistics, which is a scope notably larger than the one taken in the present chapter, Partee, ter Meulen and Wall (1990) is strongly recommended. It is a book thought for initiating mathematically nontrained students. Brainerd (1971), Wall (1972) and Partee (1978) may be still valid references, too. One chapter in Cole, Varile and Zampolli (1997) explains the main trends for connecting different mathematical models with computational developments.

The most comprehensive and updated handbook of classical as well as nonclassical formal languages is Rozenberg and Salomaa (1997).

Very cited treatises in classical formal language theory (with different levels of difficulty) include Aho and Ullman (1971-1973), Davis, Sigal and Weyuker (1994), Harrison (1978), Hopcroft and Ullman (1979), Révész (1991), Salomaa (1988), and Wood (1987). Other good books (not all of them having a completely general scope) are Brookshear (1989), Dassow and Păun (1989), Drobot (1989), Floyd and Beigel (1994), Gurari (1989), Howie (1991), Kelley (1995), Lewis and Papadimitriou (1981), Linz (1990), McNaughton (1982), Moll, Arbib and Kfoury (1988), and Sudkamp (1988).

Some of the developments in nonstandard formal language theory can be found in Csuhaj-Varjú, Dassow, Kelemen and Păun (1994), Păun (1995), Păun (1997), and Păun, Rozenberg and Salomaa (1998).

The present state-of-the-art of the field is well pictured in Martín-Vide and Mitrana (2001a), and Martín-Vide and Mitrana (2001b).

The following references may be of help to the reader interested in knowing more about linguistic applications of formal language theory: Kolb and Mönnich (1999), Levelt (1974), Manaster Ramer (1987), Martín-Vide (1994), Martín-Vide (1998), Martín-Vide (1999), Martín-Vide and Păun (2000), Păun (1994), Savitch, Bach, Marsh and Safran-Naveh (1987), Savitch and Zadrozny (1994), Sells, Shieber and Wasow (1991), and Zadrozny, Manaster Ramer and Moshier (1993).

References

8 Books

Aho, A.V., J.E. Hopcroft and J.D. Ullman. 1974. *The design and analysis of computer algorithms*. Reading, MA: Addison-Wesley.

Aho, A.V., R. Sethi and J.D. Ullman. 1986. *Compilers: principles, techniques, and tools*. Reading, MA: Addison-Wesley.

Aho, A.V. and J.D. Ullman. 1971-1973. *The theory of parsing, translation and compiling*, 2 vols. Englewood Cliffs, NJ: Prentice-Hall.

Aho, A.V. and J.D. Ullman. 1977. *Principles of compiler design*. Reading, MA: Addison-Wesley.

Amos, M. 2001. *Theoretical and experimental DNA computation*. Berlin: Springer.

Arbib, M.A. 1969. *Theories of abstract automata*. Englewood Cliffs, NJ: Prentice-Hall.

Atallah, M.J. (Editor). 1998. *CRC handbook of algorithms and theory of computation*. Boca Raton, FL: CRC.

Bavel, Z. 1983. *Introduction to the theory of automata*. Reston, VA: Reston.

Berstel, J. 1979. *Transductions and context-free languages*. Stuttgart: Teubner.

Berstel, J. and C. Reutenauer. 1988. *Rational series and their languages*. Berlin: Springer.

Book, R.V. (Editor). 1980. *Formal language theory: perspectives and open problems*. New York, NY: Academic Press.

Booth, T.L. 1967. *Sequential machines and automata theory*. New York, NY: John Wiley.

Brainerd, B. 1971. *Introduction to the mathematics of language study*. New York, NY: Elsevier.

Brainerd, W.S. and L.H. Landweber. 1974. *Theory of computation*. New York, NY: John Wiley.

Brookshear, J.G. 1989. *Theory of computation: formal languages, automata, and complexity*. Redwood City, CA: Benjamin/Cummings.

Carroll, J. and D. Long. 1989. *Theory of finite automata*. Englewood Cliffs, NJ: Prentice-Hall.

Cohen, D.I.A. 1986. *Introduction to computer theory*. New York, NY: John Wiley. (2nd ed., 1991.)

Cole, R., G.B. Varile and A. Zampolli (Editors). 1997. *Survey of the state of the art in human language technology*. Cambridge: Cambridge University Press.

Csuhaj-Varjú, E., J. Dassow, J. Kelemen and Gh. Păun. 1994. *Grammar systems: a grammatical approach to distribution and cooperation*. London: Gordon and Breach.

Dassow, J. and Gh. Păun. 1989. *Regulated rewriting in formal language theory*. Berlin: Springer.

Davis, M.D. (Editor). 1965. *The undecidable: basic papers on undecidable propositions, unsolvable problems and computable functions*. New York, NY: Raven.

Davis, M.D., R. Sigal and E.J. Weyuker. 1983. *Computability, complexity, and languages: fundamentals of theoretical computer science*. New York, NY: Academic Press. (2nd ed., 1994.)

Denning, P.J., J.B. Dennis and J.E. Qualitz. 1978. *Machines, languages, and computation*. Englewood Cliffs, NJ: Prentice-Hall.

Drobot, V. 1989. *Formal languages and automata theory*. Rockville, MD: Computer Science Press.

Eilenberg, S. 1974-1976. *Automata, languages, and machines*, 2 vols. New York, NY: Academic Press.

Engeler, E. 1968. *Formal languages*. Chicago, IL: Markham.

Floyd, R.W. and R. Beigel. 1994. *The language of machines: an introduction to computability and formal languages*. Rockville, MD: Computer Science Press.

Gécseg, F. and I. Péak. 1972. *Algebraic theory of automata*. Budapest: Akadémiai Kiadó.

Gécseg, F. and M. Steinby. 1984. *Tree automata*. Budapest: Akadémiai Kiadó.

Gill, A. 1962. *Introduction to the theory of finite-state machines*. New York, NY: McGraw-Hill.

Ginsburg, S. 1962. *An introduction to mathematical machine theory*. Reading, MA: Addison-Wesley.

Ginsburg, S. 1966. *The mathematical theory of context-free languages*. New York, NY: McGraw-Hill.

Ginsburg, S. 1975. *Algebraic and automata-theoretic properties of formal languages*. Amsterdam: North-Holland.

Ginsburg, S., S. Greibach and J.E. Hopcroft. 1969. *Studies in abstract families of languages*. Providence, RI: American Mathematical Society.

Ginzburg, A. 1968. *Algebraic theory of automata*. New York, NY: Academic Press.

Gurari, E. 1989. *An introduction to the theory of computation*. New York, NY: Computer Science Press.

Harrison, M.A. 1965. *Introduction to switching and automata theory*. New York, NY: McGraw-Hill.

Harrison, M.A. 1969. *Lectures on linear sequential machines*. New York, NY: Academic Press.

Harrison, M.A. 1978. *Introduction to formal language theory*. Reading, MA: Addison-Wesley.

Hennie, F.C. 1968. *Finite-state models for logical machines*. New York, NY: John Wiley.

Herken, R. (Editor). 1988. *The universal Turing machine*. Oxford: Oxford University Press.

Herman, G.T. and G. Rozenberg. 1975. *Developmental systems and languages*. Amsterdam: North-Holland.

Hofstadter, D.R. 1979. *Gödel, Escher, Bach: an eternal golden braid*. New York, NY: Basic Books.

Hopcroft, J.E. and J.D. Ullman. 1969. *Formal languages and their relation to automata*. Reading, MA: Addison-Wesley.

Hopcroft, J.E. and J.D. Ullman. 1979. *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

Howie, J.M. 1991. *Automata and languages*. Oxford: Oxford University Press.

Ito, M. and H. Jürgensen (Editors). 1994. *Words, languages and combinatorics 2*. Singapore: World Scientific.

Karhumäki, J., H.A. Maurer, Gh. Păun and G. Rozenberg (Editors). 1999. *Jewels are forever*. Berlin: Springer.

Karhumäki, J., H.A. Maurer and G. Rozenberg (Editors). 1994. *Results and trends in theoretical computer science*. Berlin: Springer.

Kelley, D. 1995. *Automata and formal languages: an introduction*. Englewood Cliffs, NJ: Prentice-Hall.

Kolb, H.-P. and U. Mönnich (Editors). 1999. *The mathematics of syntactic structure: trees and their logics*. Berlin: Mouton de Gruyter.

Kuich, W. and A. Salomaa. 1986. *Formal power series and languages*. Berlin: Springer.

Kuich, W. and A. Salomaa. 1986. *Semirings, automata, languages*. Berlin: Springer.

Leeuwen, J. van (Editor). 1990. *Handbook of theoretical computer science*, 2 vols. Amsterdam/Cambridge, MA: North-Holland/MIT Press.

Levelt, W.J.M. 1974. *Formal grammars in linguistics and psycholinguistics*, 3 vols. The Hague: Mouton.

Lewis, H.R. and C.H. Papadimitriou. 1981. *Elements of the theory of computation*. Englewood Cliffs, NJ: Prentice-Hall.

Linz, P. 1990. *An introduction to formal languages and automata*. Lexington, MA: D.C. Heath. (2nd ed., 1996.)

Manaster Ramer, A. (Editor). 1987. *Mathematics of language*. Amsterdam: John Benjamins.

Marcus, S. 1967. *Algebraic linguistics: analytical models*. New York, NY: Academic Press.

Martín-Vide, C. (Editor). 1994. *Current issues in mathematical linguistics*. Amsterdam: North-Holland.

Martín-Vide, C. (Editor). 1998. *Mathematical and computational analysis of natural language*. Amsterdam: John Benjamins.

Martín-Vide, C. (Editor). 1999. *Issues in mathematical linguistics*. Amsterdam: John Benjamins.

Martín-Vide, C. and V. Mitrana (Editors). 2001a. *Grammars and automata for string processing: from mathematics and computer science to biology, and back*. London: Gordon and Breach.

Martín-Vide, C. and V. Mitrana (Editors). 2001b. *Where mathematics, computer science, linguistics and biology meet*. Dordrecht: Kluwer.

Martín-Vide, C. and Gh. Păun (Editors). 2000. *Recent topics in mathematical and computational linguistics*. Bucharest: Editura Academiei Române.

McNaughton, R. 1982. *Elementary computability, formal languages, and automata*. Englewood Cliffs, NJ: Prentice-Hall.

Minsky, M.L. 1967. *Computation: finite and infinite machines*. Englewood Cliffs, NJ: Prentice-Hall.

Moll, R.N., M.A. Arbib and A.J. Kfoury. 1988. *An introduction to formal language theory*. Berlin: Springer.

Moore, E.F. (Editor). 1964. *Sequential machines: selected papers*. Reading, MA: Addison-Wesley.

Nelson, R.J. 1968. *Introduction to automata*. New York, NY: John Wiley.

Nijholt, A. 1980. *Context-free grammars: covers, normal forms and parsing*. Berlin: Springer.

Nijholt, A. 1988. *Computers and languages: theory and practice*. Amsterdam: North-Holland.

Papadimitriou, C.H. 1994. *Computational complexity*. Reading, MA: Addison-Wesley.

Partee, B.H. 1978. *Foundations of mathematics for linguistics*. Harrisburg, PA: Greylock.

Partee, B.H., A.G.B. ter Meulen and R.E. Wall. 1990. *Mathematical methods in linguistics*. Dordrecht: Kluwer.

Păun, Gh. (Editor). 1994. *Mathematical aspects of natural and formal languages*. Singapore: World Scientific.

Păun, Gh. (Editor). 1995. *Artificial life: grammatical models*. Bucharest: Black Sea University Press.

Păun, Gh. 1997. *Marcus contextual grammars*. Dordrecht: Kluwer.

Păun, Gh. (Editor). 1998. *Computing with bio-molecules: theory and experiments*. Singapore: Springer.

Păun, Gh., G. Rozenberg and A. Salomaa. 1998. *DNA computing: new computing paradigms*. Berlin: Springer.

Păun, Gh. and A. Salomaa (Editors). 1997. *New trends in formal languages: control, cooperation, and combinatorics*. Berlin: Springer.

Păun, Gh. and A. Salomaa (Editors). 1999. *Grammatical models of multi-agent systems*. London: Gordon and Breach.

Paz, A. 1971. *Introduction to probabilistic automata*. New York, NY: Academic Press.

Pin, J.-E. 1986. *Varieties of formal languages*. Oxford: Plenum. (Orig., 1984.)

Révész, G.E. 1983. *Introduction to formal languages*. New York, NY: McGraw-Hill. (2nd ed.: New York, NY: Dover, 1991.)

Rozenberg, G. and A. Salomaa. 1980. *The mathematical theory of L systems*. New York, NY: Academic Press.

Rozenberg, G. and A. Salomaa (Editors). 1986. *The book of L*. Berlin: Springer.

Rozenberg, G. and A. Salomaa (Editors). 1992. *Lindenmayer systems: impacts on theoretical computer science, computer graphics and developmental biology*. Berlin: Springer.

Rozenberg, G. and A. Salomaa (Editors). 1997. *Handbook of formal languages*, 3 vols. Berlin: Springer.

Rozenberg, G. and W. Thomas (Editors). 2000. *Developments in language theory: foundations, applications and perspectives*. Singapore: World Scientific.

Salomaa, A. 1969. *Theory of automata*. Oxford: Pergamon.

Salomaa, A. 1973. *Formal languages*. New York, NY: Academic Press.

Salomaa, A. 1981. *Jewels of formal language theory*. Rockville, MD: Computer Science Press.

Salomaa, A. 1985. *Computation and automata*. Cambridge: Cambridge University Press.

Salomaa, A. and M. Soittola. 1978. *Automata-theoretic aspects of formal power series*. Berlin: Springer.

Savitch, W.J., E.W. Bach, W. Marsh and G. Safran-Naveh (Editors). 1987. *The formal complexity of natural language*. Dordrecht: Reidel.

Savitch, W.J. and W. Zadrozny (Editors). 1994. *Mathematics of language*, Linguistics and Philosophy, 17(6). Dordrecht: Kluwer.

Sells, P., S.M. Shieber and T. Wasow (Editors). 1991. *Foundational issues in natural language processing*. Cambridge, MA: MIT Press.

Sikkel, K. 1996. *Parsing schemata: a framework for specification and analysis of parsing algorithms*. Berlin: Springer.

Sippu, S. and E. Soisalon-Soininen. 1988-1990. *Parsing theory*, 2 vols. Berlin: Springer.

Sudkamp, T.A. 1988. *Languages and machines*. Reading, MA: Addison-Wesley.

Trakhtenbrot, B.A. and Y.M. Barzdin. 1973. *Finite automata*. Amsterdam: North-Holland.

Turing, A.M. 1992. *Mechanical intelligence*, ed. by D.C. Ince. Amsterdam, North-Holland.

Wall, R.E. 1972. *Introduction to mathematical linguistics*. Englewood Cliffs, NJ: Prentice-Hall.

Wood, D. 1980. *Grammar and L forms: an introduction*. Berlin: Springer.

Wood, D. 1987. *Theory of computation*. New York, NY: John Wiley.

Zadrozny, W., A. Manaster Ramer and M.A. Moshier (Editors). 1993. *Mathematics of language*, Annals of Mathematics and Artificial Intelligence, 8(1-2). Basel: J.C. Baltzer.

9 Journals

Acta Cybernetica

Acta Informatica

BioSystems

Bulletin of the European Association for Theoretical Computer Science

Communications of the Association for Computing Machinery

Computational Intelligence

Computational Linguistics

Computers and Artificial Intelligence

Discrete Mathematics

Fundamenta Informaticae

Grammars. A Journal of Mathematical Research on Formal and Natural Languages

Information and Computation (before, *Information and Control*)

Information Processing Letters

International Journal of Computer Mathematics

International Journal of Foundations of Computer Science

Journal of Automata, Languages and Combinatorics

Journal of Computer and System Sciences

Journal of Logic, Language and Information

Journal of the Association for Computing Machinery

Journal of Universal Computer Science

Kybernetika

Linguistics and Philosophy

Mathematical Structures in Computer Science (before, *Mathematical Systems Theory*)

New Generation Computing

Revue Franaise d'Automatique, Informatique et Recherche Opérationelle (RAIRO): Informatique Théorique

Revue Roumaine de Mathématiques Pures et Appliquées

Theoretical Computer Science

10 Major Conferences

- ACL Annual Meeting of the Association for Computational Linguistics
- AFL Conference on Automata and Formal Languages
- AMiLP Algebraic Methods in Language Processing
- CIAA International Conference on Implementation and Application of Automata
- CLIN Computational Linguistics in the Netherlands Meeting
- COLING International Conference on Computational Linguistics
- DCAGRS Workshop on Descriptional Complexity of Automata, Grammars and Related Structures
- DIMACS International Meeting on DNA Based Computing
- DLT International Conference Developments in Language Theory
- ESSLLI European Summer School of Logic, Language and Information

- FCT Fundamentals of Computation Theory
- GS International Workshop Grammar Systems
- ICALP International Colloquium on Automata, Languages and Programming
- ICWLC International Colloquium on Words, Languages and Combinatorics
- IWPT International Workshop on Parsing Technologies
- LACL International Conference on Logical Aspects of Computational Linguistics
- MCU International Conference on Machines, Computations and Universality
- MFCS International Symposium on Mathematical Foundations of Computer Science
- MOL Meeting on Mathematics of Language
- UMC Conference on Unconventional Models of Computing

11 Professional Associations

- Association for Computational Linguistics, Special Interest Group on Mathematics of Language, http://www.cis.upenn.edu/ ircs/mol/mol.html
- Association for Computing Machinery, Special Interest Group on Algorithms and Computing Theory, http://sigact.acm.org/
- European Association for Theoretical Computer Science, http://www.eatcs.org/

12 Web Sites

- American Mathematical Society Preprint Server, http://www.ams.org/preprints/
- Collection of Computer Science Bibliographies, http://liinwww.ira.uka.de/bibliography/index.html
- Computing Research Repository, http://xxx.lanl.gov/archive/cs/intro.html
- Mathematics WWW Virtual Library, http://euclid.math.fsu.edu/Science/math.html
- Networked Computer Science Technical Reference Library, http://www.ncstrl.org/

13 Research Centres

- Leiden Institute of Advanced Computer Science, http://www.wi.leidenuniv.nl/CS/
- Turku Centre for Computer Science, http://www.tucs.fi/