University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, & Professional Papers

Graduate School

1973

Mathematical linguistics and automata theory and applications to biological growth models

Paul W. Bennett
The University of Montana

Follow this and additional works at: https://scholarworks.umt.edu/etd

Let us know how access to this document benefits you.

Recommended Citation

Bennett, Paul W., "Mathematical linguistics and automata theory and applications to biological growth models" (1973). *Graduate Student Theses, Dissertations, & Professional Papers*. 6670.
https://scholarworks.umt.edu/etd/6670

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.

MATHEMATICAL LINGUISTICS AND AUTOMATA THEORY AND APPLICATIONS TO BIOLOGICAL GROWTH MODELS

By

Paul W. Bennett

B.E.E., M.S., Cornell University, 1963-64

Presented in partial fulfillment of the requirements for the degree of

Master of Arts

University of Montana

1973

Approved by:

[illegible]
Chairman, Board of Examiners

[illegible]
Dean, Graduate School

July 23, 1973
Date

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: EP37471

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI EP37471
Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7-24-73

PREFACE

The idea of a mathematical grammar was introduced by Chomsky (1956) as an independent subject, formalizing the intuitive notions of a grammar in languages used for communication, in terms of a set of substitution rules which generate a set of "words" or a "language". The study of these grammars developed concurrently with automata theory, and each of these subjects has been used in studying the other, thus becoming closely intertwined.

Recently a new class of automata known as "developmental systems" has appeared, originally introduced as a model for certain types of biological growth. These automata are now being actively investigated using mathematical linguistics as a tool.

In the first two chapters we bring together the fundamental known results about the basic types of automata and mathematical grammars. In Chapters 3 and 4 we describe developmental systems, and present some language-theoretical results pertaining to them, some of which are new and some that have already appeared. Chapter 5 deals with computer simulation of developmental systems, using some specific models for illustration.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENT

The author wishes to acknowledge the patience, encouragement, and valuable help of Prof. F. N. Springsteel, which made this thesis possible.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

BASIC RESULTS OF AUTOMATA THEORY

The study of automata theory began with the introduction of Turing machines (Turing, 1936). This was probably the first mention of an abstract mathematical "mechanism". Later the idea of a finite automaton was crystallized, also as a mechanism or device, but operating in a much more restricted way. Its relation to linguistics as a recognizer of a certain type of mathematical language was quickly established, as well as the convenient properties of those languages. These results led to the search for other classes of automata capable of recognizing different classes of languages. In this chapter we outline the idea of a finite automaton and the characterization of its language in terms of regular expressions, and then briefly examine Turing machines. In the interest of brevity and clarity only outlines of proofs and constructions are given.

1. FINITE AUTOMATA

The system that is now known as a finite automaton has evolved as the most basic generalization of discrete systems, i.e. systems which can exist in any of a finite number of states and change state at discrete points in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

time. Such systems have appeared in various fields such as mechanics, biology (McCulloch and Pitts, 1943), and finally digital computer design, at which time interest in studying the basic properties of discrete systems developed.

<u>Definition</u> A <u>finite</u> <u>automaton</u> (f.a.) is a system $(K, \Sigma, \delta, q_0, F)$ where

- K is the finite set of internal <u>states</u>
- Σ is a finite input <u>alphabet</u>
- $\delta: K \times \Sigma \longrightarrow K$ is the <u>next</u> <u>state</u> <u>function</u>
- $q_0 \in K$ is the <u>initial</u> <u>state</u>
- $F \subseteq K$ is the set of <u>final</u> <u>states</u>.

<u>Definition</u> 1. Σ^* is the set of "words" or "sentences" consisting of strings of symbols in Σ, including the empty word ϵ (the word consisting of no symbols). Σ^* is called the "star closure" of Σ, or the free monoid generated by Σ.

2. $\Sigma^+ = \Sigma^* - \{\epsilon\}$.

The finite automaton M can be thought of as a machine receiving an input word of finite length, one symbol at a time. As each input symbol is received, the δ function is applied to determine the next state of the machine. The argument of the δ function can be extended to include input words, instead of single input symbols, by a recursive definition:

$$\delta(q,xa) = \delta(\delta(q,x),a) \text{ for any } x \in \Sigma^*, a \in \Sigma$$

and $\delta(q,\epsilon) = q$ where ϵ is the empty word.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

<u>Definition</u> 1. A finite automaton M <u>accepts</u> a word $x \in \Sigma^*$ if $\delta(q_0, x)$ is in F.

2. $L(M) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}$ (the set of words in Σ^* accepted by M).

3. S is a <u>regular set</u> if $S = L(M)$ for some finite automaton M.

<u>Definition</u> An equivalence relation R over a set T is <u>right invariant</u> if x R y implies xz R yz for all z in T.

<u>Theorem</u> 1.1 (Myhill, 1957) Suppose $L \subseteq \Sigma^*$. Then the following are equivalent:

1. L is a regular set

2. L is a union of equivalence classes of a right invariant equivalence relation over Σ^* of finite index. (An equivalence relation has finite index if its set of equivalence classes is finite.)

3. The equivalence relation R is of finite index, where R is defined by: x R y if and only if for all $z \in \Sigma^*$ $xz \in L$ when $yz \in L$.

Proof. $1 \Longrightarrow 2$: Suppose L is accepted by a f.a. M. Define E by x E y if and only if $\delta(q_0, x) = \delta(q_0, y)$. E is right invariant, and has finite index since K is a finite set. Then L is the union of equivalence classes containing a word x such that $\delta(q_0, x) \in F$.

$2 \Longrightarrow 3$: Any equivalence relation E satisfying con-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

dition 2 is a refinement of the equivalence relation R, hence E having finite index implies R has finite index.

3 $\Rightarrow$ 1: Construct the f.a. $M' = (K', \Sigma', \delta', q_0', F')$ as follows:

K' is the set of equivalence classes of R. Denote the equivalence class containing x by $[x]$.

$\Sigma' = \Sigma$

$\delta'([x], a) = [xa]$ (consistent since R is right invariant)

$q_0' = [\epsilon]$

$F' = \{[x] \mid x \in L\}$.

Then $\delta'(q_0', x) = [x]$, and hence M' accepts L. Q.E.D.

<u>Corollary</u> The minimum state f.a. accepting L is unique, and is isomorphic to M' of the previous theorem.

Proof. From the previous theorem, any f.a. M accepting L defines an equivalence relation in Σ^* which is a refinement of R, so that M has at least as many states as M'.

Furthermore if M has the same number of states as M', each state of M can be identified with one of the states of M'. Q.E.D.

<u>Definition</u> A non-deterministic finite automaton is a system satisfying the previous definition of a (deterministic) finite automaton, except that for any $q \in K$ and $a \in \Sigma$, $\delta(q, a)$ can be any subset of K, instead of a single state

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in K, with the interpretation that in any particular instance, the next state can be chosen to be any state contained in this subset.

A word x is accepted by a non-deterministic f.a. M if there is a sequence of states possible under the input x leading to a state in F.

Theorem 1.2 If L is accepted by a non-deterministic f.a. M, it is accepted by a deterministic f.a. M'.

Proof. M' can be constructed from M by defining the states of M' to consist of all subsets of the states of M. The set of final states of M' will be the set of all subsets of states of M containing a final state of M.

2. STATE GRAPHS AND REGULAR EXPRESSIONS

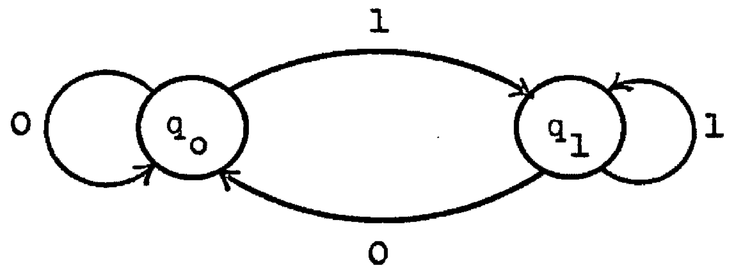
A state graph for a f.a. presents a simple picture of the operation of the machine, and has been a traditional means of specifying a particular machine behavior. Regular expressions were introduced as a way of specifying the language, or set of words, recognized by a particular f.a. We now study the relation between these two characterizations of a f.a.

Definition A state diagram or graph is a finite directed graph in which the vertices represent states of a f.a. and the arrows represent transitions between states in accord-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ance with the next state function.

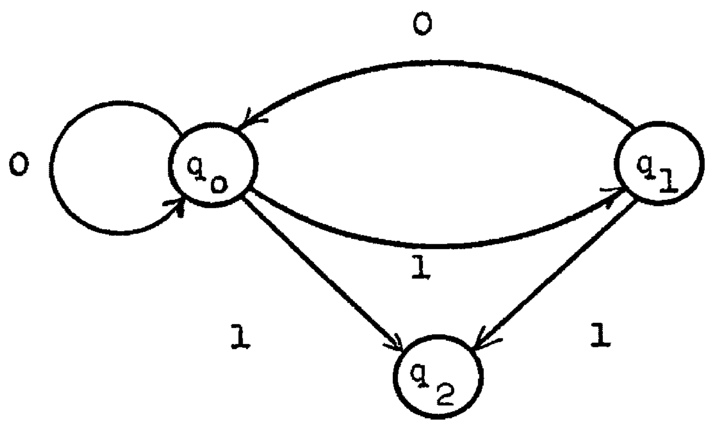
Example 1.2a Suppose $K = \{ q_0, q_1 \}$, $\Sigma = \{ 0, 1 \}$, $F = q_1$, and δ is given by $\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$, $\delta(q_1, 0) = q_0$, $\delta(q_1, 1) = q_1$. The corresponding state graph is



This is an example of a deterministic finite automaton.

Example 1.2b Suppose $K = \{ q_0, q_1, q_2 \}$, $\Sigma = \{ 0, 1 \}$, $F = q_2$, and δ is given by $\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = \{ q_1, q_2 \}$, $\delta(q_1, 0) = q_0$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 0) = \phi$, $\delta(q_2, 1) = \phi$.

The state diagram is



Note that this machine is non-deterministic.

Definition 1. $RS = \{ xy \mid x \in R, y \in S \}$.

2. $R + S = \{ x \mid x \in R \text{ or } x \in S \}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Definition A regular expression is an expression obtained by a finite number of applications of the above operations and star closure * (see definition on p. 2) to elements of $\Sigma \cup \{\epsilon\} \cup \{\phi\}$, and to expressions obtained from them by such applications of these operations.

Every regular expression represents a set of words in Σ^*, i.e. a "language". However an arbitrary subset of Σ^* may not be representable by a regular expression.

Theorem 1.3 below will state that a subset of Σ^* is a regular set, or regular language, if and only if it is representable in terms of some regular expression.

Definition If R is any set of words in Σ^* and $x \in \Sigma^*$, the derivative of R with respect to x is defined as

$$D_x R = \{ t \mid xt \in R. \}$$

The derivatives of any regular expression can be computed using the following rules:

$$\lambda(R) = \begin{cases} \epsilon & \text{if } \epsilon \text{ is in } R \\ \phi & \text{if } \epsilon \text{ is not in } R \end{cases}$$

$$\lambda(RS) = \lambda(R)\lambda(S)$$

$$D_a(a) = \epsilon$$

$$D_a(b) = \phi \text{ for } b = \epsilon \text{ or } b = \phi, \text{ or } b \neq a$$

$$D_a(R^*) = D_a(R)\, R^*$$

$$D_a(RS) = D_a(R)\, S + \lambda(R)\, D_a(S)$$

$$D_a(R+S) = D_a(R) + D_a(S)$$

Note that although ϵ denotes the empty word, it

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can be an element of a language, hence is distinguished from the empty set ϕ .

Every regular expression has a finite number of distinct derivatives.

<u>Theorem</u> 1.3 (Kleene, 1956) If M is any finite automaton, L(M) can be represented by a regular expression over Σ, and for every regular expression R there is a f.a. M such that L(M) is represented by R.

Proof. The idea of the construction in both cases is that each distinct derivative of R corresponds to a state of the machine. The initial state always corresponds to $D_\epsilon(R) = R$, and will now be denoted q_ϵ, instead of q_0 as was the case previously, to avoid confusion. The state diagram consists of transitions of the form

$$\begin{tikzcd} q_x \arrow[r, "a"] & q_{xa} \end{tikzcd}$$

where q_x is identified with $D_x(R)$, and q_{xa} with $D_{xa}(R)$, because of the relation $D_{xa}(R) = D_a(D_x(R))$.

Hence to construct the state diagram given a regular expression, compute the distinct derivatives and associate a state with each, according to the above system. The resulting f.a. is the minimal one for the given language.

Given a state diagram, to construct the corresponding regular expression, form a system of equations of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

form

$$D_x R = a_1 D_{xa_1} R + \cdots + a_n D_{xa_n} R + \alpha$$

where $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and $\alpha = \epsilon$ if the state associated with $D_x R$ is a final state, ϕ otherwise. There will be one equation for each state of M. Then use the state diagram to identify equal derivatives, and solve the system of equations for R. Q.E.D.

An inference rule that is often useful in solving the system of equations is

$$R = SR + T \quad \Rightarrow \quad R = S^* T$$

if ϵ is not in S, where R,S,T are regular expressions.

These procedures are illustrated by the following examples.

Example 1.2c Suppose $R = 0(0^*10)^*0$. Construct the corresponding state diagram.

For simplicity let D_x stand for $D_x R$:

$$D_\epsilon = R$$

$$D_0 = (0^*10)^*0$$

$$D_{00} = (0^*10)(0^*10)^*0$$

$$D_{000} = (0^*10)(0^*10)^*0$$

$$D_1 = D_{01} = D_{11} = D_{001} = D_{011} = \phi$$

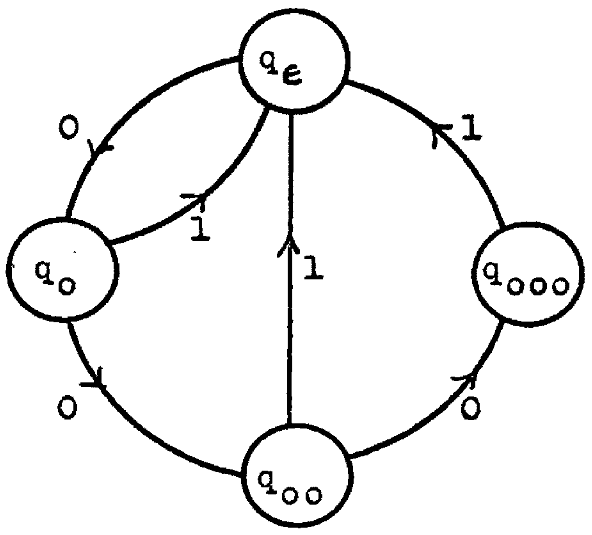
$$D_{10} = D_{100} = R$$

$$D_{010} = D_0$$

Hence there are three distinct derivatives (in addition to $D_\epsilon = R$), so there are four states, and the state

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

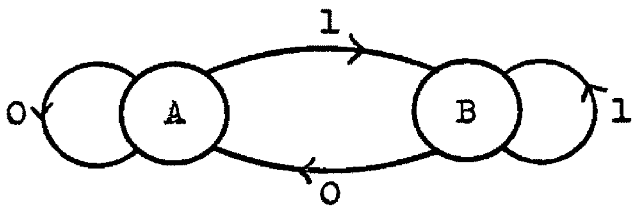
graph is:



Note that the initial state is q_ϵ and the final state is q_{00}.

This is the minimal f.a. accepting the language represented by the given regular expression.

Example 1.2d Find the regular expression associated with the diagram



The final state is B.

Associate D_ϵ with A and $D_{\epsilon 1} = D_1$ with B. Then form the equations

$$D_\epsilon = R = 0D_0 + 1D_1$$
$$D_1 = 0D_0 + 1D_{11} + \epsilon$$

From the diagram it is clear that $D_0 = D_\epsilon$, $D_{10} = D_\epsilon$, and $D_{11} = D_1$. Hence the equations become

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$R = 0R + 1D_1$$

$$D_1 = 0R + 1D_1 + \epsilon$$

Using the previously mentioned inference rule, the second equation can be solved for D_1:

$$D_1 = 1^*(0R + \epsilon) = 1^*0R + 1^*$$

This expression is substituted into the first equation to get

$$\begin{aligned} R &= 0R + 1(1^*0R + 1^*) \\ &= 0R + 11^*0R + 11^* \\ &= (0 + 11^*0)R + 11^* \\ R &= (0 + 11^*0)^*11^*. \end{aligned}$$

3. PROPERTIES OF REGULAR LANGUAGES

In this section we present the convenient properties of the class of regular languages.

Theorem 1.4 If L is a regular set, then $\Sigma^* - L = L'$ is a regular set.

Proof. L regular implies L is accepted by a f.a. $M = (K, \Sigma, \delta, q_0, F)$. Then L' is accepted by $M' = (K, \Sigma, \delta, q_0, K - F)$.

Theorem 1.5 If L_1 and L_2 are regular sets then $L_1 \cap L_2$ is regular.

Proof. L_1 is a union of equivalence classes of a right invariant equivalence relation R_1, and L_2 is the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

union of equivalence classes of a right invariant equivalence relation R_2. Then the intersection of these two unions is a union of equivalence classes of the common equivalence relation $R_1 \cap R_2$.

<u>Theorem</u> 1.6 If L_1 and L_2 are regular then $L_1 \cup L_2$ is regular.

Proof. $L_1 \cup L_2 = (L_1' \cap L_2')'$, and apply the two preceding theorems.

<u>Corollary</u> The class of regular sets forms a Boolean algebra.

<u>Theorem</u> 1.7 Any finite set is regular.

Proof. A f.a. accepting a single word $a_1 a_2 \ldots a_n$ can be constructed by identifying each a_i with a state of M, and adding an initial state and a final state. Then any finite set is a union of single words, hence is regular by application of Theorem 1.6.

<u>Theorem</u> 1.8 If L_1 and L_2 are regular, then $L_1 L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$ is regular.

Proof. A non-deterministic f.a. M_3 can be constructed which initially behaves like M_1, the f.a. accepting L_1, and as the input is read, at any point chooses either to remain as M_1 or convert to simulation of M_2, the f.a. accepting L_2. Then M_3 accepts $L_1 L_2$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Theorem 1.9 If L is regular then L^* is regular.

Proof. If M is a f.a. accepting L, then a non-deterministic f.a. M' can be constructed which acts like M until a final state is reached, then chooses either to stop or return to q_0 and continue reading the input.

With the above results, Kleene's theorem may now be restated in terms of the closure properties of regular languages.

Theorem 1.10 The class of regular sets is the smallest class containing all finite sets and closed under union, concatenation (as defined in Theorem 1.8) and star closure.

4. TURING MACHINES

The Turing machine (TM) is a device which has very general powers of computation and recognition; in fact no "procedure", i.e. finite sequence of instructions has been found that could not be modeled by a Turing machine. This leads to the conjecture, known as Church's thesis, that there is a TM which realizes any algorithm or procedure.

A Turing machine basically consists of:

1 a tape divided into cells which is infinite in length in one direction

2 a finite set Γ of tape symbols

3 a finite control which at any time contains one of a finite set K of control states

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4 a tape head which scans one cell of the tape at a time.

A Turing machine is defined as a system $T = (K, \Sigma, \Gamma, \delta, q_0, F)$, with K and Γ as specified above, and

$\Sigma \subseteq \Gamma - \{B\}$ is the input alphabet, where B is the blank symbol,

$\delta: K \times \Gamma \longrightarrow K \times \Gamma \times \{L, R\}$ is the transition function,

$q_0 \in K$ is the start state,

$F \subseteq K$ is the set of final states.

A single move involves reading the symbol under the tape head, and then

1 writing a symbol on that cell,

2 changing the control state, and

3 moving right or left one cell,

all in accordance with the transition function δ.

Initially an input word of length n is entered in the leftmost n cells of the tape. The machine, starting in state q_0 scanning the leftmost cell, then performs a computation consisting of a series of moves determined by the transition function. The machine halts if it enters a configuration for which its δ function is not defined.

When used as a recognizer, the TM accepts or rejects any input word presented to it. The language accepted by a TM is defined to be the set of words in Σ^* which cause the TM to enter a final state and halt.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

It is often convenient in Turing machine construction to make use of "modifications" of the basic TM definition:

A non-deterministic TM is not limited to a single choice for the next move in all configurations, but rather may have several choices.

A multi-track TM has its tape divided into several tracks, with a one-to-one correspondence between the cells on each pair of tracks. This essentially amounts to considering a tape symbol as a k-tuple.

A multi-tape TM has several tapes, each with its own independent tape head.

These modifications, as well as others, do not increase the computing power of the TM, and it can be shown that there is a standard TM equivalent to each of these modified machines.

As an example, consider the language $\{ 1^{2^n} \mid n \geqslant 0\}$. We describe macroscopically a TM M accepting this language. M has a second tape with its own head, which is used as a binary counter, with the least significant digit in the leftmost cell. M scans the input word moving left to right. If a symbol other than 1 is encountered, the machine halts and rejects. Every time a 1 is read, M increases the stored count by one, so that tape 2 contains a count of the number of 1's scanned. When the end of the input word is reached, M accepts if the word on tape 2 is of the form 0...01.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A linear bounded automaton (lba) is a single-tape TM which uses only the input word portion of the tape for computing. Γ contains two special endmarker symbols which are placed at the ends of the input, and which form spacial operating bounds for the machine. The terms "deterministic" and "non-deterministic" have the same meaning for lba's as for general Turing machines.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

PHRASE-STRUCTURE GRAMMARS

Mathematical grammars are formalizations of the grammars that we use in natural languages. A grammar consists of a set of symbols and a set of rules for constructing "sentences" or "words" (both terms are used interchangeably). Just as in English a sentence is made up of a noun phrase and a verb phrase, a formal grammar contains a special symbol S called a "sentence symbol", and a rule $S \rightarrow \alpha$ where α is a string of symbols, corresponding to the rule (sentence) $\longrightarrow$ (noun phrase)(verb phrase) in English. The remaining rules are used to generate sentences from α. The collection of all sentences derivable by a grammar is called the language of that grammar.

This chapter defines and examines the hierarchy of mathematical grammars.

1. THE CLASSIFICATION OF GRAMMARS

Definition A phrase structure grammar is a system $G = (N,T,P,S)$ in which

N is a finite set of variables,

T is a finite set of terminal symbols,

$S \in N$ is the start symbol or sentence symbol,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

P is a set of productions of the form $\alpha \longrightarrow \beta$, where $\alpha \in (N \cup T)^* - \{\epsilon\}$ and $\beta \in (N \cup T)^*$.

If $\alpha \longrightarrow \beta$ is a production in G and γ and δ are strings in $(N \cup T)^*$, then $\gamma \alpha \delta \Longrightarrow \gamma \beta \delta$ is a <u>direct derivation</u> in G. If $\alpha_1 \Longrightarrow \alpha_2$, $\alpha_2 \Longrightarrow \alpha_3$, ..., $\alpha_{m-1} \Longrightarrow \alpha_m$ (for some $m \geqslant 1$) then $\alpha_1 \overset{*}{\Longrightarrow} \alpha_m$ is a <u>derivation</u> in G.

<u>Definition</u> L(G), the language generated by the grammar G, is the set $\{ w \in T^* \mid S \overset{*}{\Longrightarrow} w \}$.

<u>Example</u> 2.1a Suppose $N = \{ S, A \}$, $T = \{0, 1\}$, and P consists of the productions:

$$S \longrightarrow 0A \quad (P1)$$
$$A \longrightarrow 0A \quad (P2)$$
$$A \longrightarrow 1S \quad (P3)$$
$$A \longrightarrow 0 \quad (P4)$$

Then L(G) is the set represented by the following regular expression:

$$L(G) = \underset{P1}{0} \; (\underset{P2}{0^*} \; \underset{P3}{1} \; \underset{P1}{0})^* \; \underset{P4}{0}$$

where each component arises from application of the indicated production. Examples of sentences in L(G) are 00, 0100, 000100, 00010000100. (The finite automaton accepting this language was constructed in Example 1.2c.)

It is not always easy to characterize explicitly

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the language generated by a grammar.

Definition 1. A context-sensitive grammar is a grammar with the property that if $\alpha \longrightarrow \beta$ is a production in P, then $|\alpha| \leq |\beta|$, where $|\gamma|$ denotes the number of symbols in a string γ .

Since β cannot be ϵ , a context-sensitive language cannot contain ϵ .

2. A context-free grammar is one such that for every production $\alpha \longrightarrow \beta$ in P, α is a single variable in N and β is any string of variables and terminals.

The definition implies that in a derivation any variable can be replaced independent of the context in which it appears.

3. A regular grammar is a grammar such that the only productions are of the form $A \longrightarrow aB$ or $A \longrightarrow a$ where $A, B \in N$ and $a \in T$.

Example 2.1a presented a regular grammar. Some further examples now follow.

Example 2.1b The language a^*b^* corresponds to the following regular grammar: $N = \{S, V\}$, $T = \{a, b\}$, and P consists of:

$$S \longrightarrow aS \qquad V \longrightarrow bV$$
$$S \longrightarrow bV \qquad V \longrightarrow b$$
$$S \longrightarrow a \qquad S \longrightarrow b$$

This language is called a regular language since it

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is generated by a regular grammar.

<u>Example</u> 2.1c Let $N = \{S\}$, $T = \{a, b\}$, $P = \{(S \longrightarrow aSb), (S \longrightarrow ab)\}$. Then G is a context-free grammar, with $L(G) = \{a^n b^n \mid n \geq 1\}$. Compare this context-free language with the regular language in example 2.1b.

<u>Example</u> 2.1d The language $L(G) = \{a^n b^n c^n \mid n \geq 1\}$ is context-sensitive since it corresponds to the grammar whose productions are

$$\begin{array}{lll} S \longrightarrow aSBC & \qquad & bC \longrightarrow bc \\ S \longrightarrow aBC & & cC \longrightarrow cc \\ CB \longrightarrow BC & & aB \longrightarrow ab \\ bB \longrightarrow bb & & \end{array}$$

2. REGULAR LANGUAGES

The following two theorems provide the connection between regular languages and finite automata.

<u>Theorem</u> 2.1 If $G = (N,T,P,S)$ is a regular grammar, then there is a f.a. $M = (K, \Sigma, \delta, q_o, F)$ which accepts L(G), i.e. $L(M) = L(G)$.

Proof. Construct M from G as follows:

$$K = N \cup \{A\} \qquad (A \notin N)$$

$$\Sigma = T$$

$$q_o = S$$

$$F = \begin{cases} \{A\} & \text{if P does not contain S} \\ \{S, A\} & \text{if P contains S} \end{cases}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\delta(B, a) = \begin{cases} \{C \mid P \text{ contains } B \rightarrow aC\} & \text{if } B \rightarrow a \text{ is not in } P \\ \{C \mid P \text{ contains } B \rightarrow aC\} \cup \{A\} & \text{if } B \rightarrow a \text{ is in } P. \end{cases}$$

Then the non-deterministic f.a. M accepts L(G). Q.E.D.

Theorem 2.2 If M is a finite automaton, there is a regular grammar G such that $L(G) = L(M)$.

Proof. Define G as follows: If $M = (K, \Sigma, \delta, q_0, F)$, then $G = (N, T, P, S)$ where $N = K$, $T = \Sigma$, $S = q_0$ and P is defined by

1. $B \longrightarrow aC$ is in P if $\delta(B, a) = C$
2. $B \longrightarrow a$ is in P if $\delta(B, a) = C$ and C is in F.

Then G generates L(M). Q.E.D.

3. CONTEXT-FREE LANGUAGES

Let us now examine some properties of context-free languages and some decidability questions concerning these languages.

Theorem 2.3 If G is a context-free grammar, there is an algorithm for determining if G generates a non-empty language.

Proof. This follows from the fact that if N contains k symbols, then if L(G) is non-empty there must be a minimal derivation of length less than or equal to k of a word in T^*.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

<u>Theorem</u> 2.4 If L is a context-free language, there exist constants p and q such that if z is in L, and $|z| > p$, then $z = uvwxy$ where $|vwx| \leq q$, v and x not both ϵ, and uv^iwx^iy is in L for $i \geqslant 0$.

Proof. Let p be the maximum length of all words generated by derivations of length less than or equal to n, the number of symbols in N. Then $|z| > p$ implies there is a variable A appearing twice in the derivation, hence the derivation contains a subderivation of the form $A \Longrightarrow vAx \Longrightarrow vwx$. $|vwx|$ is bounded since the derivation is finite, and $A \Longrightarrow vAx$ implies $A \Longrightarrow v^iAx^i \Longrightarrow v^iwx^i$. Since $A \Longrightarrow vwx$ is a subderivation of z, z can be written as uvwxy, and uv^iwx^iy is derivable for all $i \geqslant 0$. Q.E.D.

<u>Theorem</u> 2.5 If L is a context-free language, L is infinite if and only if L contains a word of length greater than p and less than or equal to $p + q$, where p and q are the constants of the preceding theorem.

Proof. If $w \in L$, $|w| > p$, then L is infinite by theorem 2.4. If L is infinite then there is $z = uvwxy$ in L where $|z| > p + q$, and $|vwx| \leq q$, and $uv^iwx^iy \in L$ for all i, by theorem 2.4. Then $uwy \in L$, with $|uwy| > p$. If $|uwy|$ is greater than $p + q$, the procedure can be repeated until a word of length less than or equal to $p + q$ (and greater than p) is found. Q.E.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Corollary   There is an algorithm to decide if a context-free grammar generates a finite or infinite number of words.

Certain classifications are commonly used in connection with context-free languages and grammars:

Definition   1. A grammar G is self-embedding if P contains a production $A \Longrightarrow \alpha_1 A \alpha_2$, where $\alpha_1, \alpha_2 \neq \epsilon$.

2. G is linear if P consists of $A \Longrightarrow uBv$ or $A \Longrightarrow u$ for $A, B \in N$ and $u, v \in T$.

3. G is sequential if N can be ordered such that if $A_i \longrightarrow \alpha$ is in P, then A_j is not in α for $j < i$.

4. L is bounded if $L \subseteq w_1{}^* w_2{}^* \ldots w_k{}^*$ for some k and $w_i \in T$.

5. G is ambiguous if G contains a word with more than one distinct leftmost derivation. A leftmost derivation is one in which the leftmost variable is replaced at each step.

The following theorem gives a sufficient condition for a grammar to generate a regular language. Since the proof is involved it is omitted (see Hopcroft and Ullman, 1969, p. 61).

Theorem 2.6   If G is a non-self-embedding context-free grammar then L(G) is regular.

Definition   A language L is recursive if there is an algorithm which decides whether any word x belongs to L.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Theorem 2.7 If G is context-sensitive then L(G) is recursive.

Proof. An algorithm for deciding if any word x is in L(G), by classifying words in the language according to their minimal derivation length, is given in Hopcroft and Ullman, 1969, p. 17.

4. RECURSIVELY ENUMERABLE LANGUAGES

We now wish to characterize all phrase structure languages as a general class.

Definition A set is recursively enumerable (r.e.) if a finite procedure exists which generates the elements of the set.

The transition function of any Turing machine is a finite procedure, hence a Turing machine language is always r.e. Conversely, recall that by Church's thesis there is a Turing machine corresponding to any finite algorithm.

Thus the following theorem characterizes phrase structure languages as r.e. sets.

Theorem 2.8 If G is any phrase structure grammar, then there is a TM which recognizes L(G). Conversely if any TM accepts a language L, there is a grammar G which generates L.

Proof. The constructions of a TM from a grammar,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and of a grammar given a TM, can be found in Hopcroft and Ullman, 1969, pp. 111-112.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

SYNCHRONOUS DEVELOPMENTAL MODELS

In 1968, A. Lindenmayer introduced systems which model the growth process of one-dimensional cellular arrays (Lindenmayer, 1968). These models are referred to as "Lindenmayer systems" or "developmental systems." Although the initial investigation recognized that these systems were related to automata theory, it concentrated mainly on the biological ramifications. Subsequently mathematicians have been actively studying Lindenmayer systems, for two reasons: first, the systems are interesting mathematically in their own right, from the standpoints of their computing ability and the languages they generate; second, it is possible that results from mathematical linguistics may have significant biological interpretations.

A Lindenmayer system is a linear array of cells. Each cell acts as a finite automaton, with a finite set of states and a (normally deterministic) transition function, δ, receiving an input sequence which, in the most general case, consists of the succession of states through which neighboring cells progress. The cells can change state, according to the transition function, only at discrete points in time, which are the same for all the cells. Hence we can think of the process as being timed by a discrete clock

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

having an arbitrary time interval.

There are three classifications according to the manner in which a cell receives input: in a 2L-system the states of the left and right neighbors of a cell are inputs to the cell; thus the argument of the δ function for each cell consists of the state of the cell and the two adjacent cell states. In a 1L-system a cell receives input only from the cell on its left. A 0L-system is one in which a cell receives no input, and changes state only on the basis of what its present state is. Hence three different types of cellular interactions can be modeled.

A Lindenmayer system, then is a linear array of such cells, all with the same set of possible states and governed by the same δ function. In a 2L-system the two end cells receive only partial inputs, and by convention do not change state. (Alternatively, we can think of the end states as being constant "environmental inputs".) In a 1L-system the left end cell remains constant.

The individual cells differ from ordinary finite automata in that the value of the δ function under certain values of the argument is allowed to be a string of cell states, rather than a single state, indicating cell division. This feature allows a string of cells to grow in length. Where the value of the δ function is the empty word ϵ, cell death is indicated.

Formally an iL-system $(i = 0,1,2)$ is a construct

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(A, α, δ) such that $A = \{s_1, s_2 \ldots, s_n\}$ is a finite non-empty set of cell states, $\alpha \in A^*$ is the starting configuration ($\alpha \geqslant i + 1$), and $\delta: A^{i+1} \longrightarrow \mathcal{P}(A^*)$ is the transition function.

Notation conventions for the δ function are:
For $i = 1$, δ(left input, present state) = next state.
For $i = 2$, δ(left input, present state, right input) = next state.

The argument of the δ function consists of a cell state and inputs to it during a single time interval. The domain of δ can be extended recursively to include a string of cells, and a sequence of inputs instead of a single time interval input:

$i = 0$: $\delta(s_1 \ldots s_n) = \delta(s_1) \quad \delta(s_2 \ldots s_n)$

$i = 1$: $\delta(s, s_1 \ldots s_n) = \delta(s, s_1) \quad \delta(s_1, s_2 \ldots s_n)$

and $\delta(s_1 \ldots s_m, \alpha) = \delta(s_2 \ldots s_m, \delta(s_1, \alpha))$

$i = 2$: $\delta(s\ , s_1 \ldots s_n, s_r) = \delta(s\ , s_1, s_2)$
$\delta(s_1, s_2 \ldots s_n, s)$ and $\delta(s_1 \ldots s_m, \alpha, t_1 \ldots t_m) = \delta(s_2 \ldots s_m,$
$\delta(s_1, \alpha, t_1), t_2 \ldots t_m)$.

An L-system is said to be deterministic if $\delta: A^{i+1} \longrightarrow A^*$; propagating if $\delta: A^{i+1} \longrightarrow \mathcal{P}(A^+)$;[1] and growing if it is propagating and the image of δ contains a string of length greater than 1. Hence strings generated by pro-

[1]Recall that A^+ denotes A^* without ϵ, and $\mathcal{P}(A^+)$ is the set of subsets of A^+.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

pagating systems cannot decrease in length, and growing systems can increase in length.

The set of words produced by an L-system M will be denoted $\mathcal{L}(M)$.

1. EXAMPLES OF L-SYSTEMS

Example 3.1a

As a simple example, consider the 1L-system in which $A = \{0,1\}$, $\alpha = 00$, and δ is specified by the table

		present state	
		0	1
input	0	10	0
	1	1	0

The first eight words preduced are

```
00
010
001
0100
00110
010001
00110100
0100010110
```

If the starting word is changed to 100, the output becomes

```
100
1110
1001
11100
100110
1110001
100110100
11100010110
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

If the starting configuration is 101, the output alternates between 101 and 110. Thus the set of words generated depends strongly on α, as well as the δ function.

Example 3.1b

An example of a unary developmental system is the following: $i = 0$, $A = \{1\}$, $\alpha = 1$, and $\delta(1) = 11$. Then the system generates the language $\{1^{2^n} \mid n \geq 0\}$. A Turing machine recognizing this language was described in Section 1.4.

The remaining examples illustrate special types of developmental patterns in 1L-systems. Lindenmayer has given proofs of the statements specifying the general conditions under which each type of pattern is obtained (Lindenmayer, 1968). In these statements $\lambda(\rho, \sigma)$ is the sequence of states of the rightmost cell of the resulting sequence of strings when ρ is applied to σ. λ therefore can be thought of as a kind of output function.

Example 3.1c

(Linear growth). If $\delta(\rho, \sigma) = \tau\sigma$, $\delta(\rho, \tau) = \tau$ and $\lambda(\rho, \tau) = \rho$, then $\delta(\rho^n, \sigma) = \tau^n\sigma$ for $\rho, \sigma, \tau \in A^*$ and $n \geq 0$.

Let $A = \{0,1\}$, $\alpha = 01$, $\delta(0,0) = 0$ and $\delta(0,1) = 01$. Then the set generated is $\{0^n1 \mid n \geq 1\}$:

```
01
001
0001
00001
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 3.1d

(Banded pattern) If $\delta(\rho,\sigma) = \sigma^m$ and $\lambda(\rho,\sigma) = \rho$, then $\delta(\rho^n,\sigma) = \sigma^{m^n}$, for $\rho,\sigma \in A^*$, m,n > 0.

Let $A = \{0,1\}$, $\alpha = 010$, δ given by the table:

		present state	
		0	1
input	0	1	0
	1	10	1

The output is a series of two alternating repetitious patterns:

```
010
0010
01010
0010010
010101010
0010010010010
01010101010101010
```

Example 3.1e

(Constant apical pattern). If $\delta(\rho,\sigma) = \sigma\tau$, then $\delta(\rho^{n+1},\sigma) = \delta(\rho^n,\sigma)\theta$, where $\theta = \delta(\lambda(\rho^n,\sigma),\tau)$, for $\rho,\sigma,\tau \in A^*$, $n \geqslant 1$.

Thus if $\delta(\rho,\sigma) = \sigma\tau$, then with a starting configuration of $\rho\sigma$ a series of strings is produced in which each string consists of the previous string with an additional new section concatenated (the "θ" mentioned above). The strings appear to be growing only at the right end whereas cell divisions are occurring at several places along the whole length of the string at each step.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As an example let $A = \{0,1\}$, $\alpha = 10$, δ given by:

		present state	
		0	1
input	0	1	0
	1	01	1

Output of the system is:

```
10
101
1010
101001
10100110
10100110101
1010011010100l0
1010011010100100l1001
```

Example 3.1f

(Combined constant apical and banded pattern). Let $A = \{0,1\}$, $\alpha = 0110$, δ given by: $\delta(0,1) = 1$, $\delta(1,1) = 1$, $\delta(1,0) = 0110$. Then the output is:

```
0110
0110110
0110110110110
0110110110110110110110110
```

Lindenmayer has also formulated a scheme for applying these models to branching filaments. These systems are less interesting mathematically since the output of such a system is not a set of words in the language-theoretical sense.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

DEVELOPMENTAL SYSTEM LANGUAGES

Theoretical biologists study formal languages in relation to L-systems from the point of view of discovering rules that model the development of known organisms. On the other hand the interesting questions mathematically are: what type of language does a particular L-system produce, and how general are the different kinds of L-systems in terms of the languages they are capable of generating?

Some basic results concerning the class of all L-systems will be mentioned first, then we will consider the languages resulting from each of the three types of L-systems. Theorems 4.7 and 4.8 give new concise proof constructions characterizing propagating systems. The rest of the theorems bring together known results, for which proof outlines or references to existing proofs are given.

1. SOME BASIC RESULTS

<u>Theorem</u> 4.1 If M is a non-growing L-system, then $\mathcal{L}(M)$ is finite, hence regular.

Proof. If M is non-growing then $\mathcal{L}(M)$ is length-limited. Since there are a finite number of symbols, $\mathcal{L}(M)$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is finite. Q.E.D.

If M is a non-growing OL-system, it is easy to determine the size of $\mathcal{L}(M)$. For each $a \in A$, the transition function has the form $\delta(a) = b$ for some $b \in A$, or $\delta(a) = \epsilon$. Then if α is the starting configuration of M, there is an integer t such that $\delta^t(\alpha)$ does not contain any symbols a_i for which $\delta(a_i) = \epsilon$. That is, the length of $\delta^r(\alpha)$ is constant for $r > t$. If $\delta^t(\alpha) = \epsilon$, then $\mathcal{L}(M)$ has t distinct words. If not, then $\delta^t(\alpha) = \beta = s_1 s_2 \ldots s_n$. For each s_i there is a least integer r_i for which $\delta^{r_i}(s_i) = s_i$. Then if $q = \mathrm{lcm}\{r_1, r_2, \ldots, r_n\}$, $\delta^q(\beta) = \beta$, and $\delta^p(\beta) \neq \beta$ for $p < q$. Hence $\mathcal{L}(M)$ contains $t + q$ distinct words.

<u>Theorem</u> 4.2 (Herman, van Dalen) If M is an iL-system ($i = 0$, 1 or 2) then $\mathcal{L}(M)$ is an r.e. language. Conversely any r.e. language is $\mathcal{L}(M)$ for some 2L-system M.

Proof. The class of 2L-systems, which contains the 1L and OL-systems as subclasses, is equivalent to the class of Turing machines. The constructions for this equivalence are shown in Herman, 1969 or van Dalen, 1971. Theorem 2.8 then applies. Q.E.D.

The following sections will consider the language-theoretical properties of certain subclasses of the class of all L-systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. OL-SYSTEMS

OL-systems are capable of producing finite languages; this happens when the system is non-growing (Theorem 4.1). In this case $\delta(a) \in A$ or $\delta(a) = \epsilon$ for all $a \in A$. It is also possible for a OL-system to grow initially but be length-limited, and therefore have a finite language, as in the case:

$$A = \{0,1,2,3\}, \quad \alpha = 01,$$
$$\delta(0) = 0, \quad \delta(1) = 02, \quad \delta(2) = 03, \quad \delta(3) = 0$$
$$\mathcal{L}(M) = \{01, 002, 0003, 0000\}.$$

<u>Theorem</u> 4.3 The set of deterministic OL languages has a non-empty intersection with the class of regular languages.

Proof. The machine described above provides an example. Q.E.D.

An example of a regular OL-system which is not length limited is: $A = \{0, 1\}$, $\alpha = 0$, $\delta(0) = 10$, $\delta(1) = 1$. Then this system's language is the one corresponding to the regular expression (1^*0).

A terminal symbol or state of a OL-system (A, α, δ) is a symbol $a \in A$ such that $\delta(a) = \{a\}$. A non-terminal symbol is one that does not have this property.

<u>Theorem</u> 4.4 (Lindenmayer, 1968) If M is a OL-system (A, α, δ) such that for all $a \in A$, $\delta(a) = t$ or $\delta(a) = tb$, where b is a non-terminal and t is a terminal or ϵ, then

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\mathcal{L}(M)$ is regular.

Proof. Given $M = (A, \alpha, \delta)$ satisfying the hypothesis, then $A = P \cup Q$ where P is the set of non-terminals in A and Q is the set of terminals. Construct the grammar $G = (N,T,P,S)$ where

$$N = \{ [a] \mid a \in P \}$$
$$T = A = P \cup Q$$

and the productions of P are:

1. $S \longrightarrow \alpha$
2. $\{ [a] \longrightarrow t \mid t \in Q \text{ and } \delta(a) = t \}$
3. $\{ [a] \longrightarrow tb \mid \delta(a) = tb \}$
4. $\{ [a] \longrightarrow a \mid [a] \in N \}$.

Then for any word $w \in \mathcal{L}(M)$, G derives w from α by imitating the δ function of M. Furthermore any word derivable in G is a word of $\mathcal{L}(M)$. Thus $\mathcal{L}(M) = \mathcal{L}(G)$, and the theorem follows since G is regular. Q.E.D.

An illustrative example is provided by the deterministic system mentioned earlier which generates 1^*0.

<u>Theorem</u> 4.5 The set of deterministic OL languages has a non-empty intersection with the set of non-regular context-free languages, and with the set of non-context-free languages.

Proof. 1. Let M be the machine specified by: $A = \{0,1\}$, $\alpha = 101$, $\delta(0) = 101$, $\delta(1) = 1$. Then $\mathcal{L}(M)$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is the set $\{1^n01^n \mid n \geq 1\}$, which is context-free since it is generated by the productions $S \longrightarrow 1S1$, $S \longrightarrow 0$.

2. (van Dalen) Let $M = (A, \alpha, \delta)$ with $A = \{a\}$, $\alpha = a$, $\delta(a) = aa$. Then $\mathcal{L}(M) = \{a^{2^n} \mid n \geq 0\}$. A language $\{a^i \mid i \in A\}$ is context-free only if A is an ultimately periodic index set (Ginsburg, 1966, p. 86), hence $\mathcal{L}(M)$ is not context-free. Q.E.D.

The following theorem gives a sufficient condition for a non-deterministic OL model to have a context-free language.

<u>Theorem</u> 4.6 (Lindenmayer, 1971, p. 482) If $M = (A, \alpha, \delta)$ is a OL-system such that $a \in \delta(a)$ for all $a \in A$, then $\mathcal{L}(M)$ is context-free.

Proof. Suppose $M = (A, \alpha, \delta)$ satisfies the hypothesis. For any $w = a_1a_2 \ldots a_n \in A^*$, define $[w] = [a_1][a_2] \ldots [a_n]$, and $[\epsilon] = \epsilon$.

Let $G = (N,T,P,S)$ where

$$N = \{[a] \mid a \in A\}$$
$$T = A$$

and P consists of

1. $S \longrightarrow \alpha$
2. $\{[a] \longrightarrow [w] \mid \delta(a) = w \text{ in } M\}$
3. $\{[a] \longrightarrow a \mid a \in A\}$.

Then if $\beta \in \mathcal{L}(M)$, G derives a word $[\beta]$ using rules 1 and 2, then rule 3 obtains β from $[\beta]$. Conversely

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

if $\gamma \in \mathcal{L}(G)$, any substitution of type 2 used in deriving γ from α can be simulated by M using the corresponding function transition on the subword which is replaced, and the identity transition on the rest of the symbols in the word. Hence $\mathcal{L}(M) = \mathcal{L}(G)$, which is context-free. Q.E.D.

The example in the proof of Theorem 4.5 (1) shows that the hypothesis is not a necessary condition.

An open question is: if $\mathcal{R}$ is an arbitrary context-free language is there a OL-system M such that $\mathcal{R} = \mathcal{L}(M)$?

3. 1L-SYSTEMS

1L-systems model developmental situations in which information passes in one direction along the array of cells.

Since the 1L-systems contain the OL-systems as special cases, Theorems 4.3 and 4.5 apply to 1L-systems.

Definition A left context-sensitive grammar is a context-sensitive grammar in which P consists of rules of the form $\alpha\beta \longrightarrow \gamma$ where $\alpha \in T^*$ and $\beta \in N^*$. The following theorem states that the class of left context-sensitive languages contains the 1L languages.

Theorem 4.7 If M is a propagating 1L-system, $\mathcal{L}(M)$ is left context-sensitive.

Proof. We will construct a grammar which generates

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the language of any given 1L-system. The simplest grammar makes use of endmarkers (#) on both ends of a string. If $M = (A, \alpha, \delta)$ where $A = \{a_1, a_2, \ldots, a_n\}$, let $G = (N,T,P,S)$ where

$$N = \{V_1, V_2, \ldots, V_n\} \cup \{\#\}$$
$$T = A = \{a_1, \ldots, a_n\}$$

and P consists of the rules

$$S \longrightarrow \# \alpha \#$$
$$a_i\# \longrightarrow V_i\#$$
$$a_iV_j \longrightarrow V_i\beta \quad \text{where } \beta \in \delta(a_i, a_j)$$
$$\#V_i \longrightarrow \#a_i.$$

The endmarkers are not considered to be part of a word derivable by G.

To show $\mathcal{L}(G) = \mathcal{L}(M)$, suppose α is the string $t_1t_2 \ldots t_m$. Then let $\delta(\alpha) = t_1\beta_{12}\beta_{23} \ldots \beta_{m-1,m}$ where $\beta_{ij} \in \delta(t_i, t_j)$. Now in G there is a derivation of $\delta(\alpha)$:

$$S \longrightarrow \#t_1t_2 \ldots t_m\# \longrightarrow \#t_1 \ldots t_{m-1}V_m\# \longrightarrow \#t_1 \ldots V_{m-1}\beta_{m-1,m}\#$$
$$\longrightarrow \#t_1 \ldots V_{m-2}\beta_{m-2,m-1}\beta_{m-1,m}\# \longrightarrow \cdots \longrightarrow$$
$$\#V_1\beta_{12} \ldots \beta_{m-1,m}\# \longrightarrow \#t_1\beta_{12} \ldots \beta_{m-1,m}\#.$$

Similarly starting with $\delta(\alpha)$, there is a derivation in G of any word which M can produce from $\delta(\alpha)$. By induction then, $\mathcal{L}(M) \subseteq \mathcal{L}(G)$. On the other hand, a reverse argument shows that if $\alpha \overset{*}{\Longrightarrow} w$ in G, then there is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a sequence of steps of M which generates w from α. Hence $\mathcal{L}(M) = \mathcal{L}(G)$, which is left context-sensitive. Q.E.D.

It would be possible to avoid end markers in the grammar at the cost of increasing the number of variables in N, by using special variables to stand for the end cells.

<u>Example</u> 4.3a Consider the system of Example 3.2d. For this case $G = (N,T,P,S)$ where $N = \{V_0, V_1, \#\}$, $T = \{0,1\}$, and P consists of the rules given in the theorem. The derivation of the first three words by G is as follows:

$S \longrightarrow \#010\# \longrightarrow \#01V_0\# \longrightarrow \#0V_110\# \longrightarrow \#V_0010\# \longrightarrow \#0010\#$
$\#0010\# \longrightarrow \#001V_0\# \longrightarrow \#00V_110\# \longrightarrow \#0V_0010\# \longrightarrow \#V_01010\#$
$\longrightarrow \#01010\#$.

It should be mentioned that the grammar in this theorem is more interesting mathematically than biologically, since many substitutions of the grammar are required to simulate a single time interval step of the 1L-system, and hence the mechanics of the grammar do not offer any new insight into the biological operation of the system.

4. 2L-SYSTEMS

2L-systems are the most complex type of developmental system since the cells can interact in both directions, but they are often the most natural type to use in constructing certain models.

The analog of Theorem 4.7 for 2L-systems now follows.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

<u>Theorem</u> 4.8 If M is a propagating 2L-system, $\mathcal{L}(M)$ is context-sensitive.

Proof. (This theorem was also proved by van Dalen using a more complicated grammar. See van Dalen, 1971.)

Again we will use a grammar with end markers. If $M = (A, \alpha, \delta)$, let $G = (N,T,P,S)$ where

$$N = \{ V_{ij} \mid i,j = 1,2,\ldots,n \}$$
$$T = A = \{ a_1, a_2, \ldots, a_n \}$$

and the productions in P are:

$$S \longrightarrow \# \alpha \#$$
$$\#a_i a_j \longrightarrow \#a_i V_{ij}$$
$$V_{ij} a_k \longrightarrow \beta_{ijk} V_{jk} \quad \text{where } \beta_{ijk} \in \delta(a_i, a_j, a_k)$$
$$V_{ij}\# \longrightarrow a_j\#$$

If $\alpha = t_1 t_2 \ldots t_m$, then $\delta(\alpha)$ is a word of the form $t_1 \beta_{123} \cdots \beta_{m-2,m-1,m} t_m$. There is a derivation in G of this word as follows:

$$S \longrightarrow \#t_1 t_2 \ldots t_m\# \longrightarrow \#t_1 V_{12} t_3 \ldots t_m\# \longrightarrow \#t_1 \beta_{123} V_{23} \ldots t_m\#$$

$$\longrightarrow \cdots \longrightarrow \#t_1 \beta_{123} \beta_{234} \cdots \beta_{m-2,m-1,m} V_m\#$$

$$\longrightarrow \#t_1 \beta_{123} \beta_{234} \cdots \beta_{m-2,m-1,m} t_m\# = \delta(\alpha).$$

Then if $\delta^2(\alpha)$ is any word following $\delta(\alpha)$ in $\mathcal{L}(M)$, there is a similar derivation of it in G starting with $\delta(\alpha)$. Also any word derivable by G can be produced by M using the corresponding δ function transitions. Hence by induction $\mathcal{L}(M) = \mathcal{L}(G)$, which is context-sensi-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tive since G is a context-sensitive grammar. Q.E.D.

Note that $|\alpha| \geqslant 2$ for M and G to be defined.

Example 4.4a Suppose $M = (A, \alpha, \delta)$ where $A = \{0,1\}$, $\alpha = 010$, and δ is the transition function below:

present state		right input	
	0	0	1
left input	0	00	1
	1	1	0

present state		right input	
	1	0	1
left input	0	11	1
	1	00	0

Then $\delta(\alpha) = 0\,\delta(010)0 = 0110$

$\delta^2(\alpha) = 0\,\delta(011)\,\delta(110)0 = 01000$

Applying G,

$S \longrightarrow \#010\# \longrightarrow \#0V_{01}0\# \longrightarrow \#0\ (010)V_{10}\# \longrightarrow \#011V_{10}\#$
$\longrightarrow \#0110\#$

$\#0110\# \longrightarrow \#0V_{01}10\# \longrightarrow \#01V_{11}0\# \longrightarrow \#0100V_{10}\# \longrightarrow \#01000\#.$

We now obtain a further characterization in terms of linear bounded automata (see Sec. 1.4).

Theorem 4.9 If M is a deterministic propagating 2L-system then $\mathcal{L}(M)$ is recognized by a deterministic linear bounded automaton M'.

Proof. This construction is an extension of that of Hopcroft and Ullman, 1969, p. 116. M' has a tape containing three tracks. The input string to be recognized (β) is placed on track 1 (with end markers).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Suppose $M = (A, \alpha, \delta)$ with $\alpha = a_1a_2 \ldots a_m$. M' goes through the following procedure.

1. Enters α onto track 2 with a_1 in the leftmost cell.

2. Reads a_1, a_2, a_3 and replaces a_2 with $\delta(a_1,a_2,a_3)$, shifting $a_3 \ldots a_m$ to the right if necessary. M' stores a_2 in its internal control.

3. For each consecutive triple $a_{i-1}a_ia_{i+1}$ M' replaces a_i with $\delta(a_{i-1},a_i,a_{i+1})$, stores a_i, shifts $a_{i+1} \ldots a_m$ to the right as far as necessary, and proceeds to the next triple $a_ia_{i+1}a_{i+2}$. (This procedure, continued until the right end of the string is reached, imitates a single transition of the 2L-system M.)

4. If this operation (steps 2 and 3) would cause a_m to be shifted onto the square occupied by the right end marker, M' halts and rejects.

5. After a_{m-1} is replaced by $\delta(a_{m-2},a_{m-1},a_m)$, M' then compares track 1 and track 2 square by square. If they are identical M' halts and accepts.

If the track 2 word is shorter than track 1, M' repeats the transition routine (steps 2 and 3) starting at the left of the existing track 2 word, and derives a new word.

6. If the strings on tracks 1 and 2 are the same length but not identical, M' first copies the string on track 2 onto track 3. It then returns to the left of track

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2 and repeats steps 2 through 5.

7. The procedure of steps 2 through 5 is repeated until either (a) tracks 1 and 2 are identical and M' accepts, (b) the right end limit of track 2 is exceeded and M' rejects, (c) track 2 again becomes identical to track 3. In the latter case M' halts and rejects.

M' is constructed to simulate the grammar presented in the previous theorem, so any string it computes on track 2 must be a word in $\mathcal{L}(M)$; in fact the sequence of words derived on track 2 is identical to the sequence generated by M. Since this sequence increases monotonically in length (because M is propagating) then all words in $\mathcal{L}(M)$ of length equal to $|\beta|$ occur consequtively, and there is a finite number of these. Hence if the derivation on track 2 reaches a point at which its length would exceed that of track 1, without ever matching, then β cannot be a member of $\mathcal{L}(M)$.

Steps 6 and 7 of the construction are included in case $\mathcal{L}(M)$ does contain more than one word having the same length as the input word.

Since M' operates according to a well-defined algorithm, and the δ function is deterministic, M' is deterministic. This theorem includes as special cases the classes of deterministic OL and 1L-systems. Q.E.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5. SUMMARY

We have seen that the class of languages produceable by Lindenmayer developmental systems is restricted because of the requirement of simultaneous replacement. However it is a difficult problem to determine exactly what languages they are capable of producing, and many of the results obtained to date are "intersection" theorems rather than equivalence or containment ones (although all these types have been mentioned here).

As we have seen, a deterministic OL-system (DOL-system) can be regular, but the class of regular languages they can model is probably quite limited, as shown by the fact that even a^* is not a DOL language (since a DOL-system must increase in length monotonically, and cannot do so linearly with a single letter alphabet). Theorem 4.4 shows that it is much "easier" for non-deterministic OL-systems to produce regular languages than deterministic ones. In fact if a OL transition function is constructed randomly the chances are that it will be non-context-free.

With regard to constructing a system to have a predetermined language, the examples have shown that the systems with interaction (1L and 2L) are more flexible and permit more variety than the OL-systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

COMPUTER SIMULATION OF DEVELOPMENTAL SYSTEMS

Computer programs that simulate L-systems are an aid in determining the language of a particular L-system, especially with systems having a large alphabet. A common problem, for example, is to see how the language corresponding to a fixed transition function varies for different initial configurations. Programs are given here that simulate deterministic OL, 1L and 2L systems, along with examples illustrating their use. Some of the examples are not complex enough to warrant computer analysis, but are used to show how the programs are applied.

The programs are written in the SNOBOL 4 language, which is a string manipulation system and hence well suited for this type of application, but is comparatively slow and requires a large amount of computer memory. This language allows the programs themselves to be quite short.

1. OL-SYSTEMS

Example 5.1

Figure 1a gives a program to simulate any OL-system which has an alphabet $A = \{1,2,3,4\}$. The input data consists of the transition function matrix, the initial string,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and the number of words to be outputted (in addition to the initial word). The program works from left to right, examining each character in the current string and replacing it by its successor according to the transition matrix.

Input data is entered following the END statement, in the order: $\delta(a_1)$, $\delta(a_2)$, ... $\delta(a_n)$, α, number of words; each on a separate line.

Simulation of a OL-system with a different alphabet requires only a simple modification (statements 4-7).

The output of the program in Figure 1a is the first five words of the system below:

$A = \{1,2,3,4\}$, $\alpha = 1234$, with transition function:

	present state			
	1	2	3	4
successor	11	22	33	44

The nth word in the language of this simple system is $1^{2^{n-1}}2^{2^{n-1}}3^{2^{n-1}}4^{2^{n-1}}$. This exponentially increasing language probably has no realistic biological application, but provides an example of one type of (context-sensitive) language that OL-systems are capable of producing.

Figure 1b gives the SNOBOL statistics for this example.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. 1L-SYSTEMS

Example 5.2a

The program for 1L-systems, shown in Figure 2 is similar to the one for OL-systems except that it examines pairs of characters, and works right to left, similar to the operation of the 1L grammar of Section 4.3. The program in Figure 2 incorporates the data for a specific system into the main part of the program, although this is not necessary (see next example).

The particular system in this example, like the previous one, has as its language strings consisting of four equal length bands, increasing monotonically in length. However the 1L-system by virtue of cell interactions is able to model this type of growth at a linear, rather than exponential rate, and so is more realistic as a biological model.

The data for this system are $A = \{1,2,3,4\}$, $\alpha = 1234$, with transition matrix

		present state			
		1	2	3	4
input	1	1	12	-	-
	2	-	2	23	-
	3	-	-	3	344
	4	-	-	-	4

The nth word in the language is $1^n 2^n 3^n 4^n$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 5.2b

Figure 3 presents a general program for 1L-systems, in the form accepting the alphabet $\{1,2,3\}$. This program has the ability to repeat the simulation for more than one starting configuration, with a fixed δ function. The order for entering input data after the END statement is: $\delta(1,1)$, $\delta(1,2)$, $\delta(1,3)$, $\delta(2,1)$, ... , $\delta(3,3)$, number of words (same for each case), α_1, α_2, ... , α_n.

Figure 3 illustrates the use of this program in simulating a linearly growing, repeating, banded pattern. The bands remain constant in length here, in contrast to the previous examples. The data are: $A = \{1,2,3\}$, $\alpha = 211$, δ given by:

		present state		
		1	2	3
input	1	2	2	-
	2	11	3	3
	3	1	-	1

Example 5.2c

The following system can produce several different languages, depending on α (see Figure 4): $A = \{1,2,3\}$, δ given by

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		present state		
		1	2	3
	1	3	22	33
input	2	11	1	2
	3	11	3	2

$\alpha_1 = 12, \alpha_2 = 21, \alpha_3 = 13, \alpha_4 = 121$

The four languages are all of the "constant apical" type (Section 3.2e). There are at least three different possible patterns, as the first three sets show. The fourth set is the same pattern as in the first one, but the strings grow faster in the fourth set.

3. 2L-SYSTEMS

Example 5.3a

The program imitating 2L-systems (Figure 5) operates on the same principle as the 2L grammar given in Section 4.4, and as shown her accepts data consisting of 0's and 1's. The order of entering the data is the same as for the 1L case, with the order for the δ function shown in statements 4-10 of the program.

Figure 5 shows the first 11 words for the system; $A = \{0,1\}$, $\alpha = 11111$, δ given by

present state	0	right input 0	right input 1
left input	0	0	1
	1	1	1

present state	1	right input 0	right input 1
left input	0	11	11
	1	0	0

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 5.3b

The biologists' interest in L-systems is to discover what different kinds of naturally occurring growth they are able to model. One specific phenomenon which occurs commonly in nature is length-limited growth, in which a filament grows to a predetermined length and remains at that length in a dynamic state; i.e. cells continue to divide and die even after the full length is reached. As an example of a more complex developmental system requiring computer aided analysis, we will construct an L-system which models this phenomenon and present some sample simulation runs.

The specific problem to be considered is to construct a system starting with a short initial configuration, producing strings which increase linearly up to a certain length and then remain at that length; and with the additional feature that if at any time the current string is "cut," i.e. a right-hand section removed, the string will regrow out to the limiting length.

One way to model a length-limited filament is to have the first few cells in the string act as a counter, in conjunction with a special cell which divides at each clock time. When a certain count is reached the dividing cell is replaced by a non-dividing one. This method allows one to set the limiting length at any desired number. However such a device would not have the "regrowth" feature.

A model that has this property is presented in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6. The model is a 2L-system with a ten-symbol alphabet, so that the transition function is relatively complex. The program realization (Figure 7) consists of the basic 2L-system program followed by an implementation of the function as a series of predicate statements. Figure 8a shows the first 50 strings.

The basic operation is as follows:

1. At every fourth clock time a signal is created which moves right one position at each time interval.
2. When the signal reaches the right end of the string it is reflected and becomes a left-moving signal.
3. The left-moving signal keeps a count of the number of right-moving signals it has crossed.
4. When a left-moving signal that has crossed five right signals reaches the left end, cell division is stopped, but the system continues to send out a signal on every fourth word.
5. If after growth has stopped a left-moving signal reaches the left end with a count smaller than five, cell division begins again.

Thus the length of the string is kept constant in a type of dynamic equilibrium, after the initial growth. If

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

part of the string is removed, growth is resumed. When growth again stops, the length of the string will be equal to or close to its former length. This is illustrated in Figure 8b, which shows the results of applying the program to the first five symbols in the final string of Figure 8a. (This number is selected at random.) A "0" is added at the right end, and may be thought of as an environmental input. The final length is now 22, compared to 19 for the original growth. When the first ten symbols of the final string in Figure 8a are used as the starting configuration, the result is as shown in Figure 8c.

The computer can thus be an indispensible aid in constructing and analyzing complex models. Certain functional differences could be effected in all three basic programs; for example we might wish the simulation to stop when a certain string length is exceeded, or we might want to print out only every third or fourth string. The programs given here are basic ones that can be modified to fit given situations.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-10

1                       &TRIM = 1; &ANCHOR = 1
3                       D = ARRAY(4)
4                       D<1> = INPUT; D<2> = INPUT
6                       D<3> = INPUT; D<4> = INPUT
8                       STR = INPUT
9                       NUM = INPUT
10      L1              OUTPUT = STR
11                      M = LT(M,NUM) M + 1               :F(END)
12                      X =
13      L2              STR   X LEN(1) . A  =  X D<A>     :F(L1)
14                      X = X D<A>                        :(L2)
15      END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
1234
11223344
1111222233334444
11111111222222223333333344444444
1111111111111111222222222222222233333333333333334444444444444444
```

```
NORMAL TERMINATION AT LEVEL  0
LAST STATEMENT EXECUTED WAS   11
```

Fig. 1a. Program for simulation of OL-systems with A={1,2,3,4}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
SNOBOL4 STATISTICS SUMMARY-
              797 MS. COMPILATION TIME
             2549 MS. EXECUTION TIME
              147 STATEMENTS EXECUTED,       5 FAILED
                4 ARITHMETIC OPERATIONS PERFORMED
               64 PATTERN MATCHES PERFORMED
                2 REGENERATIONS OF DYNAMIC STORAGE
                6 READS PERFORMED
                5 WRITES PERFORMED
               36 K CORE USED,     4195 FREE WORDS LEFT
            17.34 MS. AVERAGE PER STATEMENT EXECUTED
```

Fig. 1b. Program statistics for example 5.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-10

1                    &TRIM = 1; &ANCHOR = 1
3                    D = ARRAY('4,4')
4                    D<1,1> = 1; D<2,2> = 2; D<3,3> = 3; D<4,4> = 4
8                    D<1,2> = 12; D<2,3> = 23; D<3,4> = 344
11                   STR = 1234
12                   NUM = 8
13                   PAT = TAB(*(I - N)) . X  LEN(1) . A  LEN(1) . B
14         L1        OUTPUT = STR
15                   M = LT(M,NUM) M + 1      :F(END)
16                   I = SIZE(STR)
17                   N = 1
18         L2        N = LT(N,I) N + 1                  :F(L1)
19                   STR  PAT = X A D<A,B>              :(L2)
20         END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
1234
11223344
111222333444
1111222233334444
11111222223333344444
111111222222333333444444
1111111222222233333334444444
11111111222222223333333344444444
111111111222222222333333333444444444
```

Fig. 2. Program for simulation of example 5.2a.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-1Ø

1                  &TRIM = 1; &ANCHOR = 1
3                  D = ARRAY('3,3')
4                  D<1,1> = INPUT; D<1,2> = INPUT; D<1,3> = INPUT
7                  D<2,1> = INPUT; D<2,2> = INPUT; D<2,3> = INPUT
1Ø                 D<3,1> = INPUT; D<3,2> = INPUT; D<3,3> = INPUT
13                 NUM = INPUT
14       LØ        STR = INPUT                :F(END)
15                 M = Ø
16                 PAT = TAB(*(I - N)) . X LEN(1) . A LEN(1) . B
17       L1        OUTPUT = STR
18                 M = LT(M,NUM) M + 1        :F(L3)
19                 I = SIZE(STR)
2Ø                 N = 1
21       L2        N = LT(N,I) N + 1                  :F(L1)
22                 STR  PAT = X A D<A,B>              :(L2)
23       L3        OUTPUT =
24                 OUTPUT =                   :(LØ)
25       END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
211
2112
21122
211223
2112233
21122331
211223311
2112233112
21122331122
211223311223
2112233112233
21122331122331
211223311223311
2112233112233112
21122331122331122
211223311223311223
2112233112233112233
```

Fig. 3. Program for simulation of 1L-systems with A = {1,2,3}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
12
122
1221
122111
12211133
1221113333 2
122111333322 23
1221113333222311 2
1221113333222311211322
12211133332223112113221133331
1221113333222311211322113333111333222 11
```

```
21
211
2113
211333
21133322
2113332231
2113332231211
211333223121122113
21133322312112211322111333
211333223121122113221113333111333322
211333223121122113221113333111333322211333322231
```

```
13
133
1332
13323
133232
1332323
13323232
133232323
1332323232
13323232323
133232323232
```

```
121
12211
1221113
1221113333
1221113333222
1221113333222311
12211133332223112113
122111333322231121132211333
122111333322231121132211333311133322
12211133332223112113221133331113332221133332231
1221113333222311211322113333111333222113333223111133322231211
```

Fig. 4. Simulation of example 5.2c.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
SNOBOL4 (VERSION 3.4.3, JAN. 16, 1971)

DIGITAL EQUIPMENT CORP., PDP-1Ø

                  &ANCHOR = 1; &TRIM = 1
                  D = ARRAY('Ø:1,Ø:1,Ø:1')
                  D<Ø,Ø,Ø> = INPUT;  D<Ø,Ø,1> = INPUT
                  D<1,Ø,Ø> = INPUT;  D<1,Ø,1> = INPUT
                  D<Ø,1,Ø> = INPUT;  D<Ø,1,1> = INPUT
                  D<1,1,Ø> = INPUT;  D<1,1,1> = INPUT
                  STR = INPUT
                  NUM = INPUT
                  PAT = *X LEN(1) . A LEN(1) . B LEN(1) . C
        L1        OUTPUT = STR
                  M = LT(M,NUM) M + 1              :F(END)
                  X  =
                  STR  PAT = A D<A,B,C> B C
                  X  =  A
        L2        X  =  X D<A,B,C>
                  STR  PAT = X D<A,B,C> B C        :S(L2)
                  STR  RTAB(2) . P LEN(1) LEN(1) . Q = P Q :(L1)
        END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
11111
1ØØØ1
11Ø11
1Ø1111
1111ØØ1
1ØØØ111
11Ø111Ø1
1Ø111ØØ11
1111ØØ11111
1ØØØ1111ØØØ1
11Ø111ØØØ1Ø11
```

Fig. 5. Program for simulation of 2L-systems with A = {0,1}.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$A = \{0,1,2,3,4,5,6,7,8,9\}$

$(02x) = 31 \quad (x \neq 7,9)$

$(03x) = 4 \quad (x \neq 7,9)$

$(04x) = 58 \quad (x \neq 7,9)$

$(05x) = 2 \quad (x \neq 7,9)$

$(x8y) = y+1 \quad (1 < y \leq 6)$

$(x87) = 7$

$(x88) = 9$

$(x80) = 2$

$(x81) = 1 \quad (x \neq 9)$

$(981) = 8$

$(8xy) = 8 \quad (1 \leq x \leq 7)$

$(88x) = 1$

$(047) = 5$

$(049) = 9$

$(49x) = 8 \quad (x = 1,7)$

$(49x) = 18 \quad (2 \leq x \leq 5)$

$(x9y) = 9 \quad (x \neq 4,\ y = 1,7)$

$(x9y) = 1 \quad (x \neq 4,\ y \neq 1,7)$

$(09x) = 2$

$(0xy) = x \quad 1 \ (x = 2,3,\ y = 7,9)$

$(05x) = 2 \quad (x = 7,9)$

$(97x) = 1$

$(x7y) = 9 \quad (2 \leq x \leq 5)$

$(x1y) = y \quad (x \neq 8,\ 1 \leq y \leq 7)$

$(xyz) = 1 \quad (x \geq 2,\ 2 \leq y \leq 6)$

$(1xy) = 1 \quad (x \geq 2)$

$1 \leq x,y,z \leq 9$ except as indicated.

Figure 6. δ function for length-limited 2L-system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
-UNLIST
          &ANCHOR = 1; &TRIM = 1
          DEFINE('N(X,Y,Z)')
          STR = INPUT
          NUM = INPUT
L1        OUTPUT = DUPL(' ',20) STR
          M = LT(M,NUM) M + 1        :F(END)
          X =
          STR  X LEN(1) . A LEN(1) . B LEN(1) . C = A N(A,B,C)
+         B C
          X = A
L2        X = X N(A,B,C)
L3        STR  X LEN(1) . A LEN(1) . B LEN(1) . C      :F(L4)
          NEXT = N(A,B,C)
          STR  X A = X NEXT
          X = X NEXT          :(L3)
L4        STR  RTAB(2) . P LEN(1) LEN(1) . Q = P Q    :(L1)
N         N = EQ((A B C),111) 1    :S(RETURN)
          EQ(B,8)             :F(R1)
          N = GE(C,2) LE(C,6) C + 1         :S(RETURN)
          N = EQ(C,Ø) 2       :S(RETURN)
          N = EQ(C,7) 7       :S(RETURN)
          N = EQ(C,8) 9       :S(RETURN)
          N = NE(A,9) 1       :S(RETURN)
          N = 8               :S(RETURN)
R1        EQ(B,1)             :F(R2)
          N = NE(A,8) NE(C,8) NE(C,Ø) C     :S(RETURN)
          N = NE(A,8) 1       :S(RETURN)
R2        N = EQ(A,8) 8       :S(RETURN)
          EQ(A,Ø)             :F(R4)
          N = GE(B,5) 2       :S(RETURN)
          LT(C,7)             :F(R3)
          N = EQ(B,2) 31      :S(RETURN)
          N = EQ(B,3) 4       :S(RETURN)
          N = EQ(B,4) 58      :S(RETURN)
R3        N = NE((B C),49) B + 1    :S(RETURN)
          N = 9               :S(RETURN)
R4        EQ(B,9)             :F(R6)
          EQ(A,4)             :F(R5)
          N = GE(C,2) LE(C,6) 18    :S(RETURN)
          N = 8               :S(RETURN)
R5        N = GE(C,2) LE(C,5) 1     :S(RETURN)
          N = 9               :S(RETURN)
R6        N = EQ(A,1) 1       :S(RETURN)
          EQ(B,7)             :F(R7)
          N = EQ(A,9) 1       :S(RETURN)
          N = 9               :S(RETURN)
R7        N = LE(B,6) 1       :S(RETURN)
          N = B               :(RETURN)
END
Ø4911Ø
5Ø
```

Figure 7. Program for example 5.3b.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Ø2Ø
Ø31Ø
Ø41Ø
Ø581Ø
Ø218Ø
Ø3112Ø
Ø4121Ø
Ø58211Ø
Ø23811Ø
Ø311181Ø
Ø41111 8Ø
Ø5811112Ø
Ø2181121Ø
Ø31118211Ø
Ø41113811Ø
Ø581131181Ø
Ø218311118Ø
Ø31148111 12Ø
Ø41411811 21Ø
Ø58411118211Ø
Ø25811113811Ø
Ø3111811311 81Ø
Ø41111831111 8Ø
Ø5811114811112Ø
Ø2181141181121Ø
Ø311184111182 11Ø
Ø411158111138 11Ø
Ø58115118113118 1Ø
Ø2185111183111 18Ø
Ø311681111481111 2Ø
Ø4161181141181121Ø
Ø58611118411118211Ø
Ø27811115811113811Ø
Ø3918115118113118 1Ø
Ø491185111183111 18Ø
Ø981168111148111 12Ø
Ø2886118114118112 1Ø
Ø39781118411118211Ø
Ø4911811581111381 1Ø
Ø98111851181131181Ø
Ø2881168111831111 8Ø
Ø3918611811481111 2Ø
Ø4917811184118112 1Ø
Ø98711811581118211Ø
Ø27811185118113811Ø
Ø39181168111831181Ø
Ø49118611811481118Ø
Ø98117811184118112Ø
Ø28871181158111821Ø
Ø39781118511811381Ø
Ø49118116811183118Ø
```

Fig. 8a. Simulation output (a) for example 5.3b.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Ø49110
Ø98110
Ø28810
Ø39180
Ø49120
Ø98210
Ø23810
Ø311180
Ø411120
Ø5811210
Ø2182110
Ø31138110
Ø41311810
Ø583111180
Ø248111120
Ø3111811210
Ø4111182110
Ø58111138110
Ø21811311810
Ø311183111180
Ø411148111120
Ø5811411811210
Ø2184111182110
Ø31158111138110
Ø41511811311810
Ø585111183111180
Ø268111148111120
Ø3111811411811210
Ø4111184111182110
Ø58111158111138110
Ø21811511811311810
Ø311185111183111180
Ø411168111148111120
Ø5811611811411811210
Ø2186111184111182110
Ø31178111158111138110
Ø41711811511811311810
Ø587111185111183111180
Ø278111168111148111120
Ø391811611811411811210
Ø491186111184111182110
Ø981178111158111138110
Ø288711811511811311810
Ø397811185111183111180
Ø491181168111148111120
Ø981118611811411811210
Ø288117811184111182110
Ø391871181158111138110
Ø491781118511811311810
Ø987118116811183111180
Ø278111186118114811120
```

Fig. 8b. Simulation output (b) for example 5.3b.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Ø4911811680
Ø981118612Ø
Ø2881178210
Ø391871381Ø
Ø4917831180
Ø98714811 2Ø
Ø278411821Ø
Ø395811381Ø
Ø411183118Ø
Ø58111481120
Ø21814118210
Ø311181113810
Ø411118131180
Ø58111111811120
Ø21811111181210
Ø311181111118110
Ø411118111111810
Ø5811111811111180
Ø2181111181111120
Ø31118111111811210
Ø41111811111182110
Ø581111118111138110
Ø21811111811311810
Ø311181111183111180
Ø411118111148111120
Ø5811111181141181121Ø
Ø2181111184111182110
Ø31118111158111138110
Ø41111811511811311810
Ø581111118511118311118Ø
Ø218111116811114811112Ø
Ø3111811611811411811210
Ø41111861111841111182110
Ø58111117811115811113811Ø
Ø21811171181151181131181Ø
Ø311187111185111183111180
Ø411178111168111148111120
Ø5811711811611811411811210
Ø2187111186111184111182110
Ø3117811117811115811113811Ø
Ø417118117118115118113118 1Ø
Ø587111187111185111183111180
Ø278111178111168111148111120
Ø3918117118116118114118112 1Ø
Ø491187111186111184111182110
Ø981178111178111158111138110
Ø2887118117118115118113118 1Ø
Ø397811187111185111183111180
Ø491181178111168111148111120
Ø981118711811611811411811210
Ø288117811118611118411118211Ø
```

Fig. 8c. Simulation output (c) for example 5.3b.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

Aho, A. and Ullman, D. 1968. "The Theory of Languages". *Mathematical Systems Theory* 2: 97-125.

Chomsky, N. 1956. "Three Models for the Description of Language". *IRE Trans. Information Theory* 2: 113-124.

Doucet, P. and Rozenberg, G. 1971. "On OL Languages". *Information and Control* 19: 302-318.

Ginsburg, S. 1966. *The Mathematical Theory of Context-Free Languages*. New York: McGraw-Hill.

Herman, G. 1969. "The Computing Ability of a Developmental Model for Filamentous Organisms". *J. Theoretical Biology*. 25: 421-435.

Herman, G. 1970. "The Role of Environment in Developmental Models". *J. Theoretical Biology* 29: 329-342.

Hopcroft, J. and Ullman, J. 1969. *Formal Languages and Their Relation to Automata*. Reading, Mass.: Addison-Wesley.

Kleene, S. 1956. "Representation of Events in Nerve Nets and Finite Automata". *Automata Studies*, Princeton Univ. Press, Princeton, N.J.: 3-42.

Kuroda, S. 1964. "Classes of Languages and Linear Bounded Automata". *Information and Control* 7: 207-220.

Lindenmayer, A. 1968. "Mathematical Models for Cellular

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Interactions in Development". _J. Theoretical Biology_ 18: 280-315.

Lindenmayer, A. 1971. "Developmental Systems Without Cellular Interactions: Their Languages and Grammars". _J. Theoretical Biology_ 30: 455-484.

McCulloch, W. and Pitts, W. 1943. "A Logical Calculus for the Ideas Immanent in Nervous Activity". _Bull. Math. Biophysics_ 5: 115-133.

Myhill, J. 1957. "Finite Automata and the Representation of Events". Wright Air Development Center Technical Report 57-624.

Rabin, M. and Scott, D. 1959. "Finite Automata and Their Decision Problems". _IBM J. Research and Development_ 3: 115-125.

Springsteel, F. 1972. "Language Recognition by Marking Automata". _Information and Control_ 20-4: 313-330.

Turing, A. 1936. "On Computable Numbers with an Application to the Entscheidungsproblem". _Proc. London Mathematical Society_ 2-42: 230-265.

Van Dalen, D. 1971. "A Note on Some Systems of Lindenmayer". _Math. Systems Theory_ 5: 128-140.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.