Open House in Unusual Automata Theory January 1972

Brian H. Mayoh (ed.)

DAIMI PB - 15
June 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF AARHUS
Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark

INTRODUCTION

The Aarhus Open House was intended as an informal forum for the exchange of ideas on recent developments in automata theory and its applications. In order to give a fair impression of those recent developments that were discussed, these proceedings include more than a reprinting of some of the formal papers that were presented.

However, no report can convey the lively interaction between the participants that developed during the course of the open house. The success of the meeting was due both to the invited participants and to the many local staff members who were involved in so many ways.

Brian H. Mayoh

CONTENTS

Unusual Automata Theory, January 1972

List of participants:

Dr. Giorgio Ausiello — Jan. 9-22.

Istituto per le Applicationi del Calcolo
Consiglio Nazionale delle Richerche
Piazzale delle Scienze 7
00185 Roma
Italia

Francois Bancilhon — Jan.17-28.

Institut de Recherche d'Informatique et d'Automatique
Domaine de Voluceau
78 Rocquencourt
France

Dr. Paul G. Doucet — Jan. 12-21.

Filosofisch Instituut
Heidelberglaan 2
"De Uithof"
Utrecht
Holland

Professor Gabor T. Herman — Jan. 9-26.

Department of Computer Science
State University of New York/Buffalo
4226 Ridge Lea Road
Amherst, N.Y. 14226
USA

Pauline Hogeweg — Jan. 12-23.

Filosofisch Instituut
Heidelberglaan 2
"De Uithof"
Utrecht
Holland

Dr. Günther Hotz — Jan. 11-18.

Universität des Saarlandes
Saarbrücken
Bundesrepublik Deutschland

Herbert Kopp — Jan. 16-29.

Universität des Saarlandes
Saarbrücken
Bundesrepublik Deutschland

list of participants cont.:

Professor Aristid Lindenmayer Jan. 9-15.

Filosofisch Instituut
Heidelberglaan 2
"De Uithof"
Utrecht
Holland

Jan van Leeuwen Jan. 22-28.

Mathematisch Instituut
Budapestlaan
"De Uithof"
Utrecht
Holland

Professor John Myhill Jan. 9-15.

School of Mathematics
University of Leeds
Leeds LS2 9TJ
England

Professor Azaria Paz Jan. 9-28.

Technion
Israel Institute of Technology
Technion City
Haifa
Israel

Martti Penttonen Jan. 9-28.

Department of Mathematics
University of Turku
20500 Turku 50
Finland

Professor Bernd Reusch Jan. 9-29.

Institut für angewandte Mathematik und Informatik
der Universität Bonn
53 Bonn
Wegelerstrasse 6
Bundesrepublik Deutschland

Professor Arto Salomaa Jan. 9-28.

Department of Mathematics
University of Turku
20500 Turku 50
Finland

Paul Vitanyi	Jan. 9.28.

Foundation Mathematisch Centrum
2e Boerhaavestraat
49 Amsterdam (O)
Holland

Adrian Walker	Jan. 9-28.

SUNY at Buffalo
4226 Ridge Lea Road
Amherst, N.Y. 14226
USA

List of lectures:

January 11:

Gabor T. Herman:	'Polar Organisms with Apolar Individual Cells'
Gabor T. Herman:	'CELIA - A CEllular Linear Iterative Array Simulator Program'

January 12:

Azaria Paz:	'Introductory Stochastic Automata and Lnaguages'
A. Lindenmayer:	(1) 'Propagating, deterministic OL-Systems with locally catenative formulas'
	(2) 'L-Systems with interactions'

January 13:

Bernd Reusch:	'On linear and Partial Linear Realization of Automata'
A. Salomaa:	'Growth functions of DPOL-systems'
J. Myhill:	'Unprovability of the induction schema in the ramified Principia'

January 14:

G.T. Herman & A. Lindenmayer:	'Use of automata theory in the simulation of developmental processes'
J. Myhill:	'Intuitionistic Set Theory'

January 17:

A. Salomaa:	'Time-varying automata and grammars'
	'Programmed Grammars'
A. Paz:	'Nonclosure and Unsolvability Results for Stochastic Languages'

January 18:

A. Paz:	'Nonclosure and Unsolvability Results for Stochastic Languages'
G. Ausiello:	'Abstract approach to computational complexity'

January 19:

A. Salomaa:	'Control Languages'
	'Ordered Grammars'
P. Hogeweg:	'Generation of Branching-Patterns in 2L-Systems'
P.G. Doucet:	'The growth of word length in DOL-Systems'

January 20:

A. Paz:	'Word function of Markov chains'
F. Bancilhon:	'Necessary conditions for Automata Interchangeability'

January 21:

P. Vitanyi:	'DOL-Systems: letters and their propagations, the word pattern and numerical values involved in finite languages'
G. Ausiello:	'Unusual abstract computational complexity'

January 22:

B. Reusch:	'On linear and partial linear realization of automata, part II'

January 24:

A. Paz:5	'Word function of Markov chains'
J. van Leeuwen:	'Rule labeled programs - a generalization of the notion of a context-free grammar'
B. Reusch:	'Survey on recent Results on the Input-Semigroup of Linear Automata'

January 25:

A. Salomaa:	'Ordered Grammars'
	'Different approaches to probabilistic grammars'
	'Linear-space automata'
G.T. Herman:	'Synchronization fo growing cellular arrays'
B. Reusch:	'Survey of Recent Results ... '

January 26:

A. Paz:	'Decomposition of Stochastic Automata'
F. Bancilhon:	'A Geometrical Model for a Stochastic Automaton'

January 27:

A. Salomaa:	see January 25.
P. Vitanyi:	'Sexually Reproducing Cellular Automata'

Matematisk Institut Aarhus Universitet
DATALOGISK AFDELING
Ny Munkegade - 8000 Aarhus C - Danmark
Tlf. 06-12 83 55

Unusual Automata Theory

List of books in open house library:

Arbib, M.A.	: Theories of Abstract Automata
Claus, V.	: Stochastische Automaten
Codd, E.F.	: Cellular Automata
Conway, J.H.	: Regular Algebra and Finite Machines
Salomaa, A.	: Theory of Automata
Tou, J.T.	: Applied Automata Theory
Paz, A.	: Introduction to Probabilistic Automata
Banks, E.R.	: Information Processing and Transmission in Cellular Automata
Burks, A.W.	: Essays on Cellular Automata
Caianiello, E.R.	: Automata Theory
Wiener, N. & Schadé, J.P.	: Cybernetics of the Nervous System

J. Computer System Sci. 4, (1970)
Rozprawy Matematyczne, 31-40, (1963-64)
Proceedings in Applied Mathematics, Vol. XIV
Mathematical Theory of Automata, Symposium Proceedings Vol. XII
1969 ACM Symposium on Theory of Computing
1970 Second Annual ACM Symposium on Theory of Computing
1966 Seventh Annual Symposium on Switching and Automata Theory

/km

Matematisk Institut Aarhus Universitet
DATALOGISK AFDELING
Ny Munkegade - 8000 Aarhus C - Danmark
Tlf. 06-12 83 55

List of books in open house library: (continued)

Minsky, Marvin L.	: Computation: Finite and Infinite Machines
Hennie, F.C.	: Finite-State Models for Logical Machines
von Neumann, J.	: Theory of Self-Reproducing Automata
von Neumann, J.	: The Computer and the Brain
Ross Ashby, W.	: An Introduction to Cybernetics
Harrison, M.A.	: Lectures on Linear Sequential Machines
Gill, Arthur	: Introduction to the Theory of Finite-State Machines

BIOLOGICAL AUTOMATA BIBLIOGRAPHY

Lindenmayer, A. : Developmental Systems without Cellular Interactions, their languages and grammars
(J. Theor. Biol. 30 (1971), 455-484)

Lindenmayer, A. : Mathematical Models for Cellular Interactions in Development
(J. Theor. Biol. 18 (1968), 280-315)

Herman, G. T. : Computing Ability of a Developmental Model for Filamentous Organisms
(J. Theor. Biol. 25 (1969), 421-435)

Herman, G. T. : Role of Environment in Developmental Models
(J. Thoer. Biol. 29 (1970), 329-341

van Dalen, D. : A note on some Systems of Lindenmayer
(Math. Sys. Th. 5 (1971), 128-140)

Rozenberg, G. &
Doucet, P. G. : On OL-Languages
(Inf. Control 19 (1971) 302-318)

CELLULAR AUTOMATA BIBLIOGRAPHY

k: Concentrated introduction to the subject
s: Self-reproducing automata
g: Garden of Eden problem

[1]: Codd, E.F. : Cellular Automata, kap. 1-2 (k), kap. 3-8 (s)
(ACM Monograph Series, Academic Press 1968)

[2]: Burks, W. : Essays on Cellular Automata
Essay 1 (s)
Essay 6-7 (g)
(University of Illinois Press 1970)

[3]: Banks, E.R. : Information Processing and Transmission in Cellular Automata (s)
(MAC TR-81 MIT Jan. 1971)

[4]: Winograd, T. : A simple Algorithm for Self-Replication (s)
(MIT Memo No. 197)

[5]: Amoroso, S. & Cooper, G. : The Garden of Eden Theorem for Finite Configurations (g)
(Proc. of the Amer. Math. Soc. 26, 1970, pp. 158-164)

[6]: Yamada, H. & Amoroso, S. : A Completeness Problem for Pattern Generation in Tessalation Automata
(J. Comput. System Sci. 4, 1970, pp. 137-176)

[7]: Gardner, M. : On Cellular Automata, self-reproducing, The Garden of Eden and the Game "life" (s, g)
(Sci. Amer. Fber. 1971, pp. 112-117)

[8]: Gardner, M. : The Fantastic Combinations John Conway's Solitaire Game "life".

STOCHASTIC AUTOMATA BIBLIOGRAPHY

The references marked *) are recommended as introducing - the references are arranged chronologically.

*) Robin, M.O.: : Prabab ilistic Automata
Information and Control 6 - 1963 - p. 230-245.

Carlyle, I.W. : Reduced Forms for Stochastic Sequential Machines.
I. Math. Ann. Appl. 7 - 1963 - p. 167-175.

Salomaa, A : On probabilistic Automata with one input-letter.
Ann. Univ. Turku - Ser. AI85, 1965.

Even : Comments on the Minimization of Stochastic Machines. IEEE - Trans. on EC. - 14 - 1965 - p. 634-677.

*) Paz, A. : Some Aspects of Probabilistic Automata.
Information and Control 9 - 1966 - p. 26-60.

Page, C.V. : Eqvivalence between probabilistic and deterministic seqvential Machines.
Information and Control 9 - 1966 - p. 469-520.

Salomaa, A. : On Events represented by Probabilistic Automata of Different Types.
Canad. J. Math. 1966 - p. 242-251.

McLaren, R.W. : A Stochastic Automaton Model for the Synthesis of Learning Systems.
IEEE Trans, on Sys. Sci. and Cyb. - ssc 2,
2 - 1966 - p. 109-114.

TuraKeinen, P. : On non-regular Events representable in Probabilistic Automata with one input-letter.
Ann. Univ. Turku. - Ser. AI 90. 1967.

Salomaa, A. : On m-adic probabilistic Automata.
Information and Control lo, 1967 - p. 215-219.

Paz, A. : Minimization Theorems and Techniques for Seqvential Stochastic Machines.
Information and Control 11 - 1967 - p. 155-166.

Paz, A. : Fuzzy Star Functions, Probabilistic Automata and their Approximation by non-probabilistic Automata.
Journal of Comp. and Sys. Sci. 1 - 1967 - p. 371-390.

Paz, A. : Homomorplisms between Stochastic Seqvential Machines and related problems.
Math. Sys. Theory. 2 - 1968 - p. 223-245.

TuraKeinen, P. : On Stochastic Languages.
Information and Control 12 - 1968 - p. 304-313.

ChandraseKaran, B : Stochastic Automata Games.
IEEE Trans. on Sys. Sci. and Cyb. - ssc 5, 2, 1969 - p. 145-149.

Arbib, M. : "Theories of Abstract Automata". 1969.
Chapter 9: "Stochastic Automata".

*) Salomaa, A. : "Theory of Automata" - 1969
Chapter 2: "Finite non-deterministic and Probabistic Automata".

Ellis, C. : Probabilistic Languages and Automata.
Sec Ann. ACM Symp. on th. of Comp. 1970

Bancillon, F. : Dispersion metrices and Stochastic Automata Morphisms. Int. Symp. on the Th. of Mach. and Comp. - Haifa - 1971.

Paz, A. : Introduction to Probabilistic Automata.
Ac. Press. 1971.

GRAMMARS AND AUTOMATA WITH CONTROL DEVICES BIBLIOGRAPHY :

Abraham, S. : Some questions of phrase structure grammars
(Computational Linguistics 4 (1965), 61-70.)

Cohen, R.S. &
Nash, B.O. : Parallel leveled grammars
(IEEE Conf. Record of 1969 Tenth Ann. Symp.
on Switching and Automata Theory, 263-276.)

Friant, J. : Grammaires Ordonnées - grammaires matricielles
(Université de Montréal (1968), 43 pp.)

Fris, I. : Grammars with partial ordering of the rules
(Information and Control 12 (1968), 415-425)

Ginsburg, S. &
Spanier, E.H. : Control sets on grammars
(Math. Systems Theory 2 (1968), 159-177.)

Greibach, S. &
Hopcroft, J. :Scattered context grammars
(J. Comput. System Sci. 3 (1969) 233-247.)

Rosenkrantz, D. J. : Programmed Grammars and classes of formal languages
(J. Assoc. Comput. Mach. 16 (1969), 107-131.)

Salomaa, A. : On grammars with restricted use of productions
(Ann. Acad. Sci. Fennicæ. Ser. A I 454 (1969), 32 pp.)

- On the index of a context-free grammar and language
(Information and Control 14 (1969), 474-477)

- Periodically time-variant context-free grammars
(Information and Control 17 (1970), 294-311.)

Salomaa, A. : Probalistic and weighted grammars
(Information and Control 15 (1969), 529-544.)

– : Theory of Automata
(Pergamon Press (1969), 276 pp.)

– : On some families of formal languages obtained by regulated derivations
(Ann. Acad. Sci. Fennicae, Ser. A I 479 (1970).)

Stotskij, E. D. : Some restrictions on derivations in context-sensitive grammars
(Akad. Nauk SSSR Nauchno-Techn. Inform. Ser. 2 (1967), 35-38 (Russian).)

– : The notion of index in generalized grammars
(Ibid. (1969), 16-17, Russian.)

– : Generative grammars with regulated derivations
(Ibid. (1968), 28-31, Russian.)

14

PROBLEMBOOK

UNUSUAL

AUTOMATA

THEORY

JANUARY 1972

UNIVERSITY OF AARHUS
DENMARK

1) Is the family of languages generated by context-sensitive grammars with a context-free control language equal to the family of context-sensitive languages?

2) Is there a context-free language L such that every context-free grammar for L has a non-context-free Szilard-language?

3) Given a language L, in what cases is it possible to decide whether L is a Szilard-language of some grammar?

4) Give a characterization for the family of languages generated by context-free ordered grammars.

5) Is there a PD2L-growth function which is not a PD1L-growth function? If there is, study the same problem in terms of the Rozenberg (k, l)-hierarchy.

6) Can one decide the growth equivalence of two PD1L-systems? (For PD0L- and even for D0L-systems a decision procedure can be given.)

Arto Salomaa

Conjecture

Only regular languages over one letter are generated by context-free matrix grammars (original definition).

Arto Salomaa

Is there any relation between

M^{λ} the class of languages generated by context-free matrix grammars,

M_{ac} the class of languages generated by context-free λ-free matrix grammars with appearance checking?

Is either of these the same as

M the class of languages generated by context-free λ-free matrix grammars?

Is the language $\{a^{2^n} \mid n \geq 1\}$ in any of these classes?

Arto Salomaa

Is it decidable whether two propagating, deterministic OL-systems generate the same language?

More simply, when do they generate the same sequence of strings?

A. Lindenmayer

1) Is it decidable whether two context-free grammars generate the same sentential forms? (A sentential form is any word derivable from the initial letter, not only words over the terminal alphabet.) This problem is closely related to the decision problems of L-systems.

2) Develop a theory of probabilistic L-systems combining the ideas in probabilistic automata and grammars together with the ideas in L-systems.

Arto Salomaa

Exercise (solution available from G. T. Herman)

Give a symmetric 2-L system, which, starting from the string 'ab', generates a sequence of strings of length 2, 3, 5, 8, 13, ... (the Fibonacci sequence).

i) Give an algorithm which, for any regular expression R, finds the number σ (R), where

σ (R) = (μ n) (there is a OL-system with n letters which generates the language denoted by R).

(ii) Give an algorithm which, for any regular expression R, produces a OL-system with σ (R) letters, which generates the language denoted by R.

(iii) Do (i) and (ii), but with context-free grammars in place of regular expressions.

G. T. Herman

Lineare Automaten

Sei $\delta : X \times Z \to Z$ die Zustandsfunktion eines linearen Automaten. Ist $\delta(x, z_{x_0}) = z_{x_0}$, dann ist

$$h(z) = z + z_{x_0}$$

ein Isomorphismus des autonomen Automaten mit $x = 0$ auf den autonomen Automaten mit dem konstanten Eingang $x = x_0$.

{In Fall δ nicht singulär existiert für jedes x genau ein Fixpunkt z_x, so dass alle autonomen "Faktoren" isomorph sind.}

In wie weit ist diese Eigenschaft für lineare Automaten charakteristisch?

G. Hotz

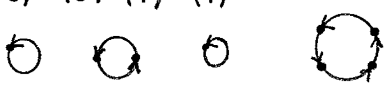
{ } falsch. Siehe: B. Reusch, Lineare Automaten
Bibliographisches Inst., Hochschulskripten 708
1969, p. 75-80.

Gegenbeispiel:

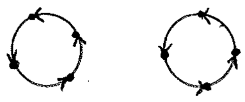
M linear über GF(2).

δ (s, x) = As + Bx mit $A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ $B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

für die Eingaben $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ ergibt sich folgender Graph:



für die Eingaben $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ ergibt sich folgender Graph:



B. Reusch

Give an example (or prove there is none) of a recursively enumerable stochastic language which is not context-sensitive.

A. Paz - A. Salomaa

The regular languages, which can be defined on the m-adic machine using rational cut points, have only 2 states. What do they lose in exchange for the saved states?

due to Paz

Exercise:

Given a natural number n, find recursion formulas which cannot be satisfied by any DOL-sequence over an n-letter alphabet.

P.G. Doucet

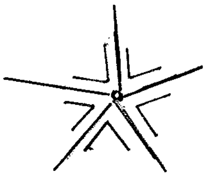
Gegeben: beliebiger Graph

Gesucht: abstrakter endl. Automat mit folgenden Eigenschaften:

1) Die Anzahl seiner Ein- und Ausgänge ist gleich der maximalen Anzahl von Kanten, die von irgendeinem Punkt der Graphen ausgehen.

2) Setzt man i jeden Punkt des Graphen ein Exemplar dieses Automaten und verbindet man Ein- und Ausgänge, die vorhandenen Kanten des Graphen entsprechen, dann berechnet dieses Automatennetz eine Einbettung des Graphen in eine 2-dim. Mannigfaltigkeit seines Geschlechtes.

Erläuterung: Jeder Automat kann eine zyklische Anordnüng der Kanten seines Punktes anzeigen; d.h. er definiert einen " ebenen Stern". Diese Sterne sind so beschaffen, dass sie eine 2-dim. Man. sein. Geschlechtes definieren.



Gibt es einen abstrakten Automaten, der dies für alle Graphen mit beschränktem Grad leistet?

Günter Hotz

Consider the class of D2L-grammars producing strings which stabilize at a certain length.

Make some reasonable assumptions about maximal production length (e.g., 2) and axiom length (e.g., 1) and then find the maximum stable length as a function of the number of letters in the alphabet.

after V.I. Varshavsky

Given linear automata M_0 , M_1 , M_2 by $[A_i , B_i , C_i , D_i]$ i = 1, 2, 3, such that $A_0 : \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$, $B_0 = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$

How are their semigroups H_0; H_1; H_2 related?

If you have semigroups in that relation and you know that H_1 and H_2 are the semigroups of some linear automata, is H_0 also the semigroup of some linear automaton?

B. Reusch

1) Are scattered context languages equal to context sensitive languages?

2) Are left context-sensitive languages (context sensitive with empty word as right context) equal to context sensitive languages?

3) Can each context-sensitive language be generated by a grammar with left context-sensitive and permutative (XY → YX) productions?

Martti Penttonen

Invarianten von formalen Systemen

Arithmetische Ausdrücke a haben einfache Invarianten: Man nehme den Homomorphismus φ der allen Zeichen ausser den Klammern (, { , [,...,), } ,] vergisst. φ (a) kann in der freien Gruppe mit den Erzeugenden (, { , [,... und mit $(^{-1} =)$, $\{^{-1} = \}$, $[^{-1} =]$ auf 1 gekürzt werden.

Gibt es weitere einfache Invarianten für andere Klassen kontextfreier Chomskysprachen ?

(Die charakteristische Funktion ist natürlich eine Invariante, aber eine uninteressante, da die Invarianten der Berechnung dieser Funktion vorgreifen sollen).

Man denke hierbei an den Satz von Schützenberger - Chomsky : L kontextfreie Chomskysprache . ⇒

Es gibt Dycksprache D und reguläres Standartereignis S und Hom. φ mit

$$\varphi\ (D \cap S) = L.$$

D entspricht dem Klammerwort von oben !

Günter Hotz

(1) Find an algorithm (or Function), other than the trial or error method, which for every integer $n > 0$ yields an arbitrary number m of pairwise relative primes $\bar{k}_1, \ldots, \bar{k}_m$ and a positive integer d (which may be zero) such that

$$\prod_{i=1}^{m} \bar{k}_i + d = n \quad \text{and}$$

$$\sum_{i=1}^{m} \bar{k}_i + d \text{ is minimal.}$$

(2) The same as (1) but $\prod_{i=1}^{m} \bar{k}_i + d \geq n$

ad (1) $\sum_{i=1}^{m} \bar{k}_i + d$ is the minimal number of letters needed to generate a finite DOL-language of exactly size n.

ad (2) $\sum_{i=1}^{m} \bar{k}_i + d$ is the minimal number of letters needed to generate a finite DOL-language of at least size n.

Paul Vitanyi

Give an algorithm which will do the following:

Input: A finite set S of finite ordered sequences of words over a finite alphabet Σ.

$$S = \{ < p_{1,0}, p_{1,1}, \dots, p_{1,n_1} >, < p_{2,0}, p_{2,1}, \dots, p_{2,n_2} >, \dots, < p_{m,0}, p_{m,1}, \dots, p_{m,n_m} > \}$$

Output: A D1L-system $\underline{L}$ such that, for $1 \leq i \leq m$, $1 \leq j \leq n_i$

$$p_{i,j-1} \overset{+}{\underset{\underline{L}}{\Rightarrow}} p_{i,j}$$

Such a system $\underline{L}$ may not exist, in which case the algorithm should produce an output indicating the non-existenxe of $\underline{L}$.

[This problem, and similar problems, appeared in the Feliciangeli & Herman paper, which is in the folder in Biological Automata. Of the 36 problems mentioned in that paper, there are 6, including the one described above, which are still open.]

G.T. Herman

Consider the set $C(Z)$ of polynomials with coefficients in Z and leading coefficient 1, and its subset $C(Z^-)$ (with leading coefficient 1 and the others non-positive).

Problems:

1) If $f \in C(Z)$, find a $g \in C(Z)$ such that $fg \in C(Z^-)$.

2) Find necessary and/or sufficient conditions for the existence of such a g (given f).

Comment:

- one necessary condition (for problem 2) is that f be not of the form $f(x) = x^n + \alpha_{n-1} x^{n-1} + \ldots + \alpha_2 x^2 + \alpha_1 x - \alpha_0$ (with all $\alpha_i \in N$).
- a DOL-language is locally catenative only if (not : if) such a g exists for the characteristic polynomial of its associated production matrix. All such matrices define a set of polynomials which is smaller than $C(Z)$ (and to which the f in the example does not belong).

P. G. Doucet

Given two DOL- (or PDOL-) grammars G and $\hat{G}$, is it possible to say whether for some k & x $\hat{w}_k = w_k x$? (or vice-versa, $w_k = \hat{w}_k y$).

due to G. Rozenberg

Very hard problem:

$f : \{0,1\}^n \to \{0,1\}$ is a "Boolean Function".

Def: f is a thresholdfunction : $\Leftrightarrow \left(\bigvee\limits_{t,\, a_n ,\, \ldots ,\, a_n \in R} (f(x_1 \ldots x_n) = 1 \Leftrightarrow \Sigma a_i\, x_i \geq t)\right)$

Def: f is k-asumable : $\Leftrightarrow \left(\bigwedge\limits_{A_1 \ldots A_k ,\, B_1 \ldots B_k \in \{0,1\}^n} (f(A_i) = 1,\right.$

$$f(B_i) = 0,\ i=1 \ldots k \Rightarrow \sum_{i=1}^{k} A_i \neq \sum_{i=1}^{k} B_i))$$

addition of vectors
over the reals!

Theorem (well-known) : f is a thresholdfunction $\Leftrightarrow \bigwedge\limits_{k \in N}$ f is k-asumable.

Theorem: if $n \leq 8$, then

f is a thresholdfunction $\Leftrightarrow$ f is 2-asumable.

proof: by enumeration (!!)

Conjecture: For every n there is a m(n) such that

f is a thresholdfunction $\Leftrightarrow \bigwedge\limits_{k \leq m(n)}$ f is k-asumable

Problem: Prove the conjecture and give some bound for m(n).

B. Reusch

There was a young man from Netherlands,
Who tried to simulate elephants,
He thought they were OL
Or even DOL
But they always turned out to be malignant.

There was a young man from Buffalo,
Who~~se~~ had firing squads on the go,
He tridd growing them,
And synchronizing them,
But he never got past LGO.

G. T. [illegible]
Poet Laurate of
Gov. Rockefeller

COMPUTATIONAL COMPLEXITY:
Main results and a commentary.

Giorgio Ausiello
Istituto per le Applicazioni del Calcolo del C.N.R.
Roma, Italy

Summary

0. Theories of computational complexity.

The theory of computational complexity has been developed as an attempt to answer basic questions both on the side of all days programming exigencies and on the side of logical foundations of computing processes: for example, "Given two functions, which one is easier to compute?" "Given two algorithms for the same function, which of them runs faster?".

Though those questions appear to be very natural questions, even to state them properly we first need to make clear what we mean as "easy to compute" , "running time", etc. In connection to this pqint we may find a wide range of approaches to the problems of computational complexity.

A. Analysis of algorithms: here the notion of complexity is defined in terms of elementary operations required to perform a particular processing of particularly structured data. For example, operations on numbers in binary notation, matrix multiplication, sorting sequences of integers, etc. In this field the problems are to find efficient (or even optimal) techniques of encoding structured data and of performing the required transformations and, at a first level of abstraction, to find classes of problems that are of equivalent degree of complexity in terms of a given complexity measure.

We will not deal with this approach, but, in order to give an idea of the kind of work people are doing in this area we give a short bibliography at the end of this paragraph.

B. Subrecursive classes. Under this name we will consider many various attempts to give a hierarchy of classes of primitive recursive functions in which, according to some a-priori definitions of complexity, functions falling in class C_i are "easier" than the functions falling in class $C_{i+1} - C_i$. In paragraph 1 we will give a brief survey of the work that has been done in this area.

C. Time and tape bounded abstract machines. Working with a particular abstract machine, very natural notions of time and space required to compute functions, immediately arise. Using these notions as complexity measures, important results have been achieved both in providing a classification of primitive recursive functions and in showing properties of more general complexity classes. Paragraph 1 will be devoted also to this approach.

D. Axiomatic computational complexity. This approach is interested in giving a very weak notion of what we can consider a complexity measure and in establishing results that hold for all such measures. For this reason this is often called a machine independent approach. Paragraph 2 will be denoted to the basic results in this area while paragraph 3 will give a survey of a few subjects on which research in abstract computational complexity has been recently carried on.

E. Information theoretic approach to computational complexity. From the point of view of the information content of a program the complexity of a string w of length n is roughly given by the length of the shortest program that given n as an input gives w as an output. Relations between highly complex strings and the notion of randomness have been investigated. A very recent paper in this area is:

Schnorr, C.P. A unified approach to the definition of random sequences, Journal of Mathematical System Science Sept. 1971 and from its bibliography most of the important papers in this area can be retraced.

An extensive bibliography on the whole field of computational complexity has been written by Marek I. Irland and Patrick C. Fisher, University of Waterloo, Dept. of Applied Analysis and Computer Science CSRR 2028 October 1970.

A bibliography of analysis of algorithms:

- D.E. Knuth, The art of computer programming, Addison Wesley.
- S. Winograd, On the time required to perform addition, J. ACM 12 (1965).
- S. Winograd, On the time required to perform multiplication, J. ACM 14 (1967).
- S. Winograd, On the number of multiplications required to compute certain functions, Proc. Nat.Acad. of Sciences 58, 5 (1967).
- V. Strassen, Gaussian elimination is not optimal, unpublished, Angewandte Mathematik der Universität, Zürich, (Dec.1968).
- M. P. Spira, The time required for group multiplication, J. ACM 16 (1969).
- P. M. Spira, On the computation time of certain classes of Boolean functions, ACM Symposium on Theory of Computing, Marina del Rey, Calif.(May 1969).
- J. E. Hopcroft, L. B. Kerr, On minimizing the number of multiplications necessary for matrix multiplication, Tech. Rep. N. 69-44, Department of Computer Science, Cornell University (Sept. 1969).
- M. O. Rabin, On proving the simultaneous positivity of linear forms,Third ACM Symposium on Theory of Computing Shaker Heights, Ohio.
- A. Borodin, Some results on the computation of polynomial forms, An Int. Symp. on the Theory of Machines and Computations, Haifa, Israel (1971).

- J. Morgenstern, On linear algorithms, An Int. Symp. on the Theory of Machines and Computations, Haifa, Israel (1971).
- I. Munro, Problems related to matrix multiplication, Symposium on Computational Complexity, N.Y.U. (1971).
- P. M. Spira, On the complexity of linear algorithms, Symposium on Computational Complexity, N.Y.U.(1971)
- S. A. Cook, The complexity of theorem proving procedures Twelfth Annual Symposium on Switching and Automata Theory, East Lansing, Michigan (1971).

1. Subrecursive classes.

In this paragraph we will deal with the notion of complexity as it is related to the structure of the definition of a function or to the specific resource that is needed to compute it on an abstract machine.

1.1 Classes based on growth rate.

The first rough classification of functions has been introduced since the early days of recursive functions theory.

The broadest class of computable functions is the class $\mathcal{R}$ of partial recursive functions. They are exactly those functions that can be computed by a turing machine, a Markov algorithm, a register machine or any other kind of mechanical device.

They can be defined in terms of base functions and operators. The set of base functions we need to define the class $\mathcal{R}$ are the following (we will refer to them as base functions from now on):

zero funtion: $0^n(x_1, \ldots, x_n) = 0 \qquad n \geq 0$

successor function: $S(x) = x + 1$

identity functions: $U_i^n(x_1, \ldots, x_n) = x_i \qquad 1 \leq i \leq n$

The operators that allow us to define new p.r. functions in terms of other p.r. functions are the following (let X be the argument vector)

composition: given $h, g_1, \ldots g_m$

$$f(X) = h(g_1(X), \ldots, g_m(X))$$

primitive recursion: given h, g

$$f(X, 0) = g(X)$$

$$f(X, y+1) = h(X, y, f(X, y))$$

minimalization : given g

$f(X) = \mu y\,(g(X,y) = 0)$

where μy means "the least y such that ..." and g must be a total function.

Hence the class $\mathcal{R}$ is the smallest class that contains the base functions and is closed under composition, primitive recursion, minimalization.

The class $\mathcal{R}$ contains functions whose growth rate is extremely fast, such as the Ackermann function. A smaller class of functions, all of which are total, is the class $\mathcal{P}$ of primitive recursive functions: $\mathcal{P}$ is the smallest class that contains the base functions and is closed under composition and primitive recursion.

Primitive recursion, though, is still a powerful operator that allow us to define functions that grow too fast. The elementary functions (Kalmar) are a proper subset of $\mathcal{P}$ and we might say that practiçally in all problems of every day life we deal only with this kind of functions. The class $\mathcal{E}$ of elementary functions is the smallest class that contains as initial functions the base functions, addition and substraction and is closed under composition, limited summation (given g, $f(X,y) = \sum_{i \leq y} g(X,i)$) and limited multiplication (given g, $f(X,y) = \prod_{i \leq y} g(X,i)$).

It can be easily shown that elementary functions do not have a growth rate faster than exponential.

The classification of recursive functions in general, primitive and elementary recursive functions is not satisfactory because it is too weak; for this reason people interested in classifying recursive functions have been working mainly on this point: finding an infinite set of classes of primitive recursive functions ordered by inclusion according to some intuitive notion of complexity.

The first notion that has been used is growth rate. Andrzey Grzegorczyck in 1953 gave the first results in this direction. We will express them using the formulation that has been given by R. Ritchie in 1965.

Let us define the following inffinite ordered set of functions:

$f_0(x, y) = x + 1$

$f_1(x, y) = x + y$

$f_2(x, y) = x \cdot y$

and for $n \geqslant 3$

$f_n(x, o) = 1$

$f_n(x, y + 1) = f_{n-1}(x, f_n(x,y))$

This set of functions is often called "spine" of the hierarchy of classes of functions that we are going to define. It is easy to see that, roughly speaking, each function is the iteration$^{(*)}$ of the preceding one. This is the same idea that has been used by Ackermann in order to define a recursive function that grows faster than any primitive recursive function.

The way to define classes based on the growth rate is, now, to put in each class one function of the spine but to forbid iteration, that is we are allowed to use primitive recursion in order to add functions to the class, provided the new function grows less than some functions already in the class.

More precisely, we say f is defined by limited recursion on g, h, j if

$$f(X, o) = g(X)$$
$$f(X, y+1) = h(X, y, f(X,y))$$
$$f(X, y) \leqslant j(X, y)$$

Then we define $\mathcal{E}_n$ to be the (n + 1)-th Grzegorczyck class if $\mathcal{E}_n$ is the smallest class that contains as initial functions the base functions and f_n, and is closed under composition and limited recursion.

Basic properties of Grzegorczyck classes are the following:

i) $\forall n \quad \mathcal{E}_n \subsetneq \mathcal{E}_{n+1}$

ii) $\bigcup_{n=0}^{\infty} \mathcal{E}_n = \mathcal{P}$

iii) $\mathcal{E}_3 = \mathcal{E}$

iv) $\forall_n \geqslant 2 \quad \forall$ one argument function $g \in \mathcal{E}_n \; f_{n+1}(x,x) \geqslant g(x)$

v) $\forall_n \geqslant 3 \quad \mathcal{E}_{n+1}$ contains a universal function for $\mathcal{E}_n$

Nevertheless, the most important property of the Grzegorczyck classification is its stability with respect to different intuitive notions of complexity as we will see in the next paragraphs.

(*)- We say that f is defined by iteration on g if $f(x) = g^x(x)$

1.2 Classes based on depth of nesting of iteration.

In 1963 Paul Axt introduced the idea of measuring the complexity of a primitive recursive function in terms of number of times the operator of primitive recursion is used in a nested way.

Accordingly, the first level of Axt hierarchy K_0, is made of all functions that are defined by composition of the base functions; K_1 is made of all functions that are defined using the primitive recursion operator at most once etc. In general a function f is in K_{n+1} if it is obtained either by composition of other functions of K_{n+1} or by primitive recursion of functionsin K_n.

In order to give a more neat formulation of this approach, in 1968 Dennis Ritchie introduced the so-called loop programs.

Essentially a loop program is a formalism to compute primitive recursive functions, that is interpreted on a register machine (Shepherdson & Sturgis machine). The instructions of a loop program are of the following type:

```
i)   x ← 0
ii)  x ← y
iii) x ← y + 1
iv)  LOOP  x
       π
     end
```

The instructions of type i), ii), iii) have a transparent meaning of assignment statements of the content of register X, respectively clear, load, add 1 and load. A LOOP-END pair is the function that gives the iterative capability to the language. It is sortof a DO statement where the sequence of instructions (program) denoted by π is executed as many times as the content of X (in case $(x) = 0$, no times at all). Content of X is assumed unchanged during execution of π.

Using loop programs D. Ritchie defined a hierarchy of classes of functions in the following way: the first level L_0 is made of all functions that have a program that does not require a LOOP instruction. Level L_n is made off all functions that have a program where a LOOP-END pair is nested n times.

Intuitively the notion of complexity that has been used by Axt and Ritchie is a notion of structural complexity, that is it has to do with the complexity of the description of the way we have to compute the function. Surprising enough are, hence, the basic results:

i) $L_2 = \mathcal{E}_3$

ii) $n \geqslant 3 \quad K_n = L_n = \mathcal{E}_{n+1}$

Inside the elementary functions the comparison between the hierarchies is rather cumbersome. D. Tzichritzis has tried to fill up the table of comparisons that is based on the conjecture $\lambda x \, [\sqrt{x}\,] \notin K_2$ while it is in L_2.

1.3 Classes based on predictability.

Among different approaches to the notion of complexity of primitive recursive functions, there has been a special interest for the notion of predictability because it has been showed that it can provide not only a new hierarchy of classes but alsoarefinement of an important class such as the class of elementary functions.

To be able of predicting the amount of resource that is needed to compute a function is the basic idea underlying Robert Ritchie's original paper (1963), and later work by Cleave and Herman.

Robert Ritchie's defined a hierarchy of recursive functions in the following way: let F_n be the (n+1) th level of the hierarchy: all functions f in F_{n+1} can be computed by Turing Machines which use in their computations an amount of tape a_f bounded by a function in F_n. Let F be the set of functions that can be computed by a Turing machine which behaves like a sequential machine, that is as soon as it scans the input it prints out the output. Since the tape needed to compute functions in F is exactly the tape needed to express (in binary) the imput and the output, the class F is the level zero of the hierarchy. Actually, the first interesting level of the hierarchy is F_1 that is the set of all functions whose computation requires an amount of tape that is bounded by a function in F. An example of function in F_1 is 2^x. In general, if $f_0(x) = x$ and $f_{i+1}(x) = 2^{f_i(x)}$ then $\forall_i \; f_i \in F_i$. Other results are:

i) $\forall_n \; F_n \subsetneqq F_{n+1}$

ii) $\mathcal{E}_2 = \{ f \mid a_g(x_1, \ldots, x_n) \leq K_g \cdot \ell(x_1, \ldots, x_n) \} \subsetneqq F_1$

iii) $\bigcup_{i=1}^{\infty} F_i = \mathcal{E}$

In the same year J. P. Cleave proposed an other hierarchy of functions based on the predictability of the number of jump instructions executed during the computation of a function on a register machine. Also in this case what we get is a refinement of the class of elementary functions but, due to the use of register machines, we can extend the hierarchy to transfinite ordinals in such a way that all Grzegorczyck levels above the elementary functions are refined through an infinite hierarchy of levels. Let Σ be a set of functions, $E(\Sigma)_{i+1}$ is the set of functions that can be computed on a jumplimited register machine allowing the following instructions:

$y \leftarrow f(x_1, \ldots, x_n) \qquad f \in \Sigma$

IF $X = 0$ THEN GOTO n ELSE GOTO m

and accessing the jump instruction a number of times bounded by some function in $E(\Sigma)_i$.

Let us take $\Sigma = \{+, \cdot, \lambda x \lambda y [\text{IF } x=y \text{ THEN } 1 \text{ ELSE } 0]\}$ and let $E(\Sigma)_{-1}$, be the set of constant functions. Then, let

$$
\begin{aligned}
E_0 &= E(\Sigma)_0 \\
E_{\omega r+s} &= E(E_{\omega r})_s \qquad r,s < \omega \\
E_{\omega r} &= \bigcup_{i<\omega r} E_i \\
E_{\omega^2} &= \bigcup_{i<\omega^2} E_i
\end{aligned}
$$

The basic result is that for $s \geqslant 1$ $E_{\omega s} = \mathcal{E}_{2+s}$

This means that if we are allowed to compute in just one step all functions of a Grzegorczyck level, then we can refine the next level with infinitely many classes based on predictability of jumps. Besides, according to Herman, the refinement obtained by Cleave is as coarse as the one given by Ritchie inside the elementary functions.

A bibliography of subrecursive classes.

R.M. Robinson, Primitive recursive functions, Bull. AMS, 53 (1947)

J. Robinson, General recursive functions, Proc.AMS, 1 (1950).

A. Grzegorczyk, Some classes of recursive functions, Rozprawy Matematyczne (1953)

R. Péter, Rekursive funktionen, Akadémiai Kiadò, Budapest (1957)

S.C. Kleene, Extension of an effectively generated class of functions by enumeration, Colloq.Math.,6 (1958)

P. Axt, On a subrecursive hierarchy and primitive recursive degrees, Trans.AMS, 92 (1959)

R.W. Ritchie, Classes of predictably computable functions, Trans.AMS 106 (1963)

J. Cleave, A hierarchy of primitive recursive functions, Zeitschr. f. math.Logik und Grundlagen d. Math. 9 (1963)

R.W. Ritchie, Classes of recursive functions based on Ackermann's function, Pacific Journal of Mathematics, 15, 3 (1965)

P. Axt, Enumeration and The Grzegorczyk hierarchy, Zeitchr. f. math. Logik und Grundlagen d. Math., (1965)

A. Cobham, The intrinsic computational difficulty of functions, Logic, Methodology and Phylosofhy of Science, Amsterdam (1965)

J.W. Robbin, Subrecursive hierarchies, Ph. D. Thesis, Princeton, (1965).

A.R. Meyer and D.M. Ritchie, "The complexity of Loop programs" Proc. 22 National ACM Conf.(1967)

R.L.Constable, Extending and refining hierarchies of computable functions, Comp. Sci.Tech.Rep.N.25, University of Wisconsin (June 1968)

D. Tsichritzis, A note on comparison of subrecursive hierarchies, Information Processing Letters 1 (1971)

J.C. Warkentin, Small classes of recursive functions and relations, Tech. Rep. CSRR 2052, U. of Waterloo, Canada (1971)

G.T. Herman, The equivalence of various hierarchies of elementary functions, Zeitsch. Logik und Grun dlagen d.Math., 17 (1971).

2. The machine dependent approach.

2.1 Time and tape required by Turing machines computations.

Among the different notions of complexity that we have bee considering until now, only the notion of predictability is related to the amount of resource that is needed to perform a given computation. In the case of Ritchie's classes, though, this is done with the purpose of stressing the constructability of certain classes of primitive recursive functions. Under a weaker point of view the notion of amount of time (tape) required by a Turing machine has served the twofold purpose of

- studying the complexity of specific problems like recognizing languages, computing functions belonging to certain classes, etc.
- introducing the notion of complexity classes, that is of classes of functions whose running time (tape) is bounded by a given function.

While the former type of results has been relevant for what we might call "low level complexity" (most of the problems are characterized by elementary functions), the latter is considered the point of departure of "high level complexity" (or abstract computational complexity) where the idea of complexity class with respect to a very general notion of resource, is developed.

Most of the work in formal languages theory, specially for what concerns the Chomsky hierarchy, has been done with the purpose of characterizing the automata that are needed to solve the recognition problem for a given class of languages.

A few results are summarized here: Let us use a Turing machine whith a read only input tape and a read / write work tape (we say that this machine operates off-line); let n be the lenght of a word:

i) regular languages are recognized with constant (zero!) work tape and linear (n) time;

ii) context-free languages are recognized with (log n) tape and n^3 time

Let the Turing machine be non-deterministic:

iii) context-sensitive languages are recognized with linear (a + b n) tape (it is still an open problem whether a deterministic TM can recognize CS languages with linear tape).

It is clear that this kind of problems are more on the side of analysis of algorithms than on the side of theoretical computational complexity. A more general kind of result is the following (Cobham):

iV) For any $n \geqslant 3$ $\lambda x[f(x)]$ is in $\mathcal{E}_n$ if and only if there is a TM that computes f using an amount of time and an amount of tape that are bounded by functions in $\mathcal{E}_n$.

This is certainly one of the most important results in the theory of subrecursive classes because it states that Grzegorczyck classes are meaningful not only from the point of view of the growth rate and of the structural complexity but also from the point of view of the computational complexity.

Unfortunately this happens only above the elementary functions; as we have already said, what happens below elementary functions is rather messy and is usually studied from the point of view of the analysis of algorithms. A few results are anyway available as far as $\mathcal{E}_2$ is concerned, for example:

v) $\lambda x[f(x)]$ is in $\mathcal{E}_2$ if and only if there is a TM which computes f using an amount of tape bounded by $a + b \log x$.

2.2 Resource bounded complexity classes.

An important question that comes out from problems in this area is the following: suppose we have a bound on the complexity of a certain class of problems; how much we have to increase the bound in order to be able of increasing the computational power? For example: we know that in constant tape a Turing machine can recognize all and only regular languages; it is known that there is a non regular context free language that can be recognized in tape log log n; which is the slowest growing function that allows a Turing machine to recognize a non regular context free language?

In general, both in case people is interested in practical computations and in case is interested in computations by abstract machines, the problem of defining complexity classes as classes of functions whose complexity is bounded by a certain total function, and, given a function, the problem of realizing whether it is contained in a given complexity class and in case it is not, the problem of finding the smaller complexity class that contains it, are central in the whole field of computational complexity.

From the point of view of machine dependent complexity the first results on complexity classes have been given by Hartmanis and Stearns in 1965. Actually they defined the notion of complexity classes for binary sequences and with respect to two particular complexity measures that are time and tape for Turing machines, but their results can be thought of as the start of all high level computational complexity studies.

A sequence α is t-computable if and only if there exists a multitape TM which prints the first n digits of the sequence α in no more than t(n) steps. Let C_t^s be the class of all t-computable sequences. The only hypotheses on t are that $t(n) \leq t(n+1)$ and $(\exists K)(\forall n)[t(n) \geq n/K]$

The result for time bounded computations is the following:

i) If $u(n)$ is real time countable, that is

- $u(n) \geqslant n$
- $u(n)$ monotone increasing
- there is a Turing machine whose output on the j-th step is 1 if $u(i)=j$ for some i, 0 otherwise;

and if $t(n)$ is a time function, then

$$\inf_{n \to \infty} \frac{t(n) \log t(n)}{u(n)} = 0$$

is a sufficient condition to say that: $C_u^s \supsetneq C_t^s$

Now let W to be a set of finite strings. We say that W is 1-tape recognizable if there is a TM that accepts each word in W and rejects any other word, and for any string of lenght n the machine head visits at most $l(n)$ squares, where l must be a monotonically increasing recursive function.

The class of all 1-tape recognizable sets is denoted by C_1^T

ii) If $l(n)$ is a constructable tape function, that is there exists a TM which stops for all input sequences and for each n takes at most $l(n)$ squares for any string of lenght n and takes exactly $l(n)$ squares for some string of lenght n , then

$$\inf_{n\to\infty} \frac{q(n)}{l(n)} = 0$$

is a sufficient condition to say that $C_1^T \supsetneqq C_q^T$

It can be noticed that the result for tape complexity classes is stronger than the result for time complexity classes, in fact the computational power of a class can be more easily expanded.

Similar results have been obtained for other measures and all of them give an interesting insight in the properties of the abstract device and of the resource that has been considered. For example if we use the number of head reversals of a TM, and if $\tau(n)$ is a constant function or a "slowly" growing function it is enough to encrease by 1 the number of reversals to encrease the computational power, while if τ is such that $\lim_{n\to\infty} \frac{n}{\tau(n)} = 0$ even by decreasing it by any constant factor we cannot reduce the computational power of the class C_τ^R.

The most recent results of this type have been proved by Hartmanis using Random Access Stored Program machines and number of instructions executed as a complexity measure ;

these results state that:

i) For any $\varepsilon > 0$ there exist arbitrarily complex functions $f_i(n)$ which can be computed in time $t_i(n)$ but cannot be computed in time $(1-\varepsilon)\, t_i(n)$ by any RASP program.

ii) There exist arbitrarily complex functions $f_i(n)$ with self modifying programs running in time $t_i(n)$, such that any fixed program for f_i of lenght ℓ cannot run faster than $(1 + 1/2\ell)\, t_i(n)$ for large n.

A bibliography of machine dependent theory of complexity.

H. YAMADA, Real time computation and recursive functions not real time computable, IRE Trans.Electronic Computers EC-11 (1962)

J. HARTMANIS, R.E. STEARNS, On the computational complexity of algorithms, Trans. Amer. Math.Soc. 117 (May 1965)

J. HARTMANIS, P.M. LEWIS II, R.E. STEARNS, Classifications of computation by time and memory requirements, IFIP Congress 1965. Vol.1 Spartan Books, Washington, D.C. (1965)

R.E. STEARNS, J. HARTMANIS, P.M. LEWIS II, Hierarchies of memory limited computations, 1965 IEEE Conference Record on Switching Circuit Theory and Logical Design, (October 1965)

P.M. LEWIS II, R.E. STEARNS, J. HARTMANIS, Memory bounds for recognition of context-free and context-sensitive languages, 1965 IEEE Conference Record on Switching Circuit Theory and Logical Design, (Oxtober 1965).

F.C. HENNIE, One tape off-line Turing machine computations, Information and Control, 8 (1966)

F.C. HENNIE, R.E. STEARNS, Two-tape simulation of multitape Turing Machines, Journal ACM, 13 (1966)

J. Hartmanis, Tape-reversal bounded Turing Machine computations, Journal of Computer and System Sciences, 2 (August 1968)

J.E. HOPCROF, J.D. ULLMAN, Formal languages and their relation to automata, Addison Wesley Publ. Co. (1969)

J. HARTMANIS, R.E. STEARNS, Automata-based computational complexity, Information Sciences, 1 (1969)

S.A. COOK,Variations on Pushdown Machines, Proceeding of the ACM, Symposium on Theory of Computing, Marina del Rey, Calif. (May 1969)

J. HARTMANIS, Computational complexity of Random Access Stored Program machines, J. of Mathematical System Science (1971)

3. The abstract approach.

3.1 Machine dependence of the notion of complexity.

It is intuitively clear that there is no such a thing as the complexity of a function unless we talk in terms of a specific way of computing that function; an attempt of studying the machine dependence of the notion of complexity has been done by Arbib and Blum in 1965 and will help us to make the statement more precise.

At this point we will introduce a few notions that are at the base of the abstract approach to computational complexity. We will hence refer to them throughout the rest of the chapter.

Let $\varphi = \{\varphi_i\}_{i=0}^{\infty}$ denote an acceptable Gödel numbering of all partial recursive functions of one variable(*) By definition the set $\{\varphi_i\}_{i=0}^{\infty}$ must satisfy the following properties:

i) $(\exists s \in \mathcal{R}_2)(\forall i,x,y)[\varphi_{s(i,x)}(y) = \varphi_i(<x,y>)]$ (s-m-n)

ii) $(\exists i)(\forall x,y)[\varphi_i(<x,y>) = \varphi_x(y)]$ (unversality)

where $\mathcal{R}_m$ is the set of all total recursive functions of m variables and $\lambda x\, \lambda y\, [<x,y>]$ is an effective bijection N×N → N

An infinite subset Φ of $\{\varphi_i\}_{i=0}^{\infty}$ will be chosen as complexity measure and denoted by $\{\Phi_i\}_{i=0}^{\infty}$ if it satisfies the following two axioms (Blum):

A1. $(\forall i,x)[\varphi_i(x) = \downarrow \text{ iff } \Phi_i(x) = \downarrow]$

(*) - For sake of simplicity all definitions will refer to functions of one variable but they can be immediately extended to more variables.

A2. the relation $\Phi_i(x) = n$ is total recursive in i, x, n;
where $\varphi_i(x) = \downarrow$ denotes $\varphi_i(x)$ defined.

Remark: by convention, $\Phi_i(x) \neq \downarrow$ denoted by $\Phi_i(x) = \uparrow$ means that for all n $\Phi_i(x) > n$.

The meaning of the axioms and their consequences are the subject of the rest of the chapter. Here we will just make use of the notation so that we can say φ_i is a program and $\Phi_i(x)$ is the number of steps that the program φ_i takes before halting (if ever) on input x . We will call the pair (φ, Φ) a class of machines.

Now let S be a moneid of total functions $\sigma : N \times N \to N$ increasing with respect to the second variable, with composition satisfying $\sigma \circ \sigma' = \lambda x \lambda y [\sigma(x, \sigma'(x,y))]$ and with identity $e = \lambda x \lambda y [y]$

Given a complexity measure Φ and a monoid S we can induce an ordering with respect to complexity.

i) f is no more difficult than g with respect to (Φ, S) that is $f \underset{\Phi,S}{\leq} g$ if domain $(g) \subseteq$ domain(f) and for every index i for g there is an index j for f and a $\sigma \in S$ such that $\sigma(x, \Phi_i(x)) \gtrsim \Phi_j(x)$ for almost all x in domain (g).

ii) two classes of machines (φ, Φ) and $(\hat{\varphi}, \hat{\Phi})$ are S-similar if both $(\varphi, \Phi) \leq_S (\hat{\varphi}, \hat{\Phi})$ and $(\hat{\varphi}, \hat{\Phi}) \leq_S (\varphi, \Phi)$ where we say that $(\varphi, \Phi) \leq_S (\hat{\varphi}, \hat{\Phi})$ iff for every j there exists an i and a $\sigma \in S$ such that $\varphi_i \approx \hat{\varphi}_j$ and $\sigma(x, \hat{\Phi}_j(x)) \geq \Phi_i(x)$. We denote S-similarity by $\approx_S$.

It immediately follows that if we take $S_0 = \{e\}$ then no two classes of machines are similar: in fact theories based on time and space (for TM, say) complexity measures make use of S_0 and, hence, have a very low degree of invariance. On the other side, if we take $S_\infty = \{\sigma \mid \sigma \in R_2$ and increasing in the second variable $\}$ practically all properties become invariant from one measure to another because, as we will see, all measures are recursively related.

The non trivial case studied by Arbib and Blum is the following: let $\tilde{S}$ be the set of all functions $\lambda x \lambda y [q(\log x, y)]$ where q is a polynomial monotonically increasing in both variables. Then all TM with base $k>1$ input and base $l>1$ output, finitely many tapes, finitely many heads per tape, are $\tilde{S}$ similar.

In this way it can easily be shown that, for example, $\lambda x [2^x]$ is $\tilde{S}$ -more difficult than $\lambda x [x]$ on any TM.

Other results on the so-called weak invariance of computational complexity have been given by Burkard and Kroon in 1971 but the basic result coming from machine dependent studies and from Arbib and Blum's work on invariance is that in a strong sense no notion of complexity can be completely machine independent without becoming trivial. As we will see later on, even in a specific measure no notion of complexity can be attached to all functions because there are functions that do not have a "best" program. So the only way of dealing with the problem in a machine independent way is to provide a theory whose theorems are proven in a machine independent way, so that even though the results are necessarily machine dependent, the methodology of producing them is machine independent.

3.2 Basic results in abstract computational complexity.

Blum's axioms for computational complexity have been introduced in the proceding chapter. We will now provide a motivation for them and state the first basic results.

According to the axioms we will consider as an acceptable measure of complexity of a recursive function another recursive function $\Phi_i = \varphi_{\sigma(i)}$ that is defined if and only if φ_i is defined and has the property that we are able to answer to the question, is the complexity of φ_i on input x equal to n ? That is we want the graph of Φ_i to be recursive. Besides, we want σ to be recursive that is we want to be able to go from functions to step-counting functions in a uniform and effective way. It is immediate to realize that very natural measures like time for Turing machines, number of instructions executed by a register machine, number of applications of rules of a Markov algorithm, number of squares scanned by a Turing machine (provided we let it be undefined if the machine does not halt), satisfy both axioms.

The first interesting consequence of this definition of complexity measure is that every two measures Φ, $\hat{\Phi}$ are recursively related, that is

i) $(\exists r \in \mathcal{R}_2)(\forall i)(\overset{\infty}{\forall} x)\,[\Phi_i(x) \leq r(x, \hat{\Phi}_i(x))$
$\qquad\qquad \& \; \hat{\Phi}_i(x) \leq r(x, \Phi_i(x))\,]$

where $\overset{\infty}{\forall}$ means almost every-where.

Together with Hartmanis work on complexity classes and Arbib and Blum's work on machine invariant properties, another result that is at the base of computational complexity theory is the work by Rabin (1960) that shows that there are arbitrarily complex characteristic functions. M.O. Rabin proved his result for a general computing device (Post algorithms) but it can be nicely expressed in abstract terms:

ii) $(\forall g \in \mathcal{R}_1)(\exists f : N \to \{0,1\})(\forall i)(\varphi_i = f)$

$$(\overset{\infty}{\forall} x)[\Phi_i(x) \geq g(x)] \quad .$$

Two remarks have to be made at this point. The first is that we need to prove the result for 0-1 valued recursive functions in order to show that the complexity of the function is not a consequence of an exceptionally large output, but is entirely involved by the computational process.

The second point is that we cannot prove a stronger result where "for all but a finite number of x" is replaced by "for all x". In fact whatever function we take it is always possible to embed a finite number of values of the function in the program so that we can compute them by table look-up with no cost. This intuitive point can be made more precise in the following way: since the lenght of the program φ_i can be assumed to be about log i and since a table can be at most as large as the program it self, it has been proved by Meyer and Mc Creight that the number of values of x that can always be "easily" computed, can be made as small as $\log_2(i) + \sigma(\log_2(i))$. The same authors have, besides, pointed out that if a 0-1 valued function has the property that the only way of computing it fast on finitely many points is to put those values in a table, it means that no subroutine can help us on infinitely many points, that is to say that no value of the function (beside the values that are in the table) can be "predicted" faster than it takes to actually compute it. In an intuitive sense, hence, those functions are "very hard" to compute; using tape for Turing machines as a complexity measure, Meyer and Mc Creight have finally proved that "very hard" functions exhibit the properties of pseudo-random functions, that is their average value is 1/2. This kind of results seem to provide an intersection between axiomatic computational complexity and Kolmogorov's approach (par.0, point D).

After having showed that, if we define $f \underset{c}{\leq} g$ iff a program for f runs faster than any program for g almost every where, $\underset{c}{\leq}$ has no maximal element, the last of the basic results we state in this paragraph, says that $\underset{c}{\leq}$ has incomparable element since there are 0-1 valued functions that do not have a best program (speed-up theorem).

iii) $(\forall r \in \mathcal{R}_2)(\exists f : N \to \{0,1\})(\forall i)(\varphi_i = f)(\exists j)(\varphi_j = f)$

$$(\overset{\infty}{\forall} x)\,[\Phi_i(x) \geq r(x, \Phi_j(x))]$$

In other words, given any function r, there are functions f such that any program for f can be speeded-up by an amount r almost every where. It is interesting to notice that f can be a very easy (in an intuitive sense) function. For example, Markov algorithms that compute the characteristic function of even lenght sequences of digits can be very easily speeded-up by an amount $\lambda x\,[2^x]$

The speed-up theorem has been proved in stronger versions where the composition $\tau \circ \Phi_i$ is substitued by iteration (Blum) and by a general recursive operator (Meyer, Fisher, Hartmanis).

3.3 Compexity classes.

Let Φ be any Blum's acceptable complexity measure; let t be any total recursive function: we are interested in characterizing the set of functions whose complexity is bounded by t almost every where

$$C_t^{\Phi} = \{ f \mid f \in \mathcal{R}_1 \text{ and } (\exists k)(\varphi_k = f)(\overset{\infty}{\forall} x)\left[\Phi_k(x) \leq t(x)\right] \}$$

As we have seen in the chapter on machine dependent complexity theory, the first property of complexity classes that people is interested in considering is how to expand them in a uniform way. The most important results in this direction are the following:

i) in any measure there is a recursive function τ such that given any total recursive function φ_i the class bounded by φ_i is strictly contained in the class bounded by $\lambda x[\tau(x, \Phi_i(x))]$. In particular if we consider sufficiently complex φ_i , such that $\Phi_i(x) \geqslant x$, wr can obtain

$$C_{\varphi_i} \subsetneqq C_{\tau \circ \Phi_i}$$

It would be interesting to find a way of increasing the computational power of the class, uniformly, in terms of the bound it-self φ_i instead of its complexity Φ_i . Unfortunately the gap theorem (Borodin) proves that this is not possible.

ii) Given any recursive function τ there are arbitrarily large functions t such that the class bounded by $\tau \circ t$ is exactly as large as the class bounded by t.

If we want to get rid of gaps we have to accept only "good" functions as bounds for a complexity class. In fact if we consider classes bounded by running times or (more weakly) by functions belonging to a "measured set" (an infinite set of different functions with recursive graph) we can prove the following compression theorem (Blum).

iii) Let $\{\gamma_i\}$ be a measured set of functions. Then in any complexity measure Φ there exists a recursive function τ such that the class bounded by γ_i is strictly contained in the class bounded by $\lambda x\,[\tau(x, \gamma_i(x))]$

The compression theorem actually shows that the gap theorem is a consequence of a wrong way of setting bounds to complexity classes. Let us call the bound functions "names" of the complexity classes. The gap theorem shows that there are different names for the same classes that are as far apart as we want while the compression theorem shows that if we pick names out of a measured set the gap property disappears. The whole problem is hence reduced to choosing "good" names for complexity classes. Can we do this for all complexity classes? The answer is yes and is given by the naming theorem (Meyer, Mc Creight).

iv) For each measure Φ there exists a measured set naming all complexity classes.

At this point we can also give a meaning to the expression "good" name. We say a function f is g-honest if there is a program φ_i for f such that the complexity of φ_i does not exceeds more than an amount g the value of the function. That is

$$(\overset{\infty}{\forall} x \in domain(f))\,[\,\Phi_i(x) \leq g\,(\max\{x, f(x)\})\,]$$

It turns out that the notion of honest set and the notion of measured set are equivalent. Hence the naming theorem can be also stated in the following way: it is always possible to give honest names to complexity classes in a uniform and effective way.

3.4 Primitive recursive functions and abstract complexity.

The notion of complexity class is an intuitively appealing notion but are we sure that when we refer to a specific resource the set of functions computable within a given bound on that resource have any interesting characterization ? We can give examples where the answer is yes and that are enough to justify research efforts in the abstract approach.

First we need to state a closure property of complexity classes: it is the <u>union theorem</u> proved by Meyer, Mc. Creight.

i) Let $\{g_i \mid i=1,2,\dots\}$ be an r.e. set of recursive functions such that for each i and x $g_i(x) < g_{i+1}(x)$ there exists a recursive function $t(x)$ such that

$$C_t = \bigcup_{i=1}^{\infty} C_{g_i}$$

In other words an r.e. hierarchy of complexity classes is it-self a complexity class.

We already know that for all subrecursive classes $\mathcal{E}_n$ above (and included) the elementary functions, if a function f is computable by a TM within time and tape bounded by a function in $\mathcal{E}_n$, f it-self is in $\mathcal{E}_n$. Now let $\{\overline{g}_i \mid i=1,2,\dots\}$ be an infinite r.e. sequence of functions in $\mathcal{E}_n$, eventually majorizing any function of $\mathcal{E}_n$(e.g. $\overline{g}_i = \lambda x\,[f_n^{(i)}(x,x)]$ where f_n is as defined in paragraph 1.1)

Now let us define

$$g_i = \max\{\bar{g}_1(x), \bar{g}_2(x), \dots, \bar{g}_i(x)\} + i$$

the sequence $\{g_i\}$ satisfies the hypotheses of the union theorem and so there must be a function t that bounds exactly the class $\mathcal{E}_n$ in terms of time or tape for TM. This is a rather interesting result that gives yet another argument in favor of the fact that Grzegorczyck classes have a very deep meaning in terms of complexity.

Since all Grzegorczyck classes have now been proved to be complexity classes with respect to some recursive bound, it is easy to draw the consequence that also primitive recursive functions form a complexity class. One point that still has to be investigated is whether we can characterize the "names" of Grzegorczyck classes and of primitive recursive functions without having to compute them using the techniques of the union theorem.

While the union theorem has played in favor of subrecursive classes another result of Blum says that if we use only a subrecursive programming language we occasionally may have to write an enormously large program for a function that have a very short program if we use a general recursive language. In order to define these concepts we first need to introduce the axiomatic notion of size of a program.

A recursive function $|| : N \to N$ is called a size function and we say $|i|$ is the size of the program φ_i if there is an effective way of listing, given n the entere finite set of programs of size n and of knowing when he listing is complete.

An acceptable size measure is, for example, given by the number of characters of a FORTRAN program or by the state-symbol product for a TM.

Now we can formulate the following result:
ii) Let g be a recursive function with infinite range: (g enumerates indices of an infinite sequence of algorithms, in particular enumerates all smallest primitive recursive schemas); let f be a recursive function; there exist integers i and j such that $\varphi_i = \varphi_{g(j)}$ and $f(|i|) < |g(j)|$. Besides there exist a recursive function h, not dependent on g and f, such that $\Phi_i(x) \leq h(x, \Phi_{g(j)}(x))$ almost every where φ_i is defined.

In other words, whatever function is f, there is a function whose smallest primitive recursive derivation is larger than a general recursive program for the same function of an amount f and besides the complexity of this program is recursively related to the complexity of the subrecursive program.

More work in the direction of evaluating advantages and disadvantages of subrecursive languages has been done by Constable and Borodin by using specific subrecursive languages (such as LOOP programs).

Another interesting result on size of programs is the one by Young and Helm showing that there are speedable characteristic functions for which not only is impossible to find the faster programs in an effective way but even their sizes cannot be recursively bounded because they grow faster than any recursive function.

A bibliography of abstract computational complexity

M.O. Rabin, Degree of difficulty of computing a function and a partial ordering of recursive sets, Tech.Rep. No.2, Hebrew U., Jerusalem, Israel, (April 1960).

M. Arbib, M. Blum, Machine dependence of degrees of difficulty, Proc. AMS, 16 (1965).

M. Blum, A machine independent theory of the complexity of recursive functions, Journal of ACM, 14 (April 1967).

M. Blum, On the size of machines, Information and Control II (1967).

A.R. Meyer, P.C. Fisher, On computational speed-up, IEEE Conf. Rec. 9th Ann.Symp. on Switching and Automata theory, (October 1968).

A.R. Meyer, D.M. Ritchie, A classification of functions by computational complexity, Proc. Hawaii International Conference on System Sciences, U. of Hawaii(1968).

R.L. Constable, Upward and downward diagonalization over axiomatic complexity classes, Tech. Rep. N.69.72. Dep. of Computer Science, Cornell Univ.,Ithaca,N.Y. (March 1969).

E.M. Mc Creight, A.R. Meyer, Classes of Computable functions defined by bounds on computation, ACM Symposium on Theory of Computing, Marina del Rey, Calif. (May 1969)

E.M. Mc Creight, A note on complex recursive characteristic functions, Dep. of Computer Science, Carnegie-Mellon U., Pittsburgh, Pa. (May 1969).

P.R. Young, A note on the union theorem and gaps in complexity classes, CSD TR, Purdue U., Lafayette, Indiana (July 1969).

J. Helm, P. Young, On size vs; efficiency for programs admitting speed-ups, CSD TR 43, Purdue U., Lafayette, Indiana (Sept. 1969).

F.D. Lewis, Unsolvability considerations in computational complexity, 2nd Ann. ACM Symposium on Theory of Computing, Northampton, Mass. (May 1970).

R.L. Constable, On the size of programs in subrecursive formalisms, ibid.

R.L. Constable, A.B. Borodin, On the efficiency of programs in subrecursive formalisms, Comp. Sci. Tecn. Repart 70-53, Cornell Univ., Ithaca.N.Y. (1970).

W.A. Burkhard, F.W. Kroon, Toward a weakly invariant complexity theory, twelfth Annual Symposium on Switching and Automata theory, East Lansing, Michigan (1971).

M. Blum, On effective procedures for speeding up algorithms, Journal of ACM, 18 (1971).

J. Hartmanis, J.E. Hopcroft, An overview of the theory of computational complexity, Journal of ACM, 18 (1971).

A.R. Meyer, E.M. Mc Creight, Computationally complex and pseudo-random zero-one valued functions, An Int. Symp. on the theory of Machines and Computations, Haifa, Israel (1971).

P. Van Emde Boas, A note on the Mc Creight-Meyer naming theorem in the theory of computational complexity, Mathematical Centrum, Amsterdam, Holland (1971).

M. Blum, Classifying complexity properties of partial recursive functions, Symposium on Computational Compexity, NYU (1971)

A. Borodin, Computational complexity and the existence of complexity gaps, Journal of ACM, 19 (1972)

R.L. Constable, The operator gap, Journal of ACM, 19 (1972)

4. Structural abstract computational complexity.

In this paragraph we will examine some of the developments of the axiomatic complexity theory with particular attention to the problem of introducing the notion of program structure. In some sense the classical Blum's work on computational complexity and most of the later papers in the fieel make use of global notions about a computational process: amount of resource needed by the computation, size of the program. As we have seen we can obtain interesting results along this way but we feel that we are not able of describing what really happens during the computation and how two programs for the same function behave in a different way according to their different structures. Also, we feel that if we were able to handle those problems, we could also get rid of many pathologies of the theory.

4.1 Stronger axioms.

In order to reduce the number of acceptable measures a few different kinds of auxiliary axioms have been proposed. Among them (Meyer, Mc Creight and Borodin):

i) properness: we want to be able of computing the running time of a function just by running the function it-self, hence a running time must be a strongly honest function, if it is large enough.

To be more precise:

$$(\exists j)(\forall e)(\Phi_e \geqslant \varphi_j)(\exists i)(\varphi_i = \Phi_e)[\Phi_i \leqslant \Phi_e]$$

ii) parallel computation: we require that, in order to be acceptable, a class of machines is able of carrying on computations where two programs are essentially run in parallel, that is, there must be a recursive function σ such that given any two programs φ_i and φ_j

$$\varphi_{\sigma(i,j)}(x) = \begin{cases} \varphi_i(x) & \text{if } \Phi_i(x) \le \Phi_j(x) \\ \varphi_j(x) & \text{(otherwise)} \end{cases}$$

and $\Phi_{\sigma(i,j)}(x) \le \min\{\Phi_i(x), \Phi_j(x)\}$

Those axioms are satisfied, for example, by the tape measure for Turing machines. They have been introduced mainly to be able of reducing the number of pathological measures but they are not enough to insure all desired properties. For example Landweber and Robertson have showed that there are proper measures with the parallel computation property for which not all complexity classes are r.e. On the other side, if we want all complexity classes to be r.e. we need the following axiom (Borodin):

iii) finite invariance: all total functions that are different only on a finite set are in the same complexity class, that is

$$(\forall f \in C_t^{\Phi})(\forall f')[(\overset{\infty}{\forall} x)[f'(x) = f(x)] \to (f' \in C_t^{\Phi})]$$

Another axiom has been proposed by Paul Young with purpose of guaranteeing that we are able, given a finite function, to find the most efficient algorithm for computing it.

iv) Principle R: let (φ, Φ) be an acceptable class of machines and $||$ and acceptable measure of size: there is a total function $c \in \mathcal{R}_3$ such that if φ_i is defined on the finite domain D_y and if there is an index j such that $x \in D_y$ implies $\varphi_i(x) = \varphi_j(x)$, with $\sum_{x \in D_y} \Phi_i(x) < \sum_{x \in D_y} \Phi_j(x)$ then $|j| \le c(\sum_{x \in D_y} \Phi_i(x), y, i)$

In other words if we have a bound on the amount of resource that is needed to run the program on a finite set of inputs, then we also have a bound on the size of the programs which need be considered in looking for programs which require a smaller amount of resource.

4.2 Weak computational complexity.

As we have pointed out before, abstract computational complexity allows us to prove things about halting computations and their properties. Besides if we want to consider the tape measure for Turing machines as an acceptable Blum measure we have to slightly change its definition:

$$\Phi_i(x) = \begin{cases} \text{number of squares scanned by the i-th TM if it halts} \\ \uparrow \text{ if the i-th TM does not halt} \end{cases}$$

This is unrealistic because for many classes of machines the resource can be used in a finite amount also when the computation diverges. Tipically a TM can cycle on a finite amount of squares and, besides, when the number of squares scanned is finite we are able of solving the halting problem. This fact is at the base of a weaker formulation of Blum's axioms for complexity (Ausiello):

A1a. if $\varphi_i(x) = \downarrow$ then $\Phi_i(x) = \downarrow$

A1b. there exists a partial recursive function N such that

$$\text{if } \Phi_i(x) = \downarrow \text{ then } \quad N(i,x) = \begin{cases} 1 & \text{if } \varphi_i(x) = \downarrow \\ 0 & \text{otherwise} \end{cases}$$

A2. the relation $\Phi_i(x) = n$ is recursive in i,x,n.

Besides proving all the results of the classical abstract theory that mainly deal with halting computations, we are now able of proving things about computations that cycle on a finite amount of resource.

Let $K=\{x \mid \varphi_x(x)=\downarrow)$ and $S=\{x \mid \Phi_x(x)=\downarrow\}$

The set S is the set of programs which use a finite amount of resource. We can prove the following results:

i) For no measure, S can be recursive.

ii) S is creative and, hence, it is recursively isomorphic to K

iii) there are measures such that S-K is recursively isomorphic to K and besides S-K and K are effectively inseparable.

The last result seems to be important in the characterization of interesting measures (in fact, the TM tape measure exhibits this property). For this reason Ivan Havel introduced additional axioms that make possible to implement loops into programs and that are enough to achieve essentially the same result.

A3a. there exists a total recursive function q such that $\varphi_{q(i,j)}(x)=\varphi_j(\varphi_i(x))$

$\varphi_{q(i,j)}(x)$ cycles on the resource Φ if for some y

$\varphi_i(x)=y$ and $\varphi_j(y)$ cycles on the resource Φ

A3b. there exists a total recursive function τ such that

$$\varphi_{\tau(i,j)}(x) = \begin{cases} \varphi_i(x) & \text{if } x > 0 \\ \varphi_j(x) & \text{if } x = 0 \end{cases}$$

$\varphi_{\tau(i,j)}(x)$ cycles on the resource Φ if either

$x > 0$ and $\varphi_i(x)$ cycles on the resource

$x = 0$ and $\varphi_j(x)$ " "

Those axioms express the possibility of serial composition and branching of programs, both preserving loops so that we are also allowed to use the expression "$\varphi_i(x)$ cycles" whenever in the proof of a theorem we are allowed to say $\varphi_i(x)$ gives output y.

As we have seen the weaker axioms are a tool to prove properties of cycling programs and, in this sense even if they weaken Blum's theory, they really extend our ability in studying how a program makes use of a certain resource.

Along the same line of development we might be interested in having a complete description of how a resource is consumed during the computation and, in this way being able to prove things about strongly divergent computations, that are those computations which neither halt nor cycle on the resource.

For this purpose Giuseppe Longo has introduced the notion of computational resource as an infinite r.e. set of recursive functions $\{\lambda x \lambda t [f_i(x,t)]\}_{i=0}^{\infty}$

each one associated to a partial recursive function $\lambda x\,[\varphi_i(x)]$, where the dependence on t is needed to describe the time progression of the consumption of the resource. Three axioms are needed to characterize a computational resource: the first one states that the amount of resource goes to zero if and only if the program halts; the second one concerns the weakness of the complexity measure and formally describes the requirement that, if no more resource is consumed, we can effectively decide whether the computation is going to converge or not; the third axiom is a request on time dependence of consumption of resource, weaker than monotony, to answer recursively to the question whether a certain amount of it is eventually being consumed or not.

4.3 Computational complexity of structured programs.

As we have seen a major concern for people working in abstract computational complexity, has been to characterize acceptable classes of machines and acceptable resources in terms of properties and behaviour of specific programs: programs to compute running times, programs that carry on parallel computations, programs that cycle or diverge.

One way of dealing with all those problems is by introducing the notion of structured programs and to study the complexity of programs through the complexity of their parts.

A few very weak results have been obtained since the early days of abstract computational complexity. Ausiello gave them the following general form: let f be a partial recursive function defined in terms of other functions $g_1, \ldots, g_n$ such that f converges if and only if some well defined assertions about the convergency of $g_1, \ldots, g_n$ hold: then we can find a total function (depending on the number of steps required to compute $g_1, \ldots, g_n$) that bounds the step counting function of f almost every-where f is defined. For example, if $\varphi_{\sigma(i,j)}(x) = \varphi_i(x) + \varphi_j(x)$ we can use the theorem to say that there exists a total recursive function τ such that

$$\varphi_i(x) = \downarrow \text{ and } \varphi_j(x) = \downarrow \text{ implies}$$
$$\Phi_{\sigma(i,j)}(x) \leq \tau(x, \max\{\Phi_i(x), \Phi_j(x)\})$$

It is clear that the statement is ewtremely weak since τ can be rather large even if we can show that to compute the bound τ to the step counting function is not much harder than computing the step counting function it-self. For this reason we need a much more detailed study on how the complexity of a program depends on the complexity of its parts.

Work in this direction has been recently developed (Meyer, Linch, Symes) by first introducing the notion of oracle and then by interpreting horacles as subroutines. Symes defined a subroutine operator as a special type of recursive operator in which oracle function calls may not be performed in parallel; the consequence of introducing the facility of oracle calls is that although not all partial recursive functions become easy to compute, they all become honest. On the other side the whole complexity theory can be transposed onto subroutine operators; for example honest subroutine operators can be defined.

In the same work he makes an attempt to use subroutine operators as components of the program structure: for example elementary register operations can be considered as 0-ary subroutine operators (i.e. independent on any oracle call) while concatenation is a 2-ary s.o. and a LOOP-END pair is a 1-ary s.o. Subroutine operators are, then, an abstraction of specific structures for programming languages and they allow us to introduce auxiliary axioms for size and cost such as: suppose we are given a suitable defined enumeration of s.o. $\{s_i\}$ there exists a recursive function τ such that if a program S is a basic s.o. S_i the size of S is given by $|S| = \tau(i,0)$, if S is a composition $S_i(S_1,\ldots,S_n)$ then $|S| = \tau(i,<|S_1|,\ldots,|S_n|)$ if we replace one subroutine with a larger one then we increase the size of the whole program. As far as the cost, similar properties have been introduced and though the research has not been carried on to find relevant consequences of the said approach we think that the basic idea of studying the computational complexity of structured programs is certainly one of the main directior in which the research in computational complexity has to be developed.

A bibliography of structural abstract computational complexity.

G. Ausiello, Weaker axioms for abstract computational complexity, Fourth Princeton Conference on Information Systems and Science,Princeton, N.J.(1970).

G. Ausiello, Parallel universal computation, ibid.

G. Ausiello, On bounds on the number of steps to compute functions, Second Annual ACM Symposium on Theory of Computing, Northampton, Mass.(1970).

L.H. Landweber, E.L. Robertson, Recursive properties of abstract complexity classes, ibid.

I. Havel, Weak complexity measures, SIGACT News (1971).

G. Ausiello, Abstract computational complexity and cycling computations, Journal of Computers and System Sciences, 5 (1971).

P.C. Fisher, Trends in computational complexity theory, Symposium on Computational Complexity, NYU (1971).

N. Lynch, A universally helped recursive set, ibid.

D.M. Symes, The extension of machine independent computational complexity theory to oracle machine computation and to the computation of finite functions, CSRR 2057, University of Waterloo, Waterloo, Ontario (Oct.1971).

P.D. Young, A note on axioms for computational complexity and computation of finite functions, Information and Control, 19 (1971)

G. Longo, Towards an abstract analysis of time progression of consumption of resources during computation, Internat. Computing Symposium, Venice, Italy (Apr. 1972).

A Geometrical Model for a Stochastic Automaton

by

F. Bancilhon

1. SUMMARY

We here introduce a new model of stochastic automaton to express the geometrical nature of the problem of equivalence between stochastic sequential machines. Then, using the concept of pseudo automata we establich relationships between the algebraic and geometrical form. These results are used to solve an open problem of minimization by covering.

2. INTRODUCTION

Among all the problems which were discussed about stochastic sequential machines, the main one seems to be the equivalence between two SSM's:

The distinction has been made between non-minimal and minimal automata. Procedures to obtain a minimal automaton are well known now. Another notion, namely the rank of an automaton appeared to be distinct from the concept of non-minimality. The non-uniqueness of some minimal automata, the existence of full rank equivalent pseudo-automata, combined with the feeling of the geometric nature of the problem (Cones and Modules) seems to indicate the solution would be in a geometrical model.

We recall the following lemma from "Necessary conditions for automata interchangeability".

Lemma: Let (Z, m) be a c-automaton of rank r, then there exists B^L, B such that

i) B $(c \times r)$ and B^L $(r \times c)$ are pseudo-stochastic matrices of rank r such that

$$B^L B = I_r$$

ii) $m' = B^L \, m \, B$ is a monoid morphism

iii) $\forall\ \pi$, $(1\times c)$ such that $<\pi, e_c> = 1$

$\exists\ \pi'$, $(1\times r)$ such that $<\pi', e_r> = 1$

and $\pi\ m(Z^*)e = \pi'\ m'(Z^*)e$ and conversally

iv) The cone generated by the rows of B is closed under the action of $m'(Z^*)$

v) $v\ m'(Z^*)e = 0 \Rightarrow v = 0$

From this lemma we intuitively define a geometrical stochastic automaton (G.S.A)

Definition A GSA is a set

$$\Sigma_G = (\mathcal{E}\ ,\ (r_i)_{i=1,\dots,c},\ Z, \mathcal{M}\ ,\ f)$$

where $\mathcal{E}$ is a vector space of finite dimension r.

$Z = X\times Y$ is as previously defined.

$\mathcal{M} : Z^* \rightarrow \mathcal{L}(\mathcal{E}, \mathcal{E})$

$f\ :\ \mathcal{E} \rightarrow \mathbb{R}$

$r_i \in \mathcal{E}\quad \forall i = 1,\dots,c$

such that:

i) $\mathcal{M}$ is a morphism monoid

i.e. $\mathcal{M}_{(z\ \ z')} = \mathcal{M}_{(z)}\mathcal{M}_{(z')}$

ii) The set $P = \{\ \pi \in \mathcal{E}\ :\ f(\pi) = 1\}$ is closed under the action of

$$\sum_{y\in Y} \mathcal{M}(y|x)\ ,\ \forall \varkappa \in X.$$

iii) $r_i \in P \qquad \forall\ i = 1,\dots,c$

$r_i = r_j \Rightarrow i = j$

$(r_i)_{i=1,\dots,c}$ is of rank r.

iv) $f\ (v\mathcal{M}(Z^*)) = 0 \Rightarrow v = 0$

v) The polyhedral cone $\mathbb{C}$ generated by $(r_i)_{i=1,\dots,c}$ is closed under the action of $\mathcal{M}(Z)$.

We define the state space as $S(\Sigma_G) = P \cap \mathbb{C}$. The behaviour function of π is the map:

$$\mu^{\Sigma_G}_{\pi} : Z^* \to \mathbb{R}$$
$$z \to f(\pi \quad (z))$$

3. RELATIONSHIP BETWEEN GSA's.

Definition: 1) Two GSA, Σ_G and Σ'_G, are equivalent iff :

$\forall \pi \in S(\Sigma_G)$, $\exists\, \pi' \in S(\Sigma'_G)$ such that $\mu^{\Sigma_G}_{\pi} = \mu^{\Sigma'_G}_{\pi'}$

and conversally.

2) Two GSA are state equivalent iff

$\forall\, r_i$, $\exists\, r'_j$ such that $\mu^{\Sigma_G}_{r_i} = \mu^{\Sigma'_G}_{r'_j}$ and conversally.

Theorem The two GSA

$$\Sigma_G = (\mathcal{E}, (r_i)_{i=1\ldots c}, Z, \mathfrak{M}, f)$$
$$\Sigma'_G = (\mathcal{E}', (r'_i)_{i=1\ldots c'}, Z, \mathfrak{M}', f')$$

are equivalent iff there exists an isomorphism $g : \mathcal{E} \to \mathcal{E}'$ such that

i) $\mathcal{E}' = g(\mathbb{C})$

ii) $\mathfrak{M}' = g^{-1} o\ \mathfrak{M} o\ g$

iii) $f' = g^{-1} o\ f$

Theorem The two GSA Σ_G and Σ'_G are state eqivalent i), ii), iii) and there exists a permutation σ on $(r'_i)_{i=1\ldots c'=c}$ such that

$$\sigma(r'_i) = g(r_i)$$

4. RELATIONSHIP BETWEEN GSA and SSM

Theorem Let Σ be a c-automaton of rank r then there exists a unique (up to an isomorphism) geometric model state equivalent with it.

Theorem Let Σ be a c-automaton then there exists a unique (up to an isomorphism) minimal GSA equivalent with it.

Theorem Let Σ_G be a GSA then there exists an SSM state equivalent with it.

5. THE DOMINANCE PROBLEM

Definition Let Σ_G be a geometric stochastic automaton
$\Sigma_G = (\mathcal{E} , (r_i), Z, \mathcal{M}, f)$.
$\Sigma'_G = (\mathcal{E}', (r'_i), Z, \mathcal{M}', f')$ is a sub-automaton iff

i) $\mathbb{C}' \subset \mathbb{C}$

ii) $\mathbb{C}$ is closed under

iii) $\mathcal{E}'$ is spanned by (r'_i)

iv) $\mathcal{M}'$ is the restriction of $\mathcal{M}$ to $\mathcal{E}'$

v) f' is the restriction of f to $\mathcal{E}'$

Theorem $\Sigma' \geq \Sigma \Leftrightarrow G(\Sigma)$ is a sub-automaton of $G(\Sigma')$ up to a permutation.

Theorem Let $\Sigma = (Z, \quad)$ be a c-automaton of rank r. If there exists Σ' such that

$$\Sigma' \geq \Sigma \ , \ c' < c, \ r' > r$$

then there exists Σ'' such that

$$\Sigma'' \geq \Sigma \ , \ c'' \leqslant c' \ , \ r'' = r$$

Necessary conditions for automata interchangeability

by

F. Bancilhon

I SUMMARY

This paper is concerned with the relationship between internal and external description of stochastic automata. The concept of pseudo-automaton is defined, together with its relations with automata. A systematic investigation of the typeof behavioural equivalence between (pseudo-) automata is used to solve two problems:

1) Given an automaton, which transformations on its internal structure lead to a partially interchangeable, dominant or totally interchangeable automaton?

2) Conversally, given two (pseudo-) automata having some behavioural equivalence, which relationship is there between their internal descriptions?

II INTRODUCTION

Definition 1. A transition output automaton (Carlyle) is

X	input alphabet	$\|X\| = a$
Y	output alphabet	$\|Y\| = b$
R	state set	$\|R\| = c$

and $p(y, r_j \mid r_i, \alpha)$

Definition 2. The algebraic representation is $\Sigma = (Z, m)$ where :

$Z = X \times Y$

$m : Z^* \rightarrow \mathbb{R}^{c^2}$ is such that

i) m is a monoid morphism

ii) $m_{ij}(\vec{z}) \in [o, 1] \quad \forall i, j, \overline{z}$

iii) $\sum_{y \in Y} m(y|\varkappa)\, e = e$

Definition 3. The state space of Σ is

$$S(\Sigma) = \left\{ \pi \, (1 \times c) \; : \; \pi_i \in [o, 1], < \pi, e> = 1 \right\}$$

Definition 4. The behaviour function induced by the automaton $\Sigma = (Z, m)$ from the initial state π is the map

$$\begin{aligned} \mu_n^{\Sigma} \; : \; & Z^* \to [o, 1] \\ & (\bar{y} \,|\, \bar{x}) \to pr \, (\bar{y} \,|\, \bar{x}, \pi) \end{aligned}$$

A behaviour function has the following properties:

i) $\mu_\pi^{\Sigma} \, (\bar{z}) \in [o, 1]$

ii) $\sum\limits_{y \in Y} \mu_\pi \, (\bar{y} y \,|\, \bar{x} x) = \mu_\pi \, (\bar{y} \,|\, \bar{x})$

iii) $\mu_\pi \, (\bar{z}) = < \pi, \; m(\bar{z} \,|\, e > = < \pi \, m(\bar{y}), \; e >$

Definition 5. $\Sigma = (Z, m)$ is a prepresentation of μ iff there exists $\pi \in S(\Sigma)$ such that

$$\mu_\pi^{\Sigma} = \mu$$

Definition 6. A pseudo-representation of an external description is a couple $\Sigma = (Z, m)$ such that

i) $\sum\limits_{y \in Y} m \, (y \,|\, x) \; e = e$

ii) $m \, (\, z \, z' \;) = m(\, z \,) \, m(z')$

iii) $\exists \; \pi \quad 1 \times c$ such that $<\pi, \; e> = 1$ and $\pi \, m \, (\, \bar{z} \,) \; e = \mu(\bar{z} \,)$

The state space associated to this pseudo-representation is

$$PS(\Sigma) = \left\{ \pi \; : \; <\pi, \; e > = 1 \right\}$$

From now we use : pseudo-automaton instead of pseudo-representation and automaton instead of algebraic representation of an automaton. Notice that every automaton Σ can be considered as a pseudo automaton, we denote it by $\tilde{\Sigma}$.

Definition 7.

1) Two (pseudo-) automata $\Sigma_1 = (Z, m_1)$ and $\Sigma_2 = (Z, m_2)$ are partially interchangeable ($\sim$) iff $\exists\, \pi_1 \in (P)\, S(\Sigma_1)$, $\pi_2 \in (P)\, S(\Sigma_2)$ such that $\mu_{\pi_1}^{\Sigma_1} = \mu_{\pi_2}^{\Sigma_2}$

2) The (pseudo-) automaton $\Sigma_1 = (Z, m_1)$ dominates ($\geq$) the(pseudo-) automaton $\Sigma_2 = (Z, m_2)$ iff
$\forall\, \pi_2 \in (P)\, S(\Sigma_2)$, $\exists\, \pi_1 \in (P)\, S(\Sigma_1)$ such that $\mu_{\pi_1}^{\Sigma_1} = \mu_{\pi_2}^{\Sigma_2}$

3) Two (pseudo-) automata $\Sigma_1 = (Z, m_1)$ and $\Sigma_2 = (Z, m_2)$ are totally interchangeable (or equivalent $\equiv$) iff $\Sigma_1 \geq \Sigma_2$ and $\Sigma_2 \geq \Sigma_1$.

III TRANSFORMATION ON THE INTERNAL STRUCTURE PRESERVING SOME BEHAVIOURAL EQUIVALENCE

* Let $L(\Sigma, e)$ be the subspace of $\mathbb{R}^c$ spanned by the infinite family $m(\bar{z})\, e : \bar{z} \in Z^*$. The rank of Σ is the dimension of $L(\Sigma, e)$.

Lemma 1 Let $\Sigma = (Z, m)$ be a c-automaton of rank r, then there exists a subspace V such that

i) $\dim(V) = c-r$
ii) $V \subset \mathrm{Ker}\,(e)$
iii) V is closed under $m(Z)$

Theorem 1 Let $\Sigma = (Z, m)$ be a c-automaton of rank r, then there exists two pseudo-stochastic matrices B ($c \times r$) and B^L ($r \times c$) of rank r such that $B^L B = I_r$ and $\Sigma' = (Z, B^L m B)$ is a full rank pseudo-automaton totally interchangeable with $\tilde{\Sigma}$.

* Let $\Sigma(Z, m)$ be a (pseudo) automaton. Let $\pi \in (P)\, S(\Sigma)$. We denote by $K(\pi, \Sigma)$ the subspace of $\mathbb{R}^c$ spanned by the infinite family $\{\pi\, m(\bar{z}) : \bar{z} \in Z^*\}$.

Theorem 2 Let Σ be a c-automaton, let $\pi \in S(\Sigma)$ be a state such that $\dim (K(\pi, \Sigma)) = r$, then there exists two pseudo-stochastic matrices B^L $(r \times c)$ and B $(c \times r)$ of rank r such that $B^L B = I_r$ and that $\Sigma = (Z, B^L m B)$ is a pseudo-automaton dominated by $\tilde{\Sigma}$.

Theorem 3 Let $\Sigma = (Z, m)$ be a c-automaton. Let π be a state such that rank $(\mu_\pi) = r$. Then there exists an r-pseudo automaton Σ' which is partially interchangeable with Σ for π.

IV CONDITIONS OF INTERCHANGEABILITY

Lemma 2 Let $\Sigma_1 = (Z, m_1)$ be a c_1-automaton, $\Sigma_2 = (Z, m_2)$ be a c_2-automaton. Let Σ_1 and Σ_2 be partially interchangeable for $\pi_1 \in P(\Sigma_1)$ and $\pi_2 \in P(\Sigma_1)$ such that

$$\text{rank } K(\pi_1, \Sigma_1) = \text{rank}(\Sigma_1) = c_1$$

$$\text{rank } K(\pi_2, \Sigma_2) = \text{rank}(\Sigma_2) = c_2$$

then i) $c_1 = c_2 = c$

ii) $\exists$ P $c \times c$ regular pseudo-stochastic such that

$$m_1 = P^{-1} m_2 P$$

iii) $\tilde{\Sigma}_1$ and $\tilde{\Sigma}_2$ are totally interchangeable.

Theorem 4 (Local interchangeability)

Let Σ_1 and Σ_2 be locally interchangeable for μ of rank r, then there exists four pseudo-stochastic matrices B_1^L $(r \times c_1)$, B_1 $(c_1 \times r)$, B_2^L $(r \times c_2)$, B_2 $(c_2 \times r)$ such that

i) $B_1^L B_1 = B_2^L B_2 = I_r$

ii) $B_1^L m_1 B_1 = B_2^L m_2 B_2$

Theorem 5 (Dominance)

$\Sigma = (Z, m)$ dominates $\Sigma' = (Z, m')$

$$\Updownarrow$$

$\exists$ B $(c' \times c)$ stochastic such that

$$m'(\bar{z})\, e = Bm(\bar{z})\, e$$

$$\Updownarrow$$

$\exists$ B $(c' \times c)$ stochastic such that

$$m' (\bar{z})\ B\ m(\bar{z})\ e = B\ m(z)\ m(\bar{z})\ e$$

Corolary 1 (dominance)

if Σ' is full rank then $\exists\ B^L$ stochastic

B pseudo : $m' = B^L\ m\ B$

Corolary 2 (Dominance)

if rank (Σ) = rank (Σ') = c

$\exists\ B$ stochastic B^L pseudo-stochastic:

$m = B^L\ m'\ B$

Theorem 6 (Equivalence)

Two totally interchangeable full rank automata have identical algebraic representation up to a permutation.

/km

THE GROWTH OF WORD LENGTH IN DOL-SYSTEMS

BY

P.G. Doucet

Summary

This paper presents a recursive and an explicit formula for the growth function of a DoL-system. It also discusses

(a) the possible recursive formulas satisfied by a DoL-language

(b) the construction of a DoL-system satisfying a given recursive formula for word length or a given growth function.

It indicates the connection with (i) the property of being locally catenative (ii) the equivalence problem for DoL-systems.

1. INTRODUCTION

DoL-systems are exceptional among formal grammars in producing words in a fixed order. One can study the way in which the length of these words grows. This paper is concerned with the following problems:

- Given a DoL-system, find a recursive relation between its word lengths.
- Find its growth function.
- The inverse problems of these two problems.

A few additional problems arise along the way.

Remark: this paper poses more questions than it answers. As a whole, it is perhaps rather sketchy, and it should be regarded as an interim report rather than a finished study.

For a definition of a DoL-system, I refer the reader to such papers as Rozenberg and Doucet [4]. As regards notation, I shall denote a DoL-system G by a triple $<\Sigma, P, x_0>$ with Σ the alphabet, P the set of production rules and $x_0 \in \Sigma^+$ the axiom.

$\&(G)$ denotes the infinite sequence of words generated by G, in order of appearance. If a sequence can be generated by a DoL-system, it is called a DoL-sequence.

L(G) denotes the language generated by G.

$\#S$ denotes the number of elements of a set S.

$|x|$ denotes the length (= number of letters) of a word x.

If $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$, then the Parikh-vector $\bar{x}$ assigned to a word x is defined as a vector in $\mathbb{N}^k$ with its i-th coordinate equal to the number of occurrences of σ_i in x. Example: if $\Sigma = \{a,b,c\}$ and x = aacac, then $\bar{x} = (3,0,2)$.

Similarly, the set P of production rules can be mapped into the k×k production matrix $A_p = ((c_{ij}))$, where c_{ij} gives the number of occurrences of σ_i in $P(\sigma_j)$; in other words, the j'th column of A equals the Parikh-vector of $P(\sigma_i)$. Where no confusion is possible, A_p is also written A.

The Parikh-sequence of G (= the sequence of the Parikh-vectors) is denoted by $\bar{\&}(A_p, \bar{x}_0)$.

Example: Let $\Sigma = \{a,b,c\}$, $P = \{a \rightarrow bc, b \rightarrow aab, c \rightarrow ab\}$, $x_0 = ac$.

Then $A_P = \begin{pmatrix} 0 & 2 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ and $\bar{x}_0 = (1,0,1)$.

Now $\bar{\mathcal{E}}(Ap,\bar{x}_0) = (1,0,1),(1,2,1),(5,4,1),\ldots\ldots$.

If $\alpha_n f(m+n)+\alpha_{n-1} f(m+n-1)+\ldots\ldots+\alpha_0 f(m+1) = 0$ is a difference equation of order n, then

$\phi(x) = \alpha_n x^n+\alpha_{n-1}x^{n-1}+\ldots\ldots+\alpha_0$ is called its associated polynomial, and vice versa. (I do not use the more customary term "characteristic polynomial" to avoid confusion in the sequel).

I shall sometimes use the name "difference equation" in a slightly wider sense than usual by also applying it to mappings into $\mathbb{N}^k$ or $\mathbb{C}^k$ instead of to functions sensu stricto.

2. "CHARACTERISTIC" DIFFERENCE EQUATION ; GROWTH FUNCTION

Lemma 1. Let $G = < \Sigma, P, x_0 >$ be a DoL-system with $\# \Sigma = k$.

Then the Parikh-sequence $\bar{\&}(Ap,\bar{x}_0)$ satisfies a difference equation of order k which is the associated equation of A_p's characteristic polynomial.

Proof: The vextors $\bar{x}_0,\bar{x}_1,\bar{x}_2,....$ can be written as $\bar{x}_0, A\bar{x}_0, A^2\bar{x}_0,....$. Application of Cayley & Hamilton's theorem produces the required result.

Example (continued):

$\phi_A(\lambda) = -\lambda^3+\lambda^2+3\lambda+1$. By lemma 1, $\bar{x}_{n+3} = \bar{x}_{n+2}+3\bar{x}_{n+1}+\bar{x}_n$ for every $n \geqslant 0$.

Note that

1 the difference equation is always monic with integer coefficients.

2 the sequence $|x_0|,|x_1|,|x_2|,...$ of word lengths satisfies the same equation.

Lemma 1 and the theory of difference equations produce an explicit expression for the growth function of a DoL-system (see also Paz and Salomaa, [2]).

Lemma 2. Let $G = < \Sigma, P, x_0 >$ be a DoL-system with $\#\Sigma = k$.

If A_p has eigenvalues $0,\lambda_1,...,\lambda_m$ with multiplicities $k_0,k_1,...,k_m$ respectively ($k_0+k_1+...+k_m = k$), then $|x_n| = \mathrm{pol}_{k_1-1}(n)\ \lambda_1^n + ... + \mathrm{pol}_{k_m-1}(n)\ \lambda_m^n$ for all $n \geqslant k_0$, where $\mathrm{pol}_i(n)$ denotes an $i^{\underline{th}}$ degree polynomial in n; these polynomials are determined by the initial word lengths $|x_{k_0}|,....,|x_k|$.

Lemma 2 gives the overall growth function of &(G) and can be considered as the k individual growth functions (of the different letters of Σ) lumped together. In fact, the individual growth functions can be kept separate; they are represented by the sequence

$\bar{\&}(Ap,\bar{x}_0)$, for which lemma 3 gives an expression.

To obtain

<u>Lemma 3</u> from lemma 2, replace the various $|x_i|$ by $\bar{x}_i$ throughout, and replace the coefficients of the polynomials $pol_i(n)$ by k-vectors (thereby extending the notion of a polynomial somewhat).

Example (continued):

A_p has eigenvalues $\lambda_1=1, \lambda_2=1-\sqrt{2}, \lambda_3=1+\sqrt{2}$. In terms of lemma 3, $k_0=0$, $k_1=1$, $k_2=1$, $k_3=1$.

So $\bar{x}_n = (-1)^n\bar{u}_1+(1-\sqrt{2})^n\bar{u}_2+(1+\sqrt{2})^n\bar{u}_3$, where the 3-vectors $\bar{u}_1,\bar{u}_2,\bar{u}_3$ ("extended polynomials" of degree 0) are determined by the initial vectors $\bar{x}_0,\bar{x}_1,\bar{x}_2$. Solving of the equations

$$(1,1,0) = \bar{u}_1+\bar{u}_2+\bar{u}_3$$
$$(2,2,1) = -\bar{u}_1+(1-\sqrt{2})\bar{u}_2+(1+\sqrt{2})\bar{u}_3$$
$$(5,5,2) = \bar{u}_1+(1-\sqrt{2})^2\bar{u}_2+(1+\sqrt{2})^2\bar{u}_3$$

yields

$$x_n = (-1)^n(0,0,0)+\left(\frac{\sqrt{2}-1}{2\sqrt{2}},\frac{\sqrt{2}-1}{2\sqrt{2}},\frac{-1}{2\sqrt{2}}\right)(1-\sqrt{2})^n+\left(\frac{\sqrt{2}+1}{2\sqrt{2}},\frac{\sqrt{2}+1}{2\sqrt{2}},\frac{-1}{2\sqrt{2}}\right)(1+\sqrt{2})^n.$$

Apparently the "ordinary" DoL growth function is a sum of exponentials (this includes oscillating behaviour, if A_p has complex eigenvalues). In case of multiple eigenvalues, growth can be mixed (exponential/polynomial). In the special case of all non-zero eigenvalues being ± 1 growth is purely exponential.

In a given DoL-system, the growth type (exponential, polynomial or mixed) may or may not depend on the axiom (Salomaa, [7]).

Note that if A_p has null columns ($k_0 \neq 0$; G is not propagating), the associated difference equation degenerates and has order $k-k_0$.

The effect is that in such a case the expressions in lemmas 2 and 3 do not hold for the first k_0 words of $\&(G)$.

3. OTHER DIFFERENCE EQUATIONS. INVERSE PROBLEMS

We saw that, given a DoL-system, both a recursive and an explicit expression can be found for the sequence $\bar{x}_0,\bar{x}_1,\bar{x}_2,\ldots$. A few other problems now arise naturally.

Problem 1

What other difference equations are satisfied by the sequence $\bar{x}_0,\bar{x}_1,\bar{x}_2,\ldots$?

Problem 2

Find a DoL-system satisfying a given difference equation.

Problem 3

Find a DoL-system to a given growth function.

Problem 1 is largely answered by lemmas 4 and 5:

Lemma 4. Let ϕ_A be the characteristic polynomial of A.
For any polynomial ψ with integer coefficients, the sequence $\bar{\mathcal{E}}(A,\bar{x}_0)$ satisfies the difference equation associated with $\phi_A\cdot\psi$.

Proof: Easy. As yet, only ψ's with integer coefficients seem to make any sense, but the restriction is of course not necessary.

Lemma 5. Let $\#\,\Sigma = k$, and $j \leqslant k$.
$\bar{\mathcal{E}}(A,\bar{x}_0)$ satisfies a difference equation of order j iff both

- its associated polynomial ψ is a divisor of ϕ_A;
- $\bar{x}_0$ is in the subspace of $\mathbb{R}^k$ induced by ψ (i.e. the null space of $\psi(A)$).

Proof: straightforward.

Example (continued):

- Multiply ψ_A by $\psi(\lambda) = \lambda^2+2$: $\phi_A\cdot\psi(\lambda) = -\lambda^5+\lambda^4+2\lambda^3+2\lambda^2+3\lambda+1$.

- Now $\bar{\mathcal{E}}(A,\bar{x}_0)$ satisfies the associated difference equation $\bar{x}_{n+5} = \bar{x}_{n+4}+2\bar{x}_{n+3}+2\bar{x}_{n+2}+3\bar{x}_{n+1}+\bar{x}_n$.
- $\chi(\lambda) = -\lambda^2+2\lambda+1$ is a divisor of $\phi_A(\lambda)$. Now $\bar{\mathcal{E}}(A,\bar{x}_0)$ satisfies the associated equation $\bar{x}_{n+2} = 2\bar{x}_{n+1}+\bar{x}_n$ iff $\bar{x}_0$ is dependent on the eigenvectors corresponding to the roots of χ. Here this means that $\bar{x}_0$ has to be of the form $\alpha_1(1+\sqrt{2},1+\sqrt{2},1)+\alpha_2(1-\sqrt{2},1-\sqrt{2},1)$. $\bar{x}_0$, however, must have integer coordinates, and this further restricts the possible form of $\bar{x}_0$ to $\alpha(1,1,1)$.

The influence of the last-mentioned restriction has yet to be studied.

Problem 2 has a partial answer:
If $\bar{x}_{n+k} = \alpha_{k-1}\,\bar{x}_{n+k-1} + \dots + \alpha_1\,\bar{x}_{n+1}+\alpha_0\bar{x}_n$ is the given equation, then the companion matrix of the associated polynomial,

$$A = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \theta \\ & \theta & \ddots & \ddots & \\ & & & 0 & 1 \\ \alpha_0 & \alpha_1 & \cdots & \cdots & \alpha_{k-1} \end{pmatrix}$$

and the corresponding set of production rules

$$\left\{ \begin{array}{l} \sigma_1 \to \sigma^{\alpha_0} \\ \sigma_2 \to \sigma_1\sigma_k^{\alpha_1} \\ \quad\vdots \\ \sigma_k \to \sigma_{k-1}\sigma_k^{\alpha_{k-1}} \end{array} \right\} \text{(or permutations)}$$

provide a solution (as does the transpose of A), for any axiom x_0. The only trouble is that A may have (and often has) negative entries, which of course disqualifies it as a grammar-representing matrix. So problem 2 reduces to the more general

Problem 4

Given a matrix with entries in $\mathbb{Z}$, find a similar matrix with entries in $\mathbb{N}$.

Except for some trivial cases, I have no solution.

A different approach is, perhaps, also feasible. As shown above, an equation which has an associated polynomial with leading coefficient 1 and the other coefficients negative (in which case I shall call both equation and polynomial monic-negative) is immediately translatable into a DoL-system. Now, given a difference equation which is not monic-negative, one can try to "multiply" it with some other equation to obtain an equation which is monic-negative. The companion matrix corresponding to this product, together with an appropriate axiom, would produce a sequence satisfying the given equation.
In a better formulation:

Problem 5

Given a difference equation whose associated polynomial ψ is not monic-negative, find a polynomial χ and a positive vector $\bar{x}_0$ such that 1. $\psi \cdot \chi$ is monic-negative.
2. the sequence $\bar{x}_0, A\bar{x}_0, A^2\bar{x}_0, \ldots$ satisfies the given equation.

Although in principe χ is free, it is easy to show that only χ's with integer coefficients can provide a solution.

Requirement 1 generates a set of inequalities. The answers I obtained to the pertinent problems are as yet too scattered to make worthwile reading, and I omit most of them. For some ψ, the required χ does not exist, such as for all ψ of the form $\psi(x) = x^n + \alpha_{n-1}x^{n-1} + \ldots + \alpha_2 x^2 + \alpha_1 x - \alpha_0$ (with all α_i positive). Often, a χ does exist: for $\psi(x) = x^2 - x + 3$ (not monic-negative), multiplication with $\chi(x) = x^3 + x^2 - 2x - 5$ yields $x^5 - x - 15$ (monic-negative).
The existence of such a polynomial χ is of some importance for studying the property of being locally catenative as discussed by

Rozenberg and Lindenmayer [5]. The connection is a simple one: in order to be locally catenative, a DoL-sequence (or rather its Parikh-sequence) must satisfy a monic-negative difference equation. The converse is of course not true.
Conversely, if one is looking for locally catenative properties in a DoL-language, one only has to examine the difference equations satisfied by its Parikh-sequence, which considerably narrows down the search area (the more so since only monic-negative equations can lead to meaningful concatenations).

The second requirement of problem 5 can be restated as $\psi(A)\bar{x}_0 = \bar{0}$ (by lemma 5). It is much harder to fulfill than the first requirement. In fact, I am not even sure that it can be fulfilled at all.

Problem 3 quickly leads to one or two other problems. If the growth function is given in the form

$$f(n) = \mathrm{pol}_{k_1}(n)\lambda_1^n + \ldots + \mathrm{pol}_{k_m}(n)\lambda_m^n \ , \qquad (*)$$

then by lemma 2 any matrix A with characteristic polynomial

$$\phi_A(x) = (x-\lambda_1)^{k_1+1} \ldots . (x-\lambda_m)^{k_m+1}$$

will realize f, provided that an axiom can be found yielding the given coefficients of the polynomials $\mathrm{pol}_{k_1}, \ldots, \mathrm{pol}_{k_m}$.
First of all, there is the matter of finding such an axiom vector. This involves the solving of a number of equations which are nonlinear except in the case of all λ's being 1 or -1 (that is, purely polynomial growth). Secondly the axiom vector must consist of positive integers, which imposes a heavy restriction on f. An f for which such an axiom exists can be called an <u>admissible growth function</u>.

<u>Problem 6</u>
For a given function of the form (*), determine whether it is an admissible growth function.

Open.

4. DECISION PROBLEMS

A number of decision problems are, or might be, connected with growth functions.

Problem 7 Given two DoL-systems $G_A = < \Sigma_A, P_A, x_0 >$ and $G_B = < \Sigma_B, P_B, x_0 >$, decide whether they are Parikh-equivalent; that is, whether $\bar{\&}(A,x_0) = \bar{\&}(B,\bar{x}_0)$.

Solution: the sequences are equal iff the first k+1 Parikh-vectors (with numbers 0,....,k) are pair-wise equal, where $k = \min(\#\Sigma_A, \#\Sigma_B)$.

Proof: Without loss of generality, assume that $\#\Sigma_A \leqslant \#\Sigma_B$.
Let ϕ_A and ϕ_B be the characteristic polynomials of A and B. Obviously, $\phi_A(A)\bar{x}_0 = \bar{0}$ and $\phi_B(B)\bar{x}_0 = \bar{0}$.
If the sequence $\bar{\&}(A,\bar{x}_0)$ can be shown to satisfy the difference equation associated with ϕ_B, it follows by induction that the elements of $\bar{\&}(A,\bar{x}_0)$ and $\bar{\&}(B,\bar{x}_0)$ are pair-wise equal. In other words, it suffices to prove that $\phi_B(A)\bar{x}_0 = \bar{0}$.
If ϕ_A has coefficients $\alpha_{k-1},\dots,\alpha_0$, then

$$\phi_A(B)\bar{x}_0 = B^k\bar{x}_0 + \alpha_{k-1}B^{k-1}\bar{x}_0 + \dots + \alpha_0\bar{x}_0$$

$$= A^k\bar{x}_0 + \alpha_{k-1}A^{k-1}\bar{x}_0 + \dots + \alpha_0\bar{x}_0 = \phi_A(A)\bar{x}_0 = \bar{0}.$$

Since $\phi_A(B)\bar{x}_0 = \bar{0}$ and ϕ_A is of lower degree than ϕ_B, ϕ_A must be a divisor of ϕ_B (by lemma 5). By lemma 4, $\phi_A(A)\bar{x}_0 = \bar{0}$ now implies $\phi_B(A)\bar{x}_0 = \bar{0}$.

Problem 8 Given two DoL-systems, decide whether they have the same growth functions.

Open.

Problem 9 Determine whether two DoL-sequences $\&(G_1)$ and $\&(G_2)$ are equal.

Open.

Problem 10. Determine whether two DoL-languages $L(G_1)$ and $L(G_2)$ are equal.

Open. For oL-languages, and even propagating oL-languages, the problem is undecidable (Blattner [1], Salomaa [6]).

References

[1] Meera Blattner, The unsolvability of the equality problem for sentential forms of context-free languages. To be published.

[2] A. Paz and A. Salomaa, Integral sequential word functions and growth equivalence of Lindenmayer systems. To appear in *Information and Control* (1972)

[3] G. Rozenberg, The equivalence problem for deterministic ToL-systems is undecidable. State Univ. of New York at Buffalo, Department of Computer Science Report 24-72 (1972)

[4] G. Rozenberg and P.G. Doucet, On oL-languages. *Information and Control* *19* (1971), 302-318.

[5] G. Rozenberg and A. Lindenmayer, Developmental systems with locally catenative formulas. Submitted to *Acta Informatica*.

[6] A. Salomaa, On sentential forms of context-free grammars. To appear in *Acta Informatica*.

[7] A. Salomaa, On exponential growth in Lindenmayer systems. To appear in *Indag.Math*.

[8] A. Szilard, Growth functions of Lindenmayer systems. University of Western Ontario, Computer Science Department Technical Report no. 4 (1971).

SYNCHRONIZATION OF GROWING CELLULAR ARRAYS

Gabor T. Herman, Wu-Hung Liu,
Stuart Rowland and Adrian Walker

Department of Computer Science
State University of New York at Buffalo
4226 Ridge Lea Road
Amherst, New York 14226

ABSTRACT

Generalized versions of the firing squad synchronization problem are investigated. The generalizations consist of allowing the linear array of automata to grow (by division of automata into two or more) while it is trying to synchronize itself. Solutions are obtained for cases when growth takes place only at the ends of the array (irrespective whether or not the rate of growth is the same at both ends) and also for cases when growth takes place in the middle of the array. Biological motivations for the generalizations are discussed.

1. Introduction

The Firing Squad Synchronization Problem (from now on FSSP) is so well known that we shall not give a detailed statement of it. Many of the references mentioned below will provide the reader with a detailed discussion. A concise statement of the problem is the following.

'Find a two input, two output automaton (Moore type) with three distinguished states (s, i and f, say) and an arbitrary but finite number of additional states, such that a one dimensional array of such automata (with the end automata constantly receiving one special input each, which shows them that they are end automata) will have the following properties:

(i) If all the automata are in state s, they will all remain in state s.

(ii) If one of the end automata is in state i, while the rest of the automata are in state s, the array will undergo a series of transitions ending up with all the automata being in state f. Furthermore, no automaton will be in state f prior to all the others being in state f.'

The FSSP originates from J. Myhill, and, according to Moore (1964), it first arose in connection with causing all parts of a self-reproducing machine to be turned on simultaneously. It was first solved by M. Minsky and J. McCarthy. Waksman (1966) gave a solution which for an array of n automata achieves synchronization in 2n - 2 steps. It can easily be shown that this time cannot possibly be made shorter. Balzer (1967) gave such a minimal time solution using only eight states. Varshavsky, Marakhovsky & Peschansky (1970) discuss some generalized versions of the FSSP. The generalizations include allowing the initial disturbance, i, to take place anywhere in the array; allowing different types of automata in the array, each type having a different speed of reaction to the initiating signal; allowing a fixed number of delays between the automata which are required to synchronize irrespective of the number of delays; and allowing a random reconnection of the automata in the array before each step (i.e., in each step, the automata are randomly partitioned into pairs, and each automaton will influence in that step only its partner in the pair). In all these generalizations, the total number of automata remains fixed during the synchronization process. Smith (1971) made repeated use of solutions of the FSSP in his two dimensional pattern recognizing cellular automata. Herman (1971b) gave a solution to the FSSP where the automata were symmetric, i.e., they could not distinguish between their left and right inputs.

In this paper we shall consider generalizations of the FSSP, where the array is allowed to grow during the synchronization process. Growth will take place by allowing the individual automata to divide into two or more automata. For this reason, rather than the more traditional type of cellular automata (see, e.g. Codd, 1968), we shall use the models proposed by Lindenmayer (1968a, b), and since then developed by Baker & Herman (1972a, b), van Dalen (1971), Doucet (1971), Herman (1969, 1970, 1971a, 1971b), Lindenmayer (1971), Rozenberg & Doucet (1971). These models allow an automaton to divide anywhere in the array, provided that itself and its two neighbors are in appropriate states.

The reason for the generalization to growing arrays is a logical extension of the original motivation. It is reasonable to assume that an organism, all of whose parts we wish to turn on simultaneously, is still growing during the synchronization process. More specifically, our attention was brought to this generalization by the study of molluscan pigmentation patterns. In an earlier study (Baker & Herman, 1972a), we have shown how the pigmentation pattern (see Plate I), originally studied by Waddington & Cowe (1969), can be discussed in terms of Lindenmayer models. We wished to extend this study to include pigmentation patterns found on the shells of different kinds of sea snails. We found that many of these (Plates II, III, IV) have recurring in them dark areas which get uniformly wider as growth takes place, and then suddenly disappear, giving us shapes of the type shown in Figure 1. If such shapes are to be explained on the basis of cellular interactions (rather than on the basis of an external influence), we must conclude that cells, which are depositing the dark region, must have succeeded in synchronizing their action to stop creating the dark pigment at the same time, even though their number was growing during the synchronization process. (For those who are interested to see more molluscan pigmentation patterns, a good reference is Marsh & Rippingdale, 1964.)

In the next section, we give a precise statement of the generalized FSSP in terms of Lindenmayer models. In section 3, we discuss the improtant preliminary problem of synchronizing the two end automata of a growing array with automata in the middle of the array. In section 4, we discuss our solution of the case when growth takes place only at the ends. In section 5, we discuss the implementation on a computer of our solution for the case of growth at the ends. In section 6, we give a solution in which growth takes place in the middle of the array. In section 7, we summarize our results and show how similar techniques can be used to solve other problems for growing arrays of automata, for example a generalization of the French flag problem of Wolpert (1968).

For reasons explained in earlier works (Herman 1971a, 1971b), we shall in all cases attempt to give solutions using symmetric automata.

2. Definitions and problem statement

If G is a non-empty finite set, G^* denotes the set of all sequences of elements of G. G^* includes the empty sequence, which is denoted by ϵ.

A Lindenmayer model L is a quadruple $<G, g, \delta, F>$, where G is a non-empty finite set of states, $g \in G$ is called the standard environmental input, δ is a function giving for the states of any three consecutive automata (cells) the sequence of states of the automata (cells) by which the middle one is replaced (δ: $G \times G \times G \rightarrow G^*$) and F is a subset of G. In this paper we shall have no occasion to use F.

For any filament $\underline{p}$, let $L(\underline{p})$ denote the length of $\underline{p}$. For any symbol a in G and any non-negative integer l, let a^l denote a sequence of l instances of a. For any rational number κ, let $[\kappa]$ denote the greatest integer which is not greater than κ, and $|\kappa|$ denote the absolute value of κ. Using this terminology, the generalized FSSP may be stated as follows.

<u>Problem 1.</u> Given any integers p, g, r, t, $0 \leqq p < q$, $0 \leqq -r < t$, find a propagating Lindenmayer model $\underline{L} = \langle G, g, \delta, F \rangle$, such that s, i, f are elements of G (all distinct from each other and from g) and $\underline{L}$ has the following properties.

(i) For any non-negative integers l and n, there exists a non-negative integer $k \geq l$ and symbols b and c in G, such that $\lambda^n (s\, s^l\, s) = b\, s^k\, c$.

(ii) For any non-negative integers l, k, and n $L(\lambda^n (s^l i s^k) = l + k + 1 + \left[\frac{np}{q}\right] + \left[\frac{n|r|}{t}\right]$,

(iii) For any non-negative inteders l and k, there exists a positive integer m, such that, for all positive integers $n < m$, $\lambda^n (s^l i s^k)$ does not contain the symbol f, but $\lambda^m (s^l i s^k)$ does not contain a symbol other than f.

In this problem statement, (i) tells us that a filament in which all cells are in state s will remain stable except, possibly, at the ends. We exclude the ends, since we want to allow the possibility that the growth described in (ii) will take place at the ends of the filament. In this case p/q and r/t are the growth rates at the two ends respectively, i.e., the filament grows p cells in a unit of time at the right end and r cells in t units time at the left end. The negativeness of r indicates growth right to left. The synchronization of the filament, after an initial disturbance i anywhere in the filament, is stated in (iii).

A Lindenmayer model is said to be *propagating* if, and only if, $\delta(d, b, c) \neq \varepsilon$, for any $d, b, c \in G$. So in a propagating Lindenmayer model cells cannot simply disappear. We shall only be discussing propagating Lindenmayer models.

If $\underline{L} = \langle G, g, \delta, F \rangle$ is a Lindenmayer model, any element of G^* is said to be a *filament* of $\underline{L}$. (From now on we shall always refer to the individual automata as cells, and linear arrays of the states of automata as filaments.)

It was seen that the standard environmental input is an element of G. It is a basic assumption of Lindenmayer models that environment affects a filament only at the end cells, and its effect can be described by elements of G. Herman (1970) proved that by constantly changing the environment, one can induce practically any type of behaviour in a filament. Since we want a filament to synchronize due to the interaction between its cells, and not due to environmental changes, we shall from now on assume that all development takes place in a constant environment. In view of this, we can give the following definitions.

For any filament $\underline{p}$ of a Lindenmayer model $\underline{L} = \langle G, g, \delta, F \rangle$, we define $\lambda_{\underline{L}}(\underline{p})$ (or $\lambda(\underline{p})$ when there is no possible concfusion about the $\underline{L}$), the *consecutive filament* as follows. If $\underline{p}$ is of the form $a_1 a_2 \ldots a_k$ $(k \geq 2)$, then $\lambda(\underline{p}) = \underline{q}_1 \underline{q}_2 \ldots \underline{q}_k$, where

$$\underline{q}_i = \delta(a_{i-1}, a_i, a_{i+1}), \quad \text{for } 2 \leq i \leq k-1,$$

$$\underline{q}_1 = \delta(g, a_1, a_2),$$

$$\underline{q}_k = \delta(a_{k-1}, a_k, g).$$

If $\underline{p} = \varepsilon$, then $\lambda(\underline{p}) = \varepsilon$, and if $\underline{p} \in G$, then $\lambda(\underline{p}) = \delta(g, \underline{p}, g)$.

For all non-negative integers n we define $\lambda^n_{\underline{L}}(p)$ (or $\lambda^n(\underline{p})$), the *n'th consecutive filament* of $\underline{p}$, by

$$\lambda^0(p) = p,$$

$$\lambda^{n+1}(p) = \lambda(\lambda^n(p)).$$

The restriction that $p < q$ and $|r| < t$ is introduced for the following reason. Suppose that growth takes place at the ends due to cell division, so the right end cell divides into p cells in a time q. A disturbance set up by i somewhere in the middle of the filament can propagate at most with the speed of one cell per unit time. If $p \geq q$, then such a disturbance can never reach the right end, let alone lead to a synchronization of the whole filament.

If we forgo the possibility that growth may take place at the ends of the filament, we can have the following stronger formulation.

Problem 2. Same as Problem 1, but replace (i) by

(i') For any non-negative integers l and n,

$$\lambda^n(s^l) = s^l.$$

In a solution of Problem 2, there will be no growth in a filament which contains only the symbol s, but growth will commence immediately after the disturbance. Hence, growth must take place near the disturbance. Cells far away from the disturbance cannot be immediately influenced by the disturbance, since in one step a cell can only influence its immediate neighbors.

It is biologically interesting (see,e.g., Herman 1971a, b) to investigate whether synchronization is possible with a model in which the individual cells lack polarity, i.e., they are symmetric. This notion is made precise by the following definitions.

A Lindenmayer model $\underline{L} = \langle G, g, \delta, F \rangle$ is said to be externally symmetric if, and only if, δ has the property that

$$\delta(d, b, c) = \delta(c, b, d)$$

for all $d, b, c \in G$. $\underline{L}$ is said to be internally symmetric if, and only if, δ has the property that

$$\delta(d, b, c) = [\delta(d, b, c)]^R$$

for all $d, b, c \in G$. ($\underline{p}^R$ denotes the filament $\underline{p}$ written in reverse order.) $\underline{L}$ is said to be symmetric if, and only if, it is both externally symmetric and internally symmetric.

Problems 1'(a, b, c) and 2'(a, b, c). Same as Problem 1 and 2, respectively, except that we insist that $\underline{L}$ should be

(a) externally symmetric,
(b) internally symmetric,
(c) symmetric.

For an externally symmetric Lindenmayer model $\delta(g, s, s) = \delta(s, s, g)$, and growth at different rates at the ends immediately upon a disturbance in the middle is impossible. The following version makes different growth rates possible, by having distinguished symbold at the ends.

Problem 3'(a, b, c). Given any integers p, q, r, t, $0 \leqq p < q$, $0 \leqq -r < t$, find an

(a) externally symmetric, respectively,
(b) internally symmetric, respectively,
(c) symmetric,

Lindenmayer model $\underline{L} = \langle G, g, \delta, F \rangle$, such that s, i, f, b, c are elements of G (all distinct from each other and from g) and $\underline{L}$ has the following properties.

(i) For any non-negative integers l and n, there exists a non-negative integer $k \geqq l$, and symbols b' and c' in G, such that

$$\lambda^n(bs^lc) = b's^kc'.$$

(ii) For any non-negative integers l, k, and n

$$L(\lambda^n(bs^lis^kc)) = l + k + 3 + \left[\frac{np}{q}\right] + \left[\frac{n|r|}{t}\right].$$

(iii) For any non-negative integers l and k, there exists a positive integer m, such that, for all positive integers $n < m$, $\lambda^n(bs^lis^kc)$ does not contain the symbol f, but $\lambda^m(bs^lis^kc)$ does not contain a symbol other than f.

Our solution to all the problems mentioned in this section will have the following basic form. At the beginning, there will be a preparatory stage, followed by a number of standard stages. At the beginning of each of the standard stages, the filament will have the following properties. It contains a number of cells which are "markers". These markers occur either singly or in pairs, but, except for the beginning of the last stage, there are never more than two consecutive markers. In particular, the two end cells are markers. The number of consecutive non-marked cells is the same everywhere in the filament, i.e., the markers divide the filament into subfilaments of equal length. Further more, except for the last stage, if the length of these subfilaments is l, then the length of the subfilaments at the beginning of the next stage is $\left[\frac{l}{2}\right]$. This implies that, irrespective of the length of the initial filament, we arrive at a stage, at the beginning of which the number of consecutive non-marker cells is everywhere 0, i.e., the whole filament is made up from markers. Furthermore, this is the first time that there is a marker cell in the filament such that both its neighbors are also markers. Thus, if we require the living state f to appear if, and only if, the cell under consideration is a marker with both its neighbors also markers, then synchronization will be achieved.

What we have described above applies equally to standard solutions of the FSSP (Waksman, 1966, Balzer 1967, Varshavsky, et ol, 1970, Herman, 1971b). However, in that problem the filament is not growing, and so the number of subsegments at the beginning of each stage is twice the number of subsegments at the beginning of the previous stage. Due to growth, this will no longer be the case for the generalized problems discussed in this paper. The placing of markers to achieve the finer and finer subdivisions becomes therefore a somewhat more complicated process. A thorough discussion of this process, when growth takes place at the ends, will be the subject matter of the next three sections.

CELLULAR AUTOMATA, FORMAL LANGUAGES AND DEVELOPMENTAL SYSTEMS

by

ARISTID LINDENMAYER

University of Utrecht, The Netherlands

A Contribution to the Symposium on Cellular Automata and Their Significance to the Foundations of Biology,

held as part of the IVth International Congress for Logic, Methodology and Philosophy of Science,

Bucharest, Romania

1971

By kind permission reprinted from:
P. Suppes et al. (eds.), Logic, Methodology and Philosophy of Science IV
North-Holland, Amsterdam, 1973.

1. "Cellular automata" are formal constructs which in the last decades came into increasing use as biological models. The first two, and still important cellular-automata constructs with biological motivation were the nerve-net models of McCulloch and Pitts (1943) and the self-reproducing cellular automata of von Neumann (published posthumously by Burks, 1966). Nerve-net models have been constructed to exhibit various kinds of simple behaviour like reflex loops or the looming response of frogs. Self-reproducing cellular automata are meant to demonstrate how very complex structures can be constructed from relatively simple components and be able to replicate themselves. The term "cellular" refers in this case to the subunits with which this construction is carried out, and does not imply an analogy of these subunits with cells of living organisms. In fact, the whole self-reproducing structure might be considered to be analogous to a single living cell.

Both of these models consist of a number of interconnected simple units. These units are switching elements in the model of McCulloch and Pitts; and input-output devices with a small number of internal states, but without any other additional memory, in the case of von Neumann's system. Both of these kinds of units are examples of what are now called "finite automata", i.e. machines with finite number of states, and sequential generation of outputs as determined by the sequence of inputs and the configuration of states. The interconnections of the units in the first model can be arbitrary in form and richness, while in the second one the units are placed in a rigid square grid, and are connected in uniform, time-invariant way with neighbouring units, e.g. to exactly four neighbours, in the von Neumann model. While McCulloch's and Pitts' constructs were subsequently shown (Kleene, 1956) to be not any more powerful computing machines than their own subunits the finite automata, the construction of von Neumann turned out to be equivalent in computing ability to Turing machines, i.e., to the most general and powerful computing devices. This difference in computing power between the two kinds of models is a consequence of the fact that the first ohes are of stable size, they consist of a constant number of units and interconnections, while the latter ones can expand, and computation can be carried out with increasingly larger numbers of subunits.

Cellular automata of both kinds have been further investigated and their biological usefulness have been commented upon by several workers (Codd, 1968, on simplified models of self-reproducing automata; Arbib, 1969, a, b, on the use of cellular ways in simulating regeneration and on self-reproducing systems in which the neighbours of a cell are allowed to change; Vitányi, 1972, on sexually reproducing automata; several of these and others being also mentioned in Burks, 1968).

2. Cellular - automata models have also been introduced (Lindenmayer, 1968) with reference to the development of organisms, rather than their self-reproduction, or their nervous behaviour[1]. In this case, the cells of the model are assumed to correspond to individual cells of growing multicellular organisms. Living cells are well established as autonomous units of metabolism, heredity and physiological function. In these models cells are, in fact, construed to be finite automata, which in any moment may be present in one of finite number of states, can receive one of ~~the~~ a finite number of inputs from neighbouring cells and give rise to one of finite number of outputs, the outputs in turn serving as inputs to other cells. As in all such models, time is to pass in discrete steps, and, whether a cell changes its state, divides, or dies at the next moment is determined by its present state and by the present inputs it receives. The outputs of cells at each moment are also determined by their states and inputs. Since cells can divide or die, the number of cells

may increase or decrease from moment to moment. Thus one can obtain a series of cellular arrays, each corresponding to a momentary developmental stage of an organism, beginning with a single cell, the fertilized egg, and increasing or decreasing in size. Furthermore, morphogenesis is simulated by the production of stable patterns in the array, provided that certain cellular states are such that they change no further under any input combinations.

It may well be asked whether these kinds of automata-theoretical models can at all be interpreted in terms of the biochemical mechanisms supposed to underlie development and morphogenesis. Goodwin (1970) posed this question in the following way: "The implication of the computer analogy is that the cell computes its own state, looks at the DNA program for further instructions, and then changes state accordingly. This is not in fact what a cell does, although formal analogy can be made between the biochemical behaviour of a cell and the operation of an automaton following a program. It may seem elementary to insist that all the operations of the automaton must at some point be interpreted in biochemical and physiological terms, when discussing such a process as epigenesis[2)], but I have been somewhat dismayed at the amount of confusion that has arisen because of the failure of those using the computer analogy to illustrate the operation of algorithmic instructions at the biochemical level".

In fact, the concept of finite automaton is general enough to give it the desired interpretation in biochemical and cell-physiological terms, one needs only to elaborate it somewhat. We may safely assume first of all that the same genes (or operons) are present in every cell of a certain organism, and that each gene may be present in an active or a passive form (masked or unmasked) and thus at each moment in each cell there is a particular combination of active and inactive genes. We can assume furthermore that in each cell the active genes produce specific enzymes, which under the proper conditions are able to convert various metabolites into other compounds, some of which accumulate in the cell, resulting in its differentiation. The change of particular genes from inactive to active form, or vice versa, is presumably determined by the product (an inducer) of some enzyme, together with a repressor protein. Whether the inducer comes from the same or another cell, we observe the induction of the synthesis of a new enzyme, which then might result in differentiation, or in the initiation of a new cell line. In a very short summary, these are the currently accepted hypotheses concerning differentiation and induction, and they represent an important portion of the mechanisms underlying development (leaving aside for the time being the spatial orientation of cell division and enlargement which represent another important aspect). We recognize of course that many different control mechanisms may be operating on the synthesis and activity of various enzymes (as e.g. discussed by Waddington, 1968); nevertheless, all of these controls can be expressed as either turning on and off the genes, or as affecting the activities of the enzymes and thus the appearance of their products.

Now as for the automata-theoretical connections of all these statements. Let us first consider all metabolites and other cellular components, except for the active proteins and nucleic acids (informational macro-molecules), and call the set of all the species of such components C.

At each moment in a given cell there would be present (in significant amounts) a particular combination of the elements of C, i.e., the cell can be associated with a particular subset of C. Let us further consider each gene and the enzyme it gives rise to as a transformation rule, transforming certain combinations of cell components (elements of C) into other components, and call the set of all such transformation (or production) rules P. In each cell at a given time there will be a certain combination of genes active, concomitantly there being also the corresponding enzymes active, i.e., the cell will possess productions representing a subset of P. To each cell at each moment one can thus assign a *state*, consisting of a particular subset of C and a particular subset of P. The *inputs* of a cell consist of the compounds (elements of C) which have entered the cell during the last time interval, and the *outputs* of the cell consist of the compounds which have left it (in significant amounts)[3].

The *next state* of the cell will depend entirely on its present state and the inputs it receives, according to some specifiable functions.

First, one needs a function which determines the next subset of P, i.e., the combination of active and inactive genes (and their enzymes) at the next moment, including also the possibility that two new subsets of P are to be specified (the cell divides), or no new subset of P is specified (the cell dies). The combination of cell components at the next moment will be determined by the present combination of components, by the input to the cell, and by the subset of P applicable to the cell at the time (the elements of P being rules which specify the interconversion of components). If the cell divides, it has to be further specified whether the cell components are to be divided equally or unequally between the daughter cells (one more function is needed in the latter case).

The *output* of a cell, i.e., the compounds which diffuse or are transported from a cell into the neighbouring cells, will depend only on the components present in the cell (assuming that it is known which cell components are transportable). If an inducible permease is involved in the transport of a compound, then the production of the permease is controlled as that of an enzyme.

It can be seen, that both the next-state and the output functions can in principle be constructed for a given population of cells, thus the cells can be validly represented by finite automata. I cannot agree, therefore, with Goodwin's objections. What he calls the "DNA program" is the set of production rules we introduced, and according to our short discussion above it can be said in a completely natural way that the next state of a cell is computed by using these production rules.

The only serious objection against these sort of mathematical constructs, that I can envision, is an objection against their discreteness. It is quite true that not only time and space appear here in discrete units (the time steps and naturally the cells themselves are these units), but the enzymes, metabolites and inducers must also be considered as discrete entities, being either present in concentrations above certain thresholds or not. The genes are of course always present in integer units anyway. Having to deal with discrete temporal and concentration parameters is clearly a disadvantage of this approach one must recognize. This is counterbalanced, however, in my view, by great conceptual and practical computational advantages, some of which will presently become more apparent, I hope.

3. As for our developmental models, up to now we have investigated primarily one-dimensional cellular arrays. Each cell in the array is construed to be a finite automaton with uniform sets of states and next- state functions throughout the array. For the sake of mathematical simplicity, we have considered states and outputs to be identical and consequently the inputs to be the states of neighbouring cells. Under these simplifying assumptions, a filamentous developmental system (called a Lindenmayer model by Herman, 1969, 1970, or an L- system by van Dalen, 1971) is a construct consisting of a finite set of states, a next-state function, and an initial array.

This construct is formally somewhat analogous to grammars, as the term is used in formal language theory (see e.g. Aho and Ullmann, 1968). There the set of states is called an alphabet, the next-state function a set of productions, and the initial array an axiom. Just as one can obtain new cellular arrays from previous ones by using the next-state function, so in language-theory one obtains new strings of symbols by the use of production rules. But the difference between our constructs and grammars lies in the fact that (1) in our case all cells in the array must simultaneously be transformed at each step; in other words, all symbols in the string must simultaneously undergo substitution, according to the production rules; and (2) in our constructs no distinction is made between terminal and non-terminal symbols as is done in the case of grammars.

In formal language theory a language is defined as the set of all terminal strings, generated from the axiom by the production rules (terminal strings are string composed only of terminal symbols). Analogously, we may define developmental languages (or L-languages) as the set of all strings generated from the axiom by simultaneous application of production rules to all symbols in each string (no restriction needed concerning terminal or non-terminal symbols).

An L-system may either be deterministic or non-deterministic, depending on whether for every state symbol there is exactly one next-state transition rule (production) or not. If a system is deterministic, then there will be a single sequence of arrays (strings) generated by the rules beginning with the initial array (axiom); in non-deterministic systems many such sequences may be produced, possibly diverging and converging at various points.

Each such sequence of strings is supposed to correspond to the entire life of an organism, each string being a momentary description of it. A deterministic system permits only one such developmental sequence to occur, while a non-deterministic one yields many possible life histories.

A further division of L-systems is that into propagating and non-propagating ones, meaning thereby systems, respectively, without production rules resulting in cell deaths, and with such rules. Such rules correspond to erasing in language theory, i.e. to productions which erase particular symbols. The presence of erasing rules in grammars is well known to give rise to much more complex languages. Similarly, preprogrammed cell death has been recognized by developmental biologists (e.g. Saunders, 1966) to be an important mechanism in many morphogenetic processes.

A third, and equally important distinguishing notion for L-systems is the size of the neighbourhood of the cells. One can namely consider systems in which each cell acts independently of all other cells, i.e., systems with context-free productions, and others in which each cell receives inputs from certain number of its neighbours on either side in the filament, in other words, systems having productions with varying lengths of context. In general one may speak of L-systems with <u>k</u> left and <u>l</u> right inputs, meaning inputs received by each cell from <u>k</u> left and <u>l</u> right neighbour cells. The biological interpretation of such contexts was already given above, i.e., the transport of metabolites or other biologically active molecules from cell to cell. If one wishes to allow the transport to take place in one direction only, as in the case of auxin being transported from apex to base in stems of higher plants, then one needs only one-sided (undirectional) context. The higher the number of neighbours contributing to the context of a cell in one direction; the faster is the transport of the active substance along the filament (relative to the length of the time steps).

4. The simplest of the L-systems in terms of context are the ones without any interaction among the cells, called OL-systems, and these have been the subject of most work so far. Some of the results obtained as well as formal definitions of OL-systems and languages are given in the Abstracts of this Congress by Doucet (1971) and van Leeuwen (1971 b). Two simple examples for OL-languages are: (1) the set of strings, consisting of the letter a, of lengths 2^n, for all non-negative integers n. This language is generated by the single production a→aa, from axiom a. (2) The set of strings, for all integers $n \geq 1$, consisting of n-times the symbol a, followed by the symbol c or nothing, followed by n-times the symbol b. This language is produced by the productions:

a→a, b→b, c→acb, c→empty string; from axiom acb.

It is interesting to note that the language consisting of n-times a followed by n-times b symbols, a well-known context-free language, cannot be generated by OL-systems. It can be generated by L-systems with one-sided inputs.

Various theorems (due to Rozenberg and Doucet, 1971; Rozenberg, 1970, 1971b; van Leeuwen, 1971a; van Dalen, 1971; Lindenmayer, 1971; Doucet, 1972; Herman, 1972a; Salomaa, manuscript) deal with the relationship of OL-languages to the languages of the Chomsky hierarchy, of which the main classes are in order of increasing complexity: regular, context-free, context-sensitive, recursive, and recursively enumerable languages (see e.g. Aho and Ullman, 1968).

It can be proven that all OL-languages are context-sensitive. In fact, the set of OL-languages intersects the set of regular, context-free and context-sensitive languages, but does not exhaust any of them (thus, there are regular languages which are not OL-languages). The usual closure properties (under set-theoretical union, intersection, etc.) do not hold for OL-languages. OL-systems which have identy productions for all symbols in their alphabet (thus for every symbol a, one has a production a→a) produce context-free languages only.

Among the above-mentioned formal results perhaps only the last one has biological significance. It shows, namely, that the fact that certain OL-systems are able to produce non-context-free languages (for instance the language simulating the development of the red alga Callithamnion roseum, Lindenmayer, 1971) is due primarily to the synchronous cell divisions and changes of state imposed by the production rules of these systems. As soon as the synchrony requirement is relaxed by adding identity production rules for all symbols in these systems, the resulting language becomes context-free.

Another biologically interesting theorem was found by van Leeuwen (1971), stating that for every OL-system G, such that every symbol in its alphabet either leads to the empty string or to a cycle but not both, there exists a propagating OL-system G^1, and a non-erasing homomorphism h such that the language generated by G is identical to the h transform of the language generated by G^1 (with the addition of the empty string to the latter, should the empty string be in the language of G).

For example, the language (the set of all strings) consisting of ab repeated all possible finite numbers of times can be generated by an OL-system, but cannot be directly generated by a propagating OL-system. A non-propagating OL-system generating this language is the one with the productions: a→ab, a→abab, b→empty string; with the axiom: ab. The same language can be obtained by the use of a non-erasing homomorphism h, such that h(a) = ab, from a language produced by a propagating OL-system, namely the one with productions: a→a, and a→aa, and with axiom: a. Thus developmental patterns which are produced by systems without cellular interactions and for which preprogrammed cell death appears to be necessary, may also be produced by a system without cell death, provided that a biologically interpretable homomorphism can be found.

Still another biologically promising aspect of OL-systems has been the study of "locally catenative" systems (Rozenberg and Lindenmayer, 1972). In this case we are concerned with propagating, deterministic OL-systems which give rise to development such that certain previous developmental stages appear as part of the whole organism at a later time. One frequently sees such cases in the development of compound leaves and branching structures. Formally, we are dealing here with sequences of strings such that (after an initial period) each string consists of the concatenation of strings which were previously encountered in the sequence. It was possible to find a certain property (called dependence) of the set of productions of propagating, deterministic OL-systems which entails the generation of such locally catenative sequences. This property has to do with the existence of certain types of cycles among the states of the system. On the other hand, given any locally catenative formula, one can always find a propagating deterministic OL-system which gives rise to a sequence satisfying that formula. Thus the observation of a locally catenative developmental pattern in nature ensures the existence of a propagating deterministic system without cellular interactions which can give rise to such a pattern. Of course, the fact that such a no-interaction system could exist does not mean that in the actual case no interactions are taking place, since every pattern which can be produced without interactions can also be produced by a system with interactions. Nevertheless, a biologist might find it useful to know that such seemingly very complex patterns can arise by development without cellular interactions and without cell death, and thus he does not need to search for interactions or occurrences of cell death to explain the underlying developmental mechanisms.

From a biological point of view an important result would be an explicit characterization of OL-languages or OL-sequences. Algebraic characterizations are available at present among the Chomsky languages for the class of regular languages only (the class which can be recognized by finite automata or by the Mc-Culloch - Pitts nerve-nets). Partial characterizations are available for the set of context-free languages, in the sense, that given a context-free language one knows certain properties it must have, but given an arbitrary language with those properties it does not follow that it is context-free. Once even a partial characterization (or algorithmic procedure) would be available for the recognition of OL-languages, one would be able to tell from the developmental descriptions of an organism or of an organ whether it is necessary that cellular interactions occur in the course of its development. This is so, because the partial characterization would provide us with a formal property for which the developmental history of the organism could be tested (just as mentioned above, in the case of the locally catenative property for certain sequences), and if the organism does not fulfill the required property, then it would follow that its development could not have taken plaoe in the absence of cellular interactions.

At the present time characterization is available only for OL-languages over one-letter alphabets (Rozenberg and Doucet, 1971; van Leeuwen, Rozenberg and Herman, in preparation), but further work is in progress on characterizations of deterministic OL-languages.

An extension of OL-systems has been proposed by Rozenberg (197~~1~~2), in which the productions are organized into tables, each table containing at least one production for each symbol, and requiring that in any one string only productions of a single table may be used.

These kinds of developmental systems, which still have no interactions among cells, but in which the entire organism may be subjected to sudden changes of programs, are particularly suited to simulate developmental processes controlled by some critical value of an environmental parameter. We may think of the sudden switch from vegetative to flowering condition in higher plants as the length of daylight (or length of night) exceeds a certain threshold. A simple table OL-model for such a change in a filamentous fungus from vegetative to reproductive state under the control of light has been presented before (Surapipith and Lindenmayer, 1969). Although more powerful than regular OL-systems, table OL-systems were shown to be properly included in the set of context-sensitive languages.

5. As to L-systems with inputs, the first point to be mentioned is that they exhibit a dramatic rise in complexity. The results obtained by Herman (1969), van Dalen (1971) and Rozenberg (1971 d,e) show that L-systems with ~~on~~ inputs from single cells from one side only (called 1L-systems) can generate nonrecursive languages, i.e., languages of the highest complexity class in the Chomsky hierarchy. With proper coding 1L-systems can simulate any Turing machines.

Nevertheless, if one considers the languages which can be directly generated by 1L-systems (i.e., without any coding or canonical extensions), then it turns out that there are many regular languages which cannot be generated by them. Thus, the set of 1L-languages intersects all the major Chomsky-classes of languages, but does not include any of them. The set of 1L-languages includes, of course, the set of OL-languages, and it also includes the set of all finite languages.

Beginning with OL- and 1L- languages, one can build an entire hierarchy of L-languages, according to the number of inputs coming to a cell from either side, each set of L-languages with a higher total number of inputs (from both sides) containing the sets with lower total numbers of inputs. But no matter how high numbers <u>k</u> of left inputs and <u>1</u> of right inputs we choose, there always remain certain regular languages which cannot be generated by a (<u>k</u> + <u>1</u>) L-system.

The generation of finite languages is of particular biological interest, since it corresponds to development of organisms or organs with a final (adult) stage. This is what is called determinate growth in plants, of which leaf growth is a good example. Among animals the occurrence of an adult stage with a final size and form is widespread. It is important, therefore, to ask what kind of developmental programs would be necassary to yield such terminating growth and development, i.e., a sequence in which strings would eventually occur repeatedly. Curiously enough, while finite languages are considered to be of the lowest complexity type in formal language theory, the production of manyof these languages by propagating L-systems requires a high degree of interactions (large contexts). An attempt has been made to find general conditions for the production of symmetric terminating growth patterns with reference to the development of leaf marginal meristems (Lindenmayer, paper in preparation), and it was shown that such a pattern can be generated by a propagating L-system with two sided inputs which is completely apolar. (In a polar system the productions distinguish one side from the other either by giving rise to different states under mirror images of inputs, or by giving rise to two cells in unequal states under any combination of inputs. An apolar system is one which is not polar.)

The most interesting aspect of developmental systems with cell interactions is their ability to simulate regeneration. To regenerate a complete organism of the original size after amputating a part at a previous stage, or to reorganize a cut-off part to form a complete organism reduced in size, are processes which definitely require interactions among cells. An abstract statement of the latter case is the "French flag problem" proposed by Wolpert (1969). Systems with two-sided inputs, capable of accomplishing this kind of regeneration have been constructed by us as well as by Arbib (1969a) and Herman (1971, 1972b). The latter investigator has even given an apolar L-system which exhibits this kind of regeneration (this work is described elsewhere in these Proceedings). A general theory of developmental systems with regeneration properties is, however, lacking.

In the previous discussions (except for that of table OL-systems) it was always assumed that the environment remained constant during the course of development, and we asked what kinds of patterns can be produced by what class of systems. However, if the environment does not remain constant but fluctuates in some unknown way, then one is faced by an entirely different sort of construction and classification problem. Herman (1970) has begun to work on this topic, as well as on the problem of constructing L-systems to fit incomplete developmental data (see also Feliciangeli and Herman, 1972).

6. There have been other developmental models based on cellular automata published recently, which differed from the ones discussed here either (1) by being more than one-dimensional, or (2) by using continuous variables (concentration, age, distance) as well as discrete variables, or by both respects (1) and (2).

For instance, Baker and Herman (1972) studied the distribution of heterocysts along filaments of blue-green algae by a one-dimensional model in which the concentration of an inhibitor (which can diffuse from cell to cell) and the age of the cell determined cell division and the transition from vegetative to heterocyst condition.

Another kind of process which can be considered in a somewhat extended one-dimensional framework is the development of branching structures. Models giving rise to branching patterns and based on L-systems have been studied by Hesper and Hogeweg (1971,1972); and probabilistic cellular models for the production of branching patterns were given by Cohen (1967), who also considered the interaction of branches in terms of exhaustion of nutrients in their surroundings.

Cellular-automata models for two - or three dimensional development have also been investigated, such as the model of Raven (1968), and Raven and Bezem (1971), for the early embryonic development of the pond snail Limnaea stagnalis, in which the sizes and positions of the cells are algorithmically computed after each cleavage; or the cellular model for avian limb development by Ede and Law (1969), in which cell movements, in addition to cell divisions, are programmed in order to simulate in two dimensions the morphogenesis of limb initials; or a cellular model for the origin of regularly spaced leaf primordia (phyllotaxis) on a cylinder surface, controlled by the diffusion of an inhibitor (Lindenmayer and Veen, unpublished manuscript).

There is, at the present time, no general theoretical framework within which we could handle the various cellular-automata models, some of which are of more than one dimension, and some of which use continuous parameters as well. Clearly, the morphogenetic role of spatially directed cell divisions, cell enlargement, and cell movements can only be investigated if such models are further developed.

At the present time there is also a lack of clearly elaborated connection between the biochemical and physiological mechanisms of development which we mentioned earlier and the automata models described afterwards. Although we maintained that in principle such a connection is possible to find, detailed interpretations have not yet been attempted, and their testable consequences have not been obtained.

These shortcomings of cellular-automata theories of development are both serious. But the results obtained so far could provide certain automata - theoretical and formal linguistic interpretations for a number of fundamental concepts of developmental biology, such as differentiation, induction, regeneration, cellular interactions, polarity, and the roles of synchrony, cell death, and environment in development. There clearly remains a great deal more of the theoretical framework to be constructed before we can speak of a "theory of development".

Footnotes

1) I would like to add here that W.Ross Ashby (1956) was among the first to enunciate clearly the method of construction for interacting automata and I owe his book a debt for learning the basic principle of this method. I am also indepted to John R. Gregg for his having introduced me to automata theory in the first place.

2) Epigenesis is meant to include both morphogenesis and differentiation, for further explanation see e.g. Waddington, 1970.

3) Walter R.Stahl (1966) has introduced the concept of "gene-enzyme automata" in his computer model of a self-reproducing cell, and I would like to acknowledge the influence of his paper in construeing here genes and enzymes as production rules. I had many interesting discussions with him, and I felt his early death as a great loss. A theoretical biologist of very broad scope, he was also among the first to be concerned with computational complexities of cells and organisms, as compared for instance with Turing machines.

REFERENCES

Aho, A.V. and Ullman, J.D., 1968, The theory of languages, Math. Systems Theory, Vol.2, pp.97-125.

Arbib, M.A., 1969 a, Self-reproducing automata - Some Implications for theoretical biology, in: Towards a Theoretical Biology, Vol.2, ed.C.H.Waddington (Edinburgh Univ.Press), pp.204-226.

Arbib, M.A., 1969 b, Theories of Abstract Automata, (Prentice-Hall, Englewood Cliffs).

Ashby, W.R., 1956, An Introduction to Cybernetics, (Chapman and Hall, London).

Baker, R. and Herman, G.T., 1972, Simulation of organisms using a developmental model, Int.J.Biomed.Computing (submitted for publication).

Burks, A.W., editor, 1966, Theory of Self-Reproducing Automata, by John von Neumann (Univ. of Illinois Press, Urbana).

Burks, A.W., editor, 1968, Essays on Cellular Automata (Univ. of Illinois Press, Urbana).

Codd, E.F., 1968, Cellular Automata (Academic Press, New York).

Cohen, D., 1967, Computer simulation of biological pattern generation processes, Nature, vol.216, pp.246-248

Dalen, D.van, 1971, A note on some systems of Lindenmayer, Math. Systems Theory, vol.5, pp.128-140.

Doucet, P.G., 1971, Some results on OL-languages, Abstracts of IVth Int.Congr.Logic, Methodol., Philos.of Sci., Bucharest, pp.87-88.

Doucet, P.G., 1972, On the membership question in some Lindenmayer systems, Indagationes Mathematicae (in press).

Ede, D.A. and Law, J.T., 1969, Computer simulation of vertebrate limb morphogenesis, Nature, vol,221, pp.244-248

Feliciangeli, H. and Herman, G.T., 1972, Algorithms for producing grammars from sample derivations: A common problem of formal language theory and developmental biology, J.Computer and Systems Sci.(submitted for publication).

Goodwin, B.C., 1970, Biological stability, in: Towards a Theoretical Biology, vol.3, ed.C.H.Waddington(Edingburgh Univ.Press), pp. 1-17.

Herman, G.T., 1969, The computing ability of a developmental model for filamentous organisms, J.Theoret.Biol., vol.25, pp.421-435.

Herman, G.T., 1970, The role of environment in developmental models, J.Theor.Biol.,vol.29, pp.329-341.

Herman, G.T., 1971, Models of cellular interactions in development without polarity in individual cells, I.General description and the problem of universal computing ability, Int.J.Systems Sci., vol 2, pp.271-289.

herman, G.T. 1972 a, Closure properties of some families of languages associated with biological systems, Information and Control (Submitted for publication).

Herman, G.T., 1972 b, Models of cellular interactions in development without polarity of individual cells, II.Problems of synchronization and regulation, Int.J.Systems Sci., (in press).

Hesper, B. and Hogeweg, P., 1972, Generation of branching patterns, J.Theoret. Biol. (submitted for publication).

Hogeweg, P. and Hesper, B. 1971, Pattern recognition in biology: A methodological analysis using computer experimentation, Abstracts of IVth Int.Congr.Logic, Methodol., Philos.of Sci., Bucharest, pp.279-280.

Kleene, S.C., 1956, Representation of events in nerve nets and finite automata, in: Automata Studies, No.34, Annals of Math. Studies, eds.C.E.Shannon and J.McCarthy, (Princeton Univ. Press), pp.1-41.

Leeuwen, J.van, 1971 b, Canonical restrictions of Lindenmayer-languages, Abstracts IVth Int.Congr.Logic,Methodol.,Philos. of Sci.,Bucharest, p.284.

Lindenmayer, A., 1968, Mathematical models for cellular interactions in development, Parts I and II, J.Theoret.Biol, vol.18, pp. 280-315.

Lindenmayer, A., 1971, Developmental systems without cellular interactions, their languages and grammars, J.Theoret.Biol., vol. 30, pp.455-484.

McCulloch, W.S. and Pitts, W., 1943, A logical calculus of the ideas immanent in nervous activity, Bull.Math.Biophys., vol.5, pp.115-133.

Raven, C.P., 1968, A model of pre-programmed differentiation of the larval head region of Limnaea stagnalis, Acta Biotheoretica, vol.18, pp.316-329.

Raven, C.P. and Bezem, J.J., 1971, Computer simulation of embryonic development, Parts I and II, Proc.Kon.Ned.Akad.Wetensch., Series C., vol.74, pp.209-233.

Rozenberg, G., 1970, Some results on OL-Languages, Parts I and II, Elektr.Rekencentrum Utrecht, Publ.Nos.93 and 95, 30pp.

Rozenberg, G., 1971 a, T-O-L systems and languages. On O-L systems with restricted use of productions. Elektr. Rekencentrum Utrecht, Publ.Nos.103 and 116, 110 pp.

Rozenberg, G., 1971 b, On some properties of propagating D-O-L systems. Elektr.Rekencentrum Utrecht, Publ.No.106, 30 pp.

Rozenberg, G., 1971 c, L-systems with interactions. Propagating L-systems with interactions. Elektr.Rekencentrum Utrecht, Publ.Nos. 120 and 121, 75 pp.

Rozenberg, G. and Doucet, P.G., 1971, On OL languages. Information and Control. vol.19, pp.302-318.

Rozenberg, G. and Lindenmayer, A., 1972, Developmental systems with locally catenative formulas, Acta Informatica (submitted for publication).

Salomaa, A., (manuscript) Formal Languages (Academic Press, New York).

Saunders, J.W., Jr., 1966, Death in embryonic systems, Science, vol.154 pp.604-612.

Stahl, W.R., 1966, A model of self-reproduction based on string processing finite automata, in: Natural Automata and Useful Simulations, ed.H.H.Pattee et al. (Spartan Books, Washington), pp. 43-72.

Surapipith, V. and Lindenmayer, A., 1969, Thioguanine-dependent light sensitivity of perithecial initiation in Sordaria fimicola J.Gen.Microbiol., vol 57, pp.227-237.

Vitányi, P.M.B., 1972, Sexually reproducing cellular automata, Math. Biosciences (submitted for publication).

Waddington, C.H., 1968, Theoretical biology and molecular biology, in: Towards a Theoretical Biology, vol. 1, ed.C.H.Waddington, (Edinburgh Univ.Press), pp.103-108.

Waddington, C.H., 1970, Concepts and theories of growth, development, differentiation and morphogenesis, in: Towards a Theoretical Biology, vol.3, ed.C.H.Waddington (Edinburgh Univ. Press), pp.177-197.

Wolpert, L., 1969, Positional information and the spatial pattern of cellular differentiation, J.Theoret.Biol., vol.25, pp.1-48.

INTEGRAL SEQUENTIAL WORD FUNCTIONS AND GROWTH EQUIVALENCE OF LINDENMAYER SYSTEMS

by

Azaria Paz & Arto Salomaa

1. INTRODUCTION

Lindenmayer systems (also called L-systems, Lindenmayer models or developmental languages) have been the object of extensive study during the past two years. The systems were introduced in connection with a theory proposed to model the development of filamentous organisms. The stages of development are represented by words corresponding to one-dimensional arrays of cells (filaments). The developmental instructions are modelled by ordinary rewriting rules or productions. These productions are applied simultaneously to all letters to reflect the simultaneity of the growth in the organism. This parallel rewriting is the main difference between Lindenmayer systems and ordinary generative grammars. There are many types of Lindenmayer systems. One distinction results from the fact that the various parts of the developing organism may or may not be in communication with each other. Different types of systems will be defined in the sequel at appropriate places. For more background material and motivation, the reader is referred to Rozenberg and Doucet (1971), Rozenberg and Lindenmayer (1971) and Salomaa (1973), as well as to the items given in their bibliographies.

A particularly interesting aspect in the study of Lindenmayer systems is the theory of growth functions. The basic paper in this field is by Szilard (1972). In the theory of growth functions only the lengths of the words matter, no attention is paid to the words themselves. This implies that many problems become solvable whose solution is unknown for L-systems in general. Also hierarchies of language families may reduce to one family of growth functions.

The basic observation behind this paper is that growth functions of certain Lindenmayer systems fit in the framework of the theory of integral sequential word functions. Functions resembling the latter have been studied extensively in the past, cf. Paz (1971), pp. 116-144, in connection with probabilistic automata. Consequently, our subsequent results might be of interest to both people working with word functions and to people interested in Lindenmayer systems. The former may read only Part 2 of this paper, although some definitions get their motivation in Part 3. (One of them is the definition of the vector η.) On the other hand, people interested in Lindenmayer systems may read Part 3 only, although they miss many of the proofs.

Basic notions concerning integral sequential word functions are introduced in Sections 2a and 2b. Some theorems, based on earlier results, are also mentioned. Section 2c gives preliminary results concerning the reduction problem which gives a solution to the minimization problem and is directly applicable to growth functions. Using some results previously known, the reduction theorem is then applied to solve the problem of realizing a given function as an integral sequential word function (Theorem 17). Some other representability problems are also considered in Section 2e. Section 2f deals with closure properties, and Section 2g with the single letter case which, in fact, corresponds to DOL-systems.

Section 3a deals with the growth functions of DOL-systems. Algorithms are given for the solution of the following problems: growth equivalence, finding all growth equivalent axioms and cell minimization. It is also shown that, for any DOL-system S and integer k, there is only a finite number of DOL-systems growth equivalent to S and having k letters in their alphabet. We also study the problem of realizing a given function as a growth function, as well as problems concerning malignant growth. The following Section 3b deals with growth functions of context-dependent Lindenmayer systems. Examples are given of such growth functions which are not DOL growth functions. Also a result concerning the 'saving of cells' in the transition from informationless to context-dependent systems is established. The last Section 3c deals with OL-systems and deterministic Lindenmayer systems with tables. For obvious reasons, the growth functions in these cases become growth relations. It is shown that the family of growth relations of DTOL-systems, properly includes the family of growth relations of OL-systems although mutual overlap holds between the corresponding language families.

2. INTEGRAL SEQUENTIAL WORD FUNCTIONS

In this section we shall study integral sequential word functions; i.e., functions $f : \Sigma^* \rightarrow N$ (Σ^* is the set of all words over a finite alphabet and N is the set of nonnegative integers) induced by a sequential integral system. The specific functions to be considered here can be used for investigating growth functions of various types of OL-systems as explained before. On the other hand, similar functions of a more general character have been studied elsewhere [see Paz (1971)] so that many theorems valid in the general case, carry over to this specific model. Whenever a proof to a theorem stated here is similar to an existing proof in the literature, we shall skip the proof here and refer the interested reader to the literature. We shall discuss here, in detail, only those aspects of the integral word functions which are pertinent to their use as a growth function and which exhibit a specific aspect different from the general case and resulting from the specific integral assumption.

a. Definitions and Notations

All the vectors and matrices considered in this section are assumed to have only nonnegative integral entries unless otherwise stated. A state vector is a vector having exactly one nonzero value. The notation η stands for a column vector of due dimension with all its entries equal to 1. The notation π will be used for row vectors. Superscripts for vectors will be used for distinguishing between them and subscripts will be used to denote a specific entry in a vector. Σ denotes a finite alphabet, Σ^* the set of all words over Σ; λ the empty word; and σ, an element of Σ.

Definition 1: An n-state integral sequential system (IS) over a finite alphabet Σ is a triple $\mathcal{A}_\pi = [\pi, \{A(\sigma)\}_{\sigma\in\Sigma}, \eta]$ where π is an n-dimensional "initial" row vector and the $A(\sigma)$ are n-dimensional matrices. When using the notation $\mathcal{A}$ instead of $\mathcal{A}_\pi$, we shall assume that the initial vector π is not yet specified, while $\mathcal{A}_{\pi^1}$ and $\mathcal{A}_{\pi^2}$ will denote two IS which differ only in their initial vector π.

Definition 2: An integral sequential word function (ISF) is a function $f^{\mathcal{A}\pi}: \Sigma^* \to N$ (the superscript will be omitted when context is clear) induced by the IS $\mathcal{A}_\pi$ and defined as $f(x) = \pi A(x)\eta$ where $x = \sigma_1 \ldots \sigma_k \in \Sigma^*$ and $A(x) = A(\sigma_1)\ldots A(\sigma_k)$ by definition. Also by definition $f(\lambda) = \pi\,\eta$.

Definition 3: Two initial vectors π^1 and π^2 for a given IS $\mathcal{A}$ are equivalent if $f^{\mathcal{A}\pi^1}(x) = f^{\mathcal{A}\pi^2}(x)$ for all $x \in \Sigma^*$.

Definition 4: Two IS $\mathcal{A}^1_{\pi^1}$ and $\mathcal{A}^2_{\pi^2}$ over the same alphabet Σ, are equivalent if

$$f^{\mathcal{A}^1_{\pi^1}}(x) = f^{\mathcal{A}^2_{\pi^2}}(x)$$

for all $x \in \Sigma^*$.

Definition 5: Two IS $\mathcal{A}^1$ and $\mathcal{A}^2$ are state equivalent if for any initial state vector π^1 for the first IS one can find an initial state vector π^2 for the second such that $\mathcal{A}^1_{\pi^1}$ is equivalent to $\mathcal{A}^2_{\pi^2}$ (notation: $\mathcal{A}^1_{\pi^1} \simeq \mathcal{A}^2_{\pi^2}$), and vice versa.

Remark: One verifies easily that if $\mathcal{A}^1$ is state equivalent to $\mathcal{A}^2$ then for any initial vector π^1 (not necessarily a state vector) there is an initial vector π^2 for $\mathcal{A}^2$ such that $\mathcal{A}^1_{\pi^1} \cong \mathcal{A}^2_{\pi^2}$ and vice versa.

Given an IS $\mathcal{A}_\pi$, $K^{\mathcal{A}}$ and $G^{\mathcal{A}\pi}$ denote the ordered infinite sets of column and row vectors respectively:

$$K^{\mathcal{A}} = [\eta(\lambda),\eta(x_1)\ldots,\eta(x_k)..]; \quad G^{\mathcal{A}\pi} = \begin{bmatrix} \pi(\lambda) \\ \pi(x_1) \\ \pi(x_2) \\ \cdot \\ \cdot \\ \cdot \\ \pi(x_k) \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

where by definition $\eta(x) = A(x)\eta$; $\eta(\lambda) = \eta$; $\pi(x) = \pi A(x)$; $\pi(\lambda) = \pi$; and x_1 x_2 ... is a fixed lexicographic order on the words in Σ^*.

Let $K(m)$, $G^{\pi}(m)$ denote the ordered subsets of $K^{\mathcal{A}}$ and $G^{\mathcal{A}\pi}$ respectively such that $\eta(x) \in K^{\mathcal{A}}(m)$ $(\pi(x) \in G^{\mathcal{A}\pi})$ for all x such that $|x| \leqslant m$ ($|x|$ denotes the length of x).

b. Some basic Theorems

Theorem 1 : For a given n-state IS $\mathcal{A}_\pi$ there exists an effective algorithm for finding a set of linearly independent vectors in $K^{\mathcal{A}}(n-1)$ such that all vectors in $K^{\mathcal{A}}$ depend linearly on them, and the same is true for $G^{\mathcal{A}\pi}(n-1)$.

For proof see Paz (1971) p. 19.

Let $\eta^1 \dots \eta^m$ and $\pi^1 \dots \pi^k$ be the two sets of vectors having the following properties:

1. $\eta^1 = \eta$; $\pi^1 = \pi$
2. $\eta^1, \dots, \eta^m$ and $\pi^1, \dots, \pi^k$ are the first vectors in $K^{\mathcal{A}}$ and $G^{\mathcal{A}\pi}$ respectively accroding to the preassigned fixed order – which are linearly independent and span their whole sets.

The matrices $H^{\mathcal{A}}$ and $L^{\mathcal{A}\pi}$ are defined as:

$$H^{\mathcal{A}} = [\eta^1 \dots \eta^m], \quad L^{\mathcal{A}\pi} = \begin{bmatrix} \pi^1 \\ \cdot \\ \cdot \\ \cdot \\ \pi^k \end{bmatrix}$$

It is clear that the ranks of the above defined matrices are $\leqslant n$.

Theorem 2: Two initial vectors for a given IS are equivalent if and only if

$$\pi^1 H^{\mathcal{A}} = \pi^2 H^{\mathcal{A}} \qquad (1)$$

For proof see Paz (1971) p. 22.

Remark: It follows from the above theorem that for a given IS $\mathcal{A}_\pi$ there are only finitely many other initial vectors π^1 equivalent to π. This follows from the fact that any such vector must have nonnegative integral entries with their sum equal to the sum of entries of π. On the other hand, all the vectors π^1 equivalent to π can be found by using the above equation (1).

Let $\mathcal{A}_\pi$ be an IS and let $\xi^i(\sigma)$ be the i-th row (assumed here to be a nonzero row) in a matrix $A(\sigma)$ for some σ. Let ξ' be an integral vector such that $\xi^i(\sigma)H^{\mathcal{A}} = \xi' H^{\mathcal{A}}$ and let $\mathcal{A}'_{\pi'}$ be an IS derived from $\mathcal{A}_\pi$ by replacing the row $\xi^i(\sigma)$ of $A(\sigma)$ with the row ξ' and replacing π with an equivalent initial integral vector π' (with respect to $\mathcal{A}$). We have the following:

Theorem 3: The IS $\mathcal{A}_\pi$ and $\mathcal{A}'_{\pi'}$ as above are equivalent.
For proof see Paz (1971) p. 23.

c. Reduction Theorems

Definition: A state i of an IS $\mathcal{A}_\pi$ is accessible if there is a word $x \in \Sigma^*$ such that $\pi(x)_i > 0$ (where $\pi(x)_i$ denotes the i-th entry of $\pi(x)$).

Theorem 4: If a state i of an n-state IS is accessible then it is accessible by a word of length $\leqslant n$. The set of accessible states can effectively be found.
Proof: Trivial.

Theorem 5: If there is a state i of an n-state IS $\mathcal{A}_\pi$ which is not accessible then the given IS can be reduced to an n-1-state equivalent IS.
Proof: Delete the i-th entry in π (which must be zero); delete the i-th columns and rows in the matrices $A(\sigma)$ (if j is an accessible state then the j-th row in $A(\sigma)$ must have a zero entry in its i-th column) and reduce η to an (n-1) dimensional vector.

Theorem 6: Let $\mathcal{A}_\pi$ be an n-state IS such that its $H^{\mathcal{A}}$ matrix has two equal rows, then $\mathcal{A}_\pi$ can be reduced to an (n-1)-state equivalent IS.

Proof: See Paz (1971) p. 23.

Example:

Consider the following IS $\mathcal{A}_\pi$:

$\pi = [1,1,1,1]$ $\Sigma = \{\sigma_1, \sigma_2\}$

$$A(\sigma_1) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 \\ 1 & 0 & 0 & 1 \end{bmatrix} \qquad A(\sigma_2) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix} \qquad \eta = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

then

$$H^{\mathcal{A}} = \begin{bmatrix} 1 & 2 \\ 1 & 0 \\ 1 & 4 \\ 1 & 2 \end{bmatrix}$$

Thus the first and fourth row of H are equal and therefore one can find a 3-state equivalent IS $\mathcal{A}'_{\pi'}$

$\pi' = (2 \ 1 \ 1)$ $\Sigma = \{\sigma_1, \sigma_2\}$

$$A'(\sigma_1) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 0 \end{bmatrix} \qquad A'(\sigma_2) = \begin{bmatrix} 2 & 0 & 0 \\ 4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \eta' = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

with

$$H^{\mathcal{A}'} = \begin{bmatrix} 1 & 2 \\ 1 & 0 \\ 1 & 4 \end{bmatrix}$$

Although it is clear that no further reduction is possible using the previous theorems 5, 6 still the above IS is equivalent to the following $\mathcal{A}''_{\pi''}$

$\pi'' = (0\ \ 2\ \ 2)$

$$A''(\sigma_1) = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 2 & 2 \end{bmatrix} \qquad A''(\sigma_2) = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 0 \end{bmatrix} \qquad \eta'' = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

that $\mathcal{A}'_{\pi'} \cong \mathcal{A}''_{\pi''}$ follows from Theorem 3. The first state is not accessible in $\mathcal{A}''_{\pi''}$ and therefore, a 2-state equivalent IS $\bar{\mathcal{A}}_{\bar{\pi}}$ can be found:

$$\bar{\pi} = [2, 2] \qquad \bar{A}(\sigma_1) = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix} \qquad \bar{A}(\sigma_2) = \begin{bmatrix} 2 & 2 \\ 0 & 0 \end{bmatrix} \qquad \bar{\eta} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Notice that the derivation of $\mathcal{A}''_{\pi''}$ from $\mathcal{A}'_{\pi'}$ was made possible by the fact that in $H'^{\mathcal{A}'}$ the first row was a convex combination of the other two rows. Such a condition is, however, not sufficient, and there are other conditions, deriving from the requirement that the resulting matrices and initial vector have only integral values, which must be considered. These considerations lead to the following problem: Given an n-state IS, give an algorithm which will decide whether there exists an equivalent IS with less than n-states and, if the answer is positive, will provide a procedure by which such an equivalent IS could be constructed.

d. General reduction problem

Let $\mathcal{A}_\pi$ be a given IS and consider the infinite ordered set of vectors $K^{\mathcal{A}}$ and the matrix $L^{\mathcal{A}_\pi}$ as defined in the previous section. Denote by $[K^{\mathcal{A}}]$, the infinite matrix whose i-th column is the i-th vector in $K^{\mathcal{A}}$. Define the infinite matrix $[K^{(\mathcal{A},\pi)}]$ as

$$[K^{(\mathcal{A},\pi)}] = L^{\mathcal{A}_\pi}[K^{\mathcal{A}}]$$

Theorem 7: Let $\mathcal{A}_\pi$ and $\mathcal{A}^*_{\pi^*}$ be two IS over the same Σ. $\mathcal{A}_\pi$ is equivalent to $\mathcal{A}^*_{\pi^*}$ if and only if there exists an integral non-negative matrix (i.e., with all its entries nonnegative integers) B^* such that its first row is π^* and

$$B^*[K^{\mathcal{A}^*}] = L^{\mathcal{A}_\pi}[K^{\mathcal{A}}] \tag{2}$$

Proof: Assume first that $\mathcal{A}_\pi \cong \mathcal{A}^*_{\pi^*}$. Then for any $x_1, x_2 \in \Sigma^*$, $\pi^*(x_1)\eta^*(x_2) = \pi(x_1)\eta(x_2)$. If the i-th row in $L^{\mathcal{A}_\pi}$ is $\pi(x_i)$ choose the i-th row in B^* to be $\pi^*(x_i)$. (The first row in $L^{\mathcal{A}_\pi}$ is π which will correspond to π^* in B^*.) This implies (2) for $\eta^*(x_2)$ and $\eta(x_2)$ are corresponding columns in $K^{\mathcal{A}^*}$ and $K^{\mathcal{A}}$ respectively.

Assume now that (2) holds for some matrix B^* with first row equal to π^*, then the first row of the equation (2) has the form

$$\pi^*[K^{\mathcal{A}^*}] = \pi[K^{\mathcal{A}}]$$

which implies that $\pi^*\eta^*(x) = \pi\eta(x)$ for all $x \in \Sigma^*$ so that the two IS are equivalent.

Let $\mathcal{A}_\pi$ be a given IS and consider the matrix $[K^{(\mathcal{A},\pi)}] = L^{\mathcal{A}_\pi}[K]^{\mathcal{A}}$ as above. It can be shown (see Paz (1971) p. 51) that it is possible to effectively construct a matrix, to be denoted $H^{(\mathcal{A},\pi)}$, such that

(1) The columns of $H^{(\mathcal{A},\pi)}$ are linearly independent vectors from the set $K^{(\mathcal{A},\pi)}$ and all other vectors in the set are a linear combination of them.

(2) The columns in $H^{(\mathcal{A},\pi)}$ are the first columns in $[K^{(\mathcal{A},\pi)}]$ satisfying the condition (1) above.

In fact, one can show that the columns of $H^{(\mathcal{A},\pi)}$ can be chosen to be vectors of the form $L^{\mathcal{A}_\pi}\eta(x)$ with $|x| \leq n-1$ (given that $\mathcal{A}_\pi$ is an n-state IS).

<u>Theorem 8</u>: Let $\mathcal{A}_\pi$ be an IS over an alphabet Σ, let m be the number of columns in its $H^{(\mathcal{A}_\pi)}$ matrix. No m^*-state IS $\mathcal{A}^*_{\pi^*}$ over the same alphabet Σ with $m^* < m$ can be equivalent to $\mathcal{A}_\pi$.

Proof: If $\mathcal{A}_\pi \cong \mathcal{A}^*_{\pi^*}$ then by (2) there exists a matrix B^* such that $B^*[K^{\mathcal{A}^*}] = L^{\mathcal{A}_\pi}[K^{\mathcal{A}}] = [K^{(\mathcal{A},\pi)}]$. But $H^{(\mathcal{A},\pi)}$ is a submatrix of

$[K^{(\mathcal{A},\pi)}]$ with m independent columns. Therefore, $[K^{(\mathcal{A},\pi)}]$ must have m independent rows which implies by (2) that $[K^{\mathcal{A}^*}]$ has at least that many rows.

Let $\mathcal{A}_\pi$ be a given IS and consider the following equations with $\Delta(\sigma)$ an unknown (not necessarily integral) matrix, for all $\sigma\in\Sigma$.

$$L^{\mathcal{A}_\pi}A(\sigma) = \Delta(\sigma)L^{\mathcal{A}_\pi} \qquad (3)$$

This equation has exactly one solution which can be found effectively. This follows from the fact that the rows of $L^{\mathcal{A}_\pi}$ are linearly independent while the rows of $L^{\mathcal{A}_\pi}A(\sigma)$ being in the set $G^{\mathcal{A}_\pi}$ depend on the rows of $L^{\mathcal{A}_\pi}$.

It follows that the equation

$$L^{\mathcal{A}_\pi}A(\sigma)H^{\mathcal{A}} = \Delta(\sigma)L^{\mathcal{A}_\pi}H^{\mathcal{A}} \qquad (4)$$

has at least one solution.

Let $\Delta(\sigma)$ be any solution of the equation (4). Then $\Delta(\sigma)$ satisfies also the following equation for all $x\in\Sigma^*$.

$$L^{\mathcal{A}_\pi}\eta(\sigma x) = L^{\mathcal{A}_\pi}A(\sigma)\eta(x) = \Delta(\sigma)L^{\mathcal{A}_\pi}\eta(x) \quad (5)$$

This follows from the fact that the vectors $\eta(x)$ are linear combinations of the columns of $H^{\mathcal{A}}$.

Let $\eta(\pi,x)$ denote the vector $L^{\mathcal{A}_\pi}\eta(x)$. Thus, $\eta(\pi,x)$ is the vector corresponding to the word x in the matrix $[K^{(\mathcal{A},\pi)}] = L^{\mathcal{A}_\pi}[K^{\mathcal{A}}]$. We have the following

Theorem 9: Given an IS $\mathcal{A}_\pi$, there are matrices $\Delta(\sigma)$, for each $\sigma\in\Sigma$, (not necessarily integral) such that

$$\Delta(\sigma)[K^{(\mathcal{A},\pi)}] = [\eta(\pi,\sigma),\eta(\pi,\sigma x_1),\eta(\pi,\sigma x_2)\ldots] \qquad (6)$$

where $\lambda, x_1, x_2, \ldots.$ is the fixed enumeration of the words in Σ^*.

Notice that while $\Delta(\sigma)$ may have nonintegral entries both $[K^{(\mathcal{A},\pi)}]$ and the infinite matrix on the right hand side of the equation (6) have only integral nonnegative values.

We are now able to prove the following

Theorem 10: Let $\mathcal{A}^*{}_{\pi^*}$ and $\mathcal{A}_\pi$ be two IS over the same alphabet Σ. $\mathcal{A}^*{}_{\pi^*}$ is equivalent to $\mathcal{A}_\pi$ if and only if there exists an integral nonnegative matrix B^* with first row equal to π^* such that the following equations hold true:

$$B^*\eta^* = \eta(\pi,\lambda),\ B^* A^*(\sigma)H^{\mathcal{A}^*} = \Delta(\sigma)B^* H^{\mathcal{A}^*} \qquad \text{all } \sigma\in\Sigma \tag{7}$$

where $\Delta(\sigma)$ are matrices as defined in (6).

Proof: We know already that the conditions of the Theorem are equivalent to the existence of a matrix B^* as required and satisfying the equation (2); i.e.,

$$B^*[K^{\mathcal{A}^*}] = [K^{(\mathcal{A},\pi)}] \tag{8}$$

We will show first that the condition (8) above is equivalent to the following

$$B^*\eta^* = \eta(\pi,\lambda) \text{ and } B^* A^*(\sigma)\eta^*(x) = \Delta(\sigma)B^*\eta^*(x), \quad \text{all } x\in\Sigma \tag{9}$$

with the same matrix B^*.

Clearly, $A^*(\sigma)\eta^*(x) = \eta^*(\sigma x)$ and by (8) $B^*\eta^*(\sigma x) = \eta(\pi,\sigma x)$. On the other hand, by (8), we have that $B^*\eta^*(x) = \eta(\pi,x)$ and it follows from (6) that $,\Delta(\sigma)\eta(\pi,x) = \eta(\pi,\sigma x)$. Thus, $(8)\Longrightarrow(9)$.

Assume now, that (9) holds true, then we prove by induction on the legnth of x that $B^*\eta^*(x) = \eta(\pi,x)$. For $x=\lambda$ (9) and (8) are identical.

For $x = \sigma x'$ we have by (9) and by the induction hypothesis that $B^*\eta^*(\sigma x') = B^* A^*(\sigma)\eta^*(x') = \Delta(\sigma)B^*\eta^*(x') = \Delta(\sigma)\eta(\pi, x')$.

This implies by (6) that $B^*\eta^*(\sigma x') = \eta(\pi, \sigma x')$ as required. Thus, $(9) \Longrightarrow (8)$.

We prove now that (9) is equivalent to (7) with same matrix B^*. That $(9) \Longrightarrow (7)$ is trivial for the columns in $H^{\mathcal{A}^*}$ are of the form $\eta^*(x)$ for some $x \in \Sigma^*$. The converse is also easy, for any column of the form $\eta^*(x)$ can be expressed as a linear combination of the columns of $H^{\mathcal{A}^*}$ by definition.

Corollary 11: Let $\mathcal{A}^*_{\pi^*}$ and $\mathcal{A}_\pi$ be two IS over the same alphabet Σ. Let $J^{\mathcal{A}_\pi}$ be a matrix whose columns are the columns in $[K^{(\mathcal{A},\pi)}]$ corresponding to the same words in Σ^* as the columns in $H^{\mathcal{A}^*}$ and in the same order. $\mathcal{A}^*_{\pi^*}$ is equivalent to $\mathcal{A}_\pi$ if and only if there exists a nonnegative integral matrix B^* whose first row equals π^* and such that

$$B^*\eta = \eta(\pi,\lambda) \text{ and } B^*H^{\mathcal{A}^*}(\sigma) = J^{\mathcal{A}_\pi}(\sigma), \quad \text{for all } \sigma \in \Sigma. \qquad (10)$$
$$(H^{\mathcal{A}^*}(\sigma) = A^*(\sigma)H^{\mathcal{A}^*} \text{ and } J^{\mathcal{A}_\pi}(\sigma) = \Delta(\sigma)J^{\mathcal{A}_\pi}).$$

Notice that the columns of $H^{\mathcal{A}^*}(\sigma)$ are columns in $[K^{\mathcal{A}^*}]$ and, therefore, all the entries in $H^{\mathcal{A}^*}(\sigma)$ are nonnegative and integral. Similarly, the columns in $J^{\mathcal{A}_\pi}(\sigma)$ are the columns in $[K^{\mathcal{A}_\pi}]$, corresponding ot the same words in Σ^* as the columns in $H^{\mathcal{A}^*}(\sigma)$ and therefore, the entries in $J^{\mathcal{A}_\pi}(\sigma)$ are alsononnegative and integral.

Corollary 12: Let $\mathcal{A}^*_{\pi^*}$ be an n-state IS and let $\mathcal{A}_\pi$ be another IS over the same alphabet Σ. Then $\mathcal{A}^*_{\pi^*}$ is equivalent to $\mathcal{A}_\pi$ if and only if there exists a nonnegative integral matrix B^* whose first row equals π^* and such that

$$B^*[K^{\mathcal{A}^*}(n)] = [K^{(\mathcal{A},\pi)}(n)] \qquad (11)$$

where $[K^{\mathcal{A}^*}(n)]$ is the matrix whose columns are the vectors $\eta^*(x)$

with $|x| \le n$ and similarly for $[K^{(\mathcal{A},\pi)}(n)]$.

Proof: $K^{\mathcal{A}^*}(n)$ includes the vector η and it follows from Theorem 1 that the columns of $H^{\mathcal{A}^*}(\sigma)$ are columns in $[K^{\mathcal{A}^*}(n)]$ so that (11) implies (10). On the other hand, we have as an immediate consequence of corollary 11, that (10) implies (11).

Remark: One can use now the above Corollary 12 and prove the decidability of the equivalence problem of two IS. We shall postpone, however, this problem and discuss it in a later section of this paper where an easier algorithm will be suggested for it.

We are now able to settle the minimization problem for IS.

Theorem 13: Given an n-state IS $\mathcal{A}_\pi$, there exists an effective algorithm which will construct another equivalent m-state IS $\mathcal{A}^*_{\pi^*}$ with $m < n$, if such an $\mathcal{A}^*_{\pi^*}$ exists, or will decide that no such $\mathcal{A}^*_{\pi^*}$ exists.

Proof: We shall exhibit an algorithm which will perform the required task. Each step of the algorithm will be followed by an explanation if necessary- We shall need the following notations:

Let $\xi^1, \xi^2, \dots, \xi^k$ be a set of n-dimensional vectors then $\mathcal{S}(\xi^1, \xi^2, \dots \xi^k)$ denotes the minimal hypersphere in n-space with center at origin including the point vectors $\xi^1, \dots, \xi^k$ in its interior or on its boundary. If $\mathcal{V}$ is a matrix whose rows are $\xi^1, \dots, \xi^k$ then, by definition, $\mathcal{S}(\mathcal{V}) = \mathcal{S}(\xi^1, \dots, \xi^k)$.

Algorithm for Theorem 13

Step 1: Given the IS $\mathcal{A}_\pi$, let t be the number of columns in the matrix $H^{(\mathcal{A},\pi)}$ and let n be the number if states of $\mathcal{A}_\pi$. Set $m = t$.

Step 2: If $m = n$ stop. There is no $\mathcal{A}^*_{\pi^*}$ with less than n states and equivalent to $\mathcal{A}_\pi$ (for $m = t$ this follows from Theorem 8). Else, go to the next step.

Step 3: Construct the matrix $[K^{(\mathcal{A},\pi)}(m)]$ and let p be the number of its rows. (It is clear that $t \le p \le n$.)

Step 4: Construct a matrix B^* with p rows and m columns such that:

a) All its entries are nonnegative integers.

b) The sum of its columns is equal to the column vector $\eta(\pi,\lambda)$.

c) The matrix B^* has not been used in a previous application of step 4 of the algorithm.

If no such B^* matrix can be found then set $m = m+1$ and go to step 2. Else, go to the next step.

Explanation: The matrix B^* as constructed in step 4 is intended to be the matrix satisfying the equation (10) in Corollary 11, which implies the conditions a) and b). The third condition c) is inserted here for the case where the algorithm will come back to step 4 after going through other steps. It is clear that there are only finitely many matrices B^* satisfying the conditions a) and b) for fixed m. Therefore, because of condition c), the algorithm will pass through step 4 only finitely many times before changing the parameter m.

Step 5: If the chosen matrix B^* has no column with all its entries zero entries, then go to the next step. Else, go to step 10.

Step 6: Construct a matrix $\mathcal{V}$ with m rows and same number of columns as the matrix $[K^{(\mathcal{A},\pi)}(m)]$, with all its entries nonnegative integers, with all its rows (when considered as point vectors) in the interior or on the boundary of $\mathcal{S}([K^{(\mathcal{A},\pi)}(m)])$, with first column equal to η, such that $\mathcal{V}$ has not been used in a previous application of step 6, and such that $\mathcal{V}$ satisfies the equation $B^* \mathcal{V} = [K^{(\mathcal{A},\pi)}(m)]$.

If no such $\mathcal{V}$ matrix can be found, then go to step 4. Elsego to the next step.

Explanation: The $\mathcal{V}$ matrix is intended to be the $[K^{\mathcal{A}^*}(m)]$ matrix satisfying the equation (11) in Corollary 12. According to that equation, the rows of $[K^{(\mathcal{A},\pi)}(m)]$ must be integral nonnegative combinations of the rows of $\mathcal{V}$ and every row of $\mathcal{V}$ must be used in the construction of some row of $[K^{(\mathcal{A},\pi)}(m)]$ (the matrix B^* has no all-zero columns by step 5). It is thus clear that no pointvector outside $\mathcal{S}([K^{(\mathcal{A},\pi)}(m)])$ can participate in the formation of the rows of $[K^{(\mathcal{A},\pi)}(m)]$. This implies that there are only finitely many matrices $\mathcal{V}$ satisfying the conditions in step 6 so that step 6 will be used only finitely many times before changing the matrix B^*.

Step 7: Let the columns in $\mathcal{V}$ corresponding to the columns $\eta(\pi,x)$ in $[K^{(\mathcal{A},\pi)}(m)]$ be denoted by $\eta^*(x)$ where the same argument x occurs in both vectors if they are in the same place. Choose a maximal set of linearly independent column vectors in $\mathcal{V}$ such that the vectors chosen are the first vectors, according to their order in $\mathcal{V}$, satisfying the required property (maximal linearly independent set). Denote the matrix whose columns are the above chosen columns ordered according to their original order in $\mathcal{V}$ by $H^{\mathcal{A}^*}$ (this step relies on Theorem 1). Finally, construct the matrix $H^{\mathcal{A}^*}(\sigma)$ as follows: For every column $\eta^*(x)$ in $H^{\mathcal{A}^*}$, let the corresponding column in $H^{\mathcal{A}^*}(\sigma)$ be the column $\eta^*(\sigma x)$.

Step 8: Solve the equations $A^*(\sigma)H^{\mathcal{A}^*} = H^{\mathcal{A}^*}(\sigma)$ for every $\sigma\in\Sigma$, subject to the condition that all the entries in $A^*(\sigma)$ be integral and nonnegative. If for some σ no solution can be found then go to step 6. Else go to the next step.

Explanation: The first column in $H^{\mathcal{A}^*}$ is a vector with all its entries equal to 1 while the entries in $A^*(\sigma)$ be integral and nonnegative. It follows that there may be only finitely many matrices $A^*(\sigma)$ satisfying the equation in step 8 so that this step is decidable.

Step 9: Let π^* be the first row of B^* then $(\pi, \{A^*(\sigma)\}, \eta^*)$ is an m-state IS equivalent to the given one. (The reader will prove this easily on the basis of the previous Theorems and Corollaries.) Stop.

Step 10: (This step is applicable only if the chosen B^* matrix in step 4 has one or more zero columns. For the sake of simplicity, we assume here that B^* has only one all-zero column, the last one. The other cases are dealt with similarly. Of course, B^* cannot be an all-zero matrix.) Let $B^{*\prime}$ be the matrix derived from B^* be deleting its last (all-zero) column. Construct a matrix $\mathcal{V}'$ with m-1 rows satisfying the equation $B^{*\prime}\mathcal{V}' = [K^{(\mathcal{A},\pi)}(m)]$ with all its entries nonnegative integers, with all its rows (when considered as point vectors) in the interior or on the boundary of $\mathcal{S}([K^{(\mathcal{A},\pi)}(m)])$ and with all entries in its first column equal to 1. Let the columns in $\mathcal{V}'$ corresponding to the columns $\eta(\pi, x)$ in $[K^{(\mathcal{A},\pi)}(m)]$ be denoted $\eta^{*\prime}(x)$ as in step 7. Let $\mathcal{V}'(m-1)$ be the submatrix of $\mathcal{V}'$ with columns corresponding to words x with $|x| \leq m-1$. Construct the matrix $\mathcal{V}'(m-1)(\sigma)$ as follows: for every column $\eta^{*\prime}(x)$ in $\mathcal{V}'(m-1)$ let the corresponding column in $\mathcal{V}'(m-1)(\sigma)$ be the column $\eta^{*\prime}(\sigma x)$. Finally, expand the matrix $\mathcal{V}'$ to a matrix $\mathcal{V}$ with m rows as follows: The first m-1 rows of $\mathcal{V}$ are as in $\mathcal{V}'$. The subvector of the last row of $\mathcal{V}$ which belongs to columns corresponding to words x with $|x| \leq m-1$ (considered as a point vector) is in the interior or on the boundary of $\bigcup_{\sigma\in\Sigma}\mathcal{S}(\mathcal{V}'(m-1)(\sigma))$. The entries in the last row of $\mathcal{V}$ which belong to columns corresponding to words x with $|x| = m$ are left free at this stage of the algorithm and will be fixed, if possible, at a later stage. The matrix $\mathcal{V}$ above should be chosen so that it differs in its fixed entries, from any other matrix $\mathcal{V}$ chosen in a previous application of step 10 of the algorithm. If no such matrix $\mathcal{V}$ can be found then go to step 4; otherwise, go to the next step.

Explanation: Assume that the equations $A^{*\prime}(\sigma)\mathcal{V}'(m-1) = \mathcal{V}'(m-1)(\sigma)$ have solutions $A^{*\prime}(\sigma)$, such that $A^{*\prime}(\sigma)$ is an (m-1)x(m-1) nonnegative integral matrix, for every $\sigma\in\Sigma$. Then an (m-1)-state equivalent IS to the given IS $\mathcal{A}_\pi$ can be constructed (by Corollary 12 and by the fact that $B^{*\prime}\mathcal{U}'(m) = [K^{(\mathcal{A},\pi)}(m)])$. This is impossible for in this case the algorithm would have stopped with a positive answer at an earlier stage. One may assume, therefore, that there is a $\sigma\in\Sigma$ such that no $(m-1)\times(m-1)$ nonnegative and integral matrix $A^{*\prime}(\sigma)$ exists which solves the equation $A^{*\prime}(\sigma)\mathcal{U}'(m-1) = \mathcal{U}'(m-1)(\sigma)$. One must, therefore, expand this equation to the equation $A^{*}(\sigma)\mathcal{U}(m-1) = \mathcal{U}(m-1)(\sigma)$ with $\mathcal{U}(m-1)$ having m rows, and try to solve this equation for an $m\times m$ matrix $A^{*}(\sigma)$ having non-zero (and integral) entries, in that part of its last column corresponding to first $m-1$ rows. This implies that the last row of $\mathcal{U}(m-1)$ must be in the interior or on the boundary of $\bigcup_{\sigma\in\Sigma}\mathfrak{S}(\mathcal{U}'(m-1)(\sigma))$ as required. It is easily seen that the number of possible matrices $\mathcal{U}$ as constructed in step 10 for fixed m is finite.

Step 11: From the matrices $\mathcal{U}(m-1)$ and $\mathcal{U}(m-1)(\sigma)$ as constructed in the previous step, construct the matrices $H^{\mathcal{A}*}$ and $H^{\mathcal{A}*}(\sigma)$ respectively, as in step 7. Some, but not all, entries in the last row of $H^{\mathcal{A}*}(\sigma)$ may not be fixed yet. For example, the first column of $H^{\mathcal{A}*}(\sigma)$ (for any σ) has the form $\eta^{*}(\sigma)$ which is a column in $\mathcal{U}(m-1)$ provided that $m\geq 2$ and this will always be assumed (the other case is trivial). Thus, all the entries in the first column of $H^{\mathcal{A}*}(\sigma)$ are fixed, for any $\sigma\in\Sigma$.

Step 12: Solve the equations:

$$A^{*}(\sigma)H^{\mathcal{A}*} = H^{\mathcal{A}*}(\sigma)$$

for $m\times m$ matrices $A^{*}(\sigma)$ with nonnegative and integral entries. If no such solution exists then go to step 10; otherwise, go to step 9.

Explanation: The matrix $A^{*}(\sigma)$ must have nonnegative and integral entries and the sum of its columns must equal the first column of $H^{\mathcal{A}*}(\sigma)$ (which is fixed). This implies that there are only finitely many possible solutions to the equations in step 12 which can be ennumerated and checked one after another. This observation (which is true also for the eighth step in the algorithm) leads to the following.

Corollary 14: Given an n-state IS$\mathcal{A}_{\pi}$, there are finitely many equivalent m-state IS$\mathcal{A}^{*}_{\pi^{*}}$ to it for any fixed m (including the case where there is no m-state equivalent IS for the given one).

Remark: If a solution to the equations in step 12 can be found such that if fits the fixed entries in $H^{\mathcal{A}^{*}}(\sigma)$, then the free entries in those matrices (and also in$\mathcal{V}(m)$) will be fixed by that solution.

e) Representability of integral word functions

The following problem will be considered in this section. Given an integral word function f over an alphabet Σ, $f : \Sigma^{*} \rightarrow N$ (where by "given" we understand that there is an effective procedure by which the values $f(x)$ can be computed in finitely many steps for each $x \in \Sigma^{*}$). Is the given function representable in the form $f = f^{\mathcal{A}_{\pi}}$ where $\mathcal{A}_{\pi}$ is an IS?

Definition 6: Let $f : \Sigma^{*} \rightarrow N$ be an integral word function. Let $\lambda, x^{1}, x^{2}, \ldots$ be a length preserving enumeration of the words in Σ^{*} and let $\mathcal{H}(f)$ be the infinite matrix whose i-j entry is $f(x^{i} x^{j})$. The rank of $f(r(f))$ is defined as the rank of the matrix $\mathcal{H}(f)$; i.e., the maximal number of linearly independent rows (or columns) in it. Notice that $r(f) = \text{rank}\,\mathcal{H}(f)$ may assume an infinite value.

The following theorems can now be proved (the reader is refered to Paz (1971) p. 134 for proofs of similar theorems).

Theorem 15: If $f = f^{\mathcal{A}_{\pi}}$ where $\mathcal{A}_{\pi}$ is an n-state IS then f has finite rank and $r(f) \leq n$.

Theorem 16: If f is a given integral word function such that $r(f) \leq n$, then a "pseudo integral" sequential system $\mathcal{A}_{\pi}$ (meaning that the matrices $A(\sigma)$ and vectors π and η are not necessarily nonnegative and integral, but the function $f^{\mathcal{A}_{\pi}}$ has only nonnegative and integral values)can be found with number of states $\leq n$ and such taht $f = f^{\mathcal{A}_{\pi}}$.

In addition to the above two theorems, one can also prove the following additional theorem which is peculiar to the integral nonnegative case (and is not true, in general).

Theorem 17: Let f be a given integral word function such that $r(f) \leq n$. If $f = f^{\mathcal{A}^*\pi^*}$ with $\mathcal{A}^*_{\pi^*}$ a true IS then the true IS $\mathcal{A}^*_{\pi^*}$ (or an equivalent true IS) can be found.

Proof: Let $\mathcal{A}_\pi$ be the pseudo IS satisfying $f = f^{\mathcal{A}_\pi}$ as constructed in Theorem 16. For any m, the matrix $[K^{(\mathcal{A},\pi)}(m)]$ has only integral nonnegative values (the entries in that matrix are of the form $\pi(x)\eta(y) = f^{\mathcal{A}_\pi}(xy) = f(xy)$ with $x, y \in \Sigma^*$) although the matrices $A(\sigma)$ and the vectors π and η may have negative and nonintegral values.

Delete step 2 from the algorithm proving Theorem 13 and apply the modified algorithm to the pseudo TS $\mathcal{A}_\pi$ above. It is easily seen that the modified lagorithm will search for a true m-state equivalent IS $\mathcal{A}^*_{\pi^*}$ with m growing larger and larger until such an equivalent IS is found and then stop. The algorithm will not stop and will run forever only if there is no true IS equivalent to the given pseudo IS$\mathcal{A}_\pi$.

sary condition for an integral word function f to be representable in the form $f = f^{\mathcal{A}_\pi}$ with $\mathcal{A}_\pi$ an IS.

Theorem 18: Let f be an integral word function. f is representable in the form $f = f^{\mathcal{A}_\pi}$, with $\mathcal{A}_\pi$ an IS, only if for every $x \in \Sigma^*$ there exists a set of numbers $c_0, c_1 \ldots , c_{n-1}$ such that for every $y, z \in \Sigma^*$ the following equality holds:

$$f(yx^n z) = \sum_{i=1}^{n} c_{n-i} f(yx^{n-i} z)$$

For proof see Paz (1971) p. 137.

f) Closure properties of ISF

Theorem 19: Any word function f over an alphabet Σ of the form $f(x) = c$ for all $x \in \Sigma^*$ with c a nonnegative integer is an ISF.
Proof: Let $\pi = (c\ 0)$, $A(\sigma) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ for all $\sigma \in \Sigma$ and $\eta = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ then $\pi A(x)\eta = c$ for all $x \in \Sigma^*$.

Theorem 20: If $f^{\mathcal{A}_\pi}$ and $f^{\mathcal{A}_{\pi'}}$ are ISF over the same alphabet Σ then so is $c_1 f + c_2 f^{\mathcal{A}'_{\pi'}}$ where c_1 and c_2 are nonnegative integers. and $(c_1 f^{\mathcal{A}_\pi} + c_2 f^{\mathcal{A}'_{\pi'}})(x) = c_1 f^{\mathcal{A}_\pi}(x) + c_2 f^{\mathcal{A}'_{\pi'}}(x)$.

Proof: Let $\mathcal{A}^*_{\pi^*}$ be defined as follows:

$$\pi^* = [c_1\pi \;\; c_2\pi'] \quad A^*(\sigma) = \begin{Bmatrix} A(\sigma) & 0 \\ 0 & A'(\sigma) \end{Bmatrix} \quad \eta^* \begin{Bmatrix} = \eta \\ \eta' \end{Bmatrix}$$

It is easy to see that $f^{\mathcal{A}^*_{\pi^*}} = c_1 f^{\mathcal{A}_\pi} + c_2 f^{\mathcal{A}'_{\pi'}}$.

Corollary 21: The equivalence problem for two IS over the same alphabet is decidable (i.e., one can decide effectively whether two IS are equivalent).

Proof: Let $f^{\mathcal{A}_\pi}$ and $f^{\mathcal{A}'_{\pi'}}$ be two ISF. Construct the IS $\mathcal{A}^*$ with $A^*(\sigma)$ and η^* as in theorem 20. Let $\pi^1 = (\pi 0 \ldots 0)$ and $\pi^2 = (0 \ldots 0 \pi')$ where the number of zero entries in π^1 and π^2 equals the dimension of π' and π respectively. It is clear that $f^{\mathcal{A}_\pi} = f^{\mathcal{A}'_{\pi'}}$ if and only if π^1 and π^2 are equivalent vectors for $\mathcal{A}^*$.

Now use Theorem 2.

Theorem 22: If $f^{\mathcal{A}_\pi}$ and $f^{\mathcal{A}'_{\pi'}}$ are ISF over the same alphabet Σ then so is $f^{\mathcal{A}_\pi} \cdot f^{\mathcal{A}'_{\pi'}}$ where $(f^{\mathcal{A}_\pi} \cdot f^{\mathcal{A}'_{\pi'}})(x) = f^{\mathcal{A}_\pi}(x) \cdot f^{\mathcal{A}'_{\pi'}}(x)$.

Proof: Define the IS $\mathcal{A}^*_{\pi^*}$ with $\pi^* = \pi \otimes \pi'$, $A^*(\sigma) = A(\sigma) \otimes A'(\sigma)$ and $\eta^* = \eta \otimes \eta'$ where the operation $\otimes$ stands for Kronecker product of matrices; i.e., if $A = [a_{ij}]$ and $B = [b_{kl}]$ are (not necessarily square) matrices of order $m \times n$ and $p \times q$ respectively then $A \otimes B = C = [C_{ik,jl}] = [a_{ij} b_{kl}]$ by definition and the double indices ik, jl of the elements of C are ordered lexicographically

$$ik = 11, 12, \ldots, 1p, \ldots, m1, \ldots, mp;$$
$$jl = 11, 12, \ldots, 1q, \ldots, n1, \ldots, nq;$$

One proves easily that $f^{\mathcal{A}^*_{\pi^*}} = f^{\mathcal{A}_\pi} \cdot f^{\mathcal{A}'_{\pi'}}$ (see Paz (1971) pp. 101, 147).

Theorem 23: Let $f^{\mathcal{A}_\pi}$ be an ISF and define the word function g_y over the same alphabet Σ with $y \in \Sigma^*$ as $g_y(x) = f^{\mathcal{A}_\pi}(yx)$. Then $g_y(x)$ is an ISF.

Let $f^{\mathcal{A}_\pi}$ be an ISF and define the word function g over the same alphabet Σ as $g(\lambda) = 1$ and $g(\sigma x) = f^{\mathcal{A}_\pi}(x)$ for all $\sigma \in \Sigma$ and all

$x \in \Sigma^*$. Then g is an ISF:

Proof: Define the IS $\mathcal{A}^*_{\pi^*}$ such that $\mathcal{A}_* = \mathcal{A}$ and $\pi^* = \pi A(y) = \pi(y)$. Then $\pi^* A^*(x)\eta^* = \pi A(y) A(x)\eta = f^{\mathcal{A}\pi}(yx)$. Thus, $g_y = f^{\mathcal{A}^*\pi^*}$ as required.

Define the IS $\mathcal{A}^*_{\pi^*}$ as follows:

$$\pi^* = (1\ 0 \ldots 0)\ ,\ A^*(\sigma) = \left\{\begin{matrix} 0 & \pi \\ \cdot & \\ \cdot & A(\sigma) \\ \cdot & \\ 0 & \end{matrix}\right\} \quad \eta^* = \left\{\begin{matrix} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{matrix}\right\}$$

where π^*, $A^*(\sigma)$ and η^* are vectors and matrices of dimension $n+1$ if the dimensions of the vectors and matrices of $\mathcal{A}_\pi$ are n. Clearly, $g(x) = f^{\mathcal{A}^*\pi^*}(x)$.

g) Single letter case

All the properties of IS and ISF proved so far, are true, of course, for the case where the alphabet Σ, over which the functions are considered, consists of a single letter. There are, however, some additional properties peculiar to the single letter case. These properties will be discussed in this section.

Given a word function over a single letter alphabet $\Sigma = \{\sigma\}$, $f : \Sigma^* \to N$, we shall use the notation $f(n)$ for $f(\sigma^n)$ so that the function is considered as a function $f : N \to N$.

Theorem 24: For integral word-functions f over a single letter alphabet the following 4 conditions are equivalent:

(1) $f = f^{\mathcal{A}_\pi}$ for some pseudo IS $\mathcal{A}_\pi$.

(2) The infinite Hankel matrix of $f, \mathcal{H}(f)$ such that its ij entry equals $f(i+j)$, is of finite rank.

(3) The generating function of the infinite series $\sum_{i=0}^{\infty} f(i)x^i$ is rational (i. e., there are two polynomials in x; $p(x)$ and $q(x)$, such taht $p(x) = q(x) \sum_{i=0}^{\infty} f(i)x^i$ where equality means that the coefficients of x^i are the same in both sides of the equation).

(4) There exists an integer $n \leq$ the number of states of $\mathcal{A}_\pi$ and constants $c_0, c_1 \ldots c_{n-1}$, such that for every integer $m \geq 0$ the following difference equation holds true

$$f(m+n) = c_{n-1} \; f(m+n-1) + c_{n-2} \; f(m+n-2) + \ldots + C_0 f(m) \qquad (12)$$

For proof see Paz (1971, b).

Every one of the four aspects exhibited in the above theorem can be helphul in the study of ISF and the growth function represented by them. Thus, the generating function approach has been used extensively by Szilard (1972) while the first and second aspect are dealt with in this paper, in a more general context.

We would like to stress, here, also, the usefulness of the 4-th condition which is exhibited in the following theorems:

Theorem 25: The growth of an ISF over a single letter is either polynomial or exponential or a combination of polynomial and exponential growth.
Proof: The relation (12) considered as a difference equation, homogeneous with constant coefficients, has solutions of the types stated in the theorem only.

Remark: The general solution of the difference equation (12) depends on initial conditions and it may happen that the growth of a specific solution is polynomial for at particular set of initial conditions and the growth is exponential for another set of initial conditions. It can also happen in other cases that the growth is polynomial for any set of initial conditions. Those cases are worth mentioning when applications to biological growth are considered.

Theorem 26: Let $f = f^{\mathcal{A}_\pi}$ be an ISF over a single letter such that for some integers m and n, $f(m) = f(m+1) = \ldots = f(m+n)$ but there is $i > n$ such that $f(m+i) \neq f(m)$ then $\mathcal{A}_\pi$ has at least $n+1$ states.
Proof: if $\mathcal{A}_\pi$ has no more than n states then (12) holds true with $n_1 \leq n$ constants. Insert the values $f(m+n_1) = f(m+n_1-1) = \ldots = f(m)$

into it for the given m (and after cancelling the equal values) we get that $\sum_{i=0}^{n_1-1} c_i = 1$.

Let i be the first integer $i > n$ such that $f(m) \neq f(m+i)$ and insert now into (12) the values $f(m+i) \neq f(m+i-1) = f(m+i-2) = \ldots = f(m+i-n)$. We have $f(m+i) = \sum_{j=1}^{n_1} c_{n_1-j} f(m+i-j) = f(m+i-1) \sum_{j=1}^{n_1} c_{n_1-j} = f(m+i-1)$ a contradiction.

Corollary 27: Let f be a word function over a single letter alphabet such that for every integer n there are integers m and $i > n$ such that $f(m+i) \neq f(m+n) = f(m+n-1) = \ldots = f(m)$ then f is not an ISF.
Proof: By Theorem 26 any IS representing f must have infinitely many states.

3. GROWTH FUNCTIONS OF LINDENMAYER SYSTEMS

a) Growth in DOL-systems

We begin by defining the notions of a DOL-system and its growth function.

A deterministic informationless Lindenmayer system or, shortly, a DOL-system is an ordered triple

$$S = (\Sigma, v, \delta) , \qquad (13)$$

where Σ is a finite nonempty set (the alphabet), $v \in \Sigma^+$ (the axiom) and δ is a mapping of Σ into Σ^* . (Σ^* was defined before. Σ^+ is the set of all nonempty words over Σ.) By considering δ as a homomorphism, we define $\delta(w)$, for any $w \in \Sigma^*$. By definition, $\delta^0(w)=w$ and δ^i denotes the composition of i copies of δ, for $i \geq 1$. The language generated by the DOL-system S is defined by

$$L(S) = \{ \delta^n(v) \mid n \geq 0 \}$$

and its growth function by

$$f_S(n) = |\delta^n(v)| , \quad n \geq 0 ,$$

where (as before) vertical bars denote the length of the word.

For $\delta \in \Sigma$, the pair $(\sigma, \delta(\sigma))$ is written $\sigma \rightarrow \delta(\sigma)$ and called a production. Our system is propagating or, shortly, a PDOL-system if δ is a mapping into Σ^+ , i. e. , $\delta(\sigma) \neq \lambda$, for each $\sigma \in \Sigma$. As usual, the system being an L-system means that rewriting happens in a parallel manner, i. e. , each letter is rewritten at every step of a derivation. The system being a 0-system means that rewriting is context-free, i. e. , the individual letters (the "cells") do not communicate with each other. Finally, the system being deterministic means that, for each $\sigma \in \Sigma$, there is exactly one production with σ on its left side. The general theory of integral sequential word functions developed in Part 2 is directly applicable to the growth functions of DOL-systems. In fact, the latter correspond to the single letter case of word functions. The general case will be applied to DTOL-systems in Section 3c. The context-dependent DL-systems considered in Section 3b possess an entirely different theory of growth.

As an example, consider the PDOL-system

$$S = (\{a, b\},\ a,\ \{a \rightarrow b,\ b \rightarrow ab\}).$$

The consecutive values of $f_S(n)$ in this case form the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, 21, ...

The growth equivalence problem for the class of DOL-system is the problem of deciding for any two DOL-systems whether or not their growth functions are the same. The growth equivalence problem for any class of deterministic L-systems is defined in the same way.

For DOL-systems, the problem of finding growth equivalent axioms is defined as follows. Given a DOL-system (13), to find all DOL-systems with the same growth function, Σ and δ as (13). Clearly, the number of such systems is finite since the new axiom has to be of the same length as v. The cell minimazation problem consists of finding, for any given DOL-system, a growth equivalent (i.e., having the same growth function) DOL-system with minimal cardinality of the alphabet. The following problem of realizing a given growth with a given number of cells is more general: given any DOL-system S and an integer $k \geq 1$, to find all DOL-systems which are growth equivalent to S and whose alphabet consists of k letters. (of course, there may be no such DOL-systems.) Finally the problem of realizing a function g, from nonnegative integers into nonnegative integers, as a growth function consists of finding, for any such g, a DOL-system S with $g = f_S$, provided such a system S exists.

We will now study each of these problems, using the results established in Part 2.

For a DOL-system (13) with the alphabet $\Sigma = \{a_1, \ldots, a_k\}$, we define the following matrices. The <u>initial vector</u>, π, is the k-dimensional row vector such that its ith component equals the number of occurrences of the letter a_i in the axiom v, for i=1, ... , k. The final vector, η, is the k-dimensional column vector with all components equal to 1. The <u>growth matrix</u>, A, is the k-dimensional square matrix whose (i, j)th entry equals the number of occurrences of a_j in $\delta(a_i)$, for i, j=1,...,k. These matrices are introduced because from the point of view of growth the order of letters in v and in each $\delta(a_i)$ is immaterial. The following theorem is a direct consequence of the definitions.

<u>Theorem 28</u>: For any DOL-system S, its growth function can be expressed in the form

$$f_S(n) = \pi A^n \eta , \tag{14}$$

where A^0 is the identity matrix I. Furthermore, if m is the lengt of the longest right side of the productions then

$$f_S(n) \leqq m^n |v| , \text{ for all } n \geqq 0 . \tag{15}$$

The representation (14) reduces the theory of growth functions of DOL-systems to the theory of integral sequential word functions (single letter case). The inequality (15) can be replaced by more detailed characterization in Theorem 25.

We now use Theorem 2 to solve the porblem of finding growth equivalent axioms.

<u>Theorem 29</u>: An algorithm for finding all growth equivalent axioms consists in finding all solutions π^2 to the equation (1), where π^1 is the initial vector of the given DOL-system.

As an example, consider the PDOL-system with the axiom $a^2 b^2 c^3$ and productions $a \to ab^2 c^4$, $b \to a^2 b^4 c^8$, $c \to a^4 b^8 c^{16}$. Its representation in terms of π, A, η is:

$$\pi = (2\ 2\ 3) \qquad A = \begin{Bmatrix} 1 & 2 & 4 \\ 2 & 4 & 8 \\ 4 & 8 & 16 \end{Bmatrix} \qquad \eta = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

The H matrix for it is

$$H = \begin{Bmatrix} 1 & 7 \\ 1 & 14 \\ 1 & 28 \end{Bmatrix}$$

The equation $\pi H = (x\ y\ z)H$ has only two solutions (x, y, z):

$$(2, 2, 3) \quad \text{and} \quad (0, 5, 2) .$$

The first corresponds to the original axiom. Hence, the only other growth equivalent axiom is $b^5 c^2$.

The generating function of $f_S(n)$ is defined to be the formal sum

$$F_S(x) = \sum_{n=0}^{\infty} f_S(n) x^n .$$

Theorem 30: For any DOL-system S , the generating function of its growth function equals $\pi(I-Ax)^{-1}\eta$. The growth equivalence problem for DOL-systems is solvable.

Proof: We note first that the matrix $I-Ax$ is nonsingular because the elements of its main diagonal are of the form $1-ax$, whereas the remaining elements are of the form $a'x$. The first sentence of the theorem now follows by the representation (14) and the matrix equation

$$(I-Ax)^{-1} = \sum_{n=0}^{\infty} A^n x^n .$$

The generating function thus obtained is of the form $p(x)/q(x)$, where p and q are polynomials with integer coefficients. For another DOL-system S_1 with the generating function $p_1(x)/q_1(x)$ for its growth function, S and S_1 are growth equivalent if and only if $p(x)q_1(x) = q(x)p_1(x)$, where the equality sign denotes the identity of the polynomials. Hence, the second sentence of the theorem follows.

The same decision method for PDOL-systems has been given by Szilard (1972). Another method has been given by Doucet (1972). A further decision method results as a special case of Corollary 21. Note also that Theorem 30 gives a method of determining the growth function of any DOL-system.

By Theorem 13 and Corollary 14 (cf. also the proof of Theorem 17), we obtain the following results.

Theorem 31: The cell minimization problem for DOL-systems is solvable, and so is the problem of realizing a given growth with a given number of cells.

We mention another application of Corollary 14:

Theorem 32: For any DOL-system S and integer k, there is only a finite number of DOL-systems growth equivalent to S and having k letters in their alphabet.

The problem of realizing a function as a growth function has been studied extensively by Szilard(1972). His methods give the answer for the case of PDOL-realizations of polynomials. Theorem 17 gives the following general result.

Theorem 33: There is an effective procedure with the following properties. Given a function g (from nonnegative integers into nonnegative integers) and a finite upper bound n for the rank of g, the procedure will output a DOL-system whose growth function equals g, provided such a system exists. If there is no such system, the procedure will run forever.

The procedure of Theorem 33 does not work if no upper bound n is given. However, if g results from experiments with a finitary device, it is clear that such an upper bound exists.

In many cases the closure properties discussed in Section 2f will give more practical methods for realizing functions as growth functions. For instace, the growth function of the PDOL-system with the axiom a and productions a→ab, b→b equals the function n+1. If one wants to realize $(n+1)^2$ as a growth function, then one simply takes the Kronecker products of the matrices of the given system, obtaining the PDOL-system with the axiom a and productions a → abcd, b → bd, c → cd, d → d. The new system realizes the growth $(n+1)^2$ but it is not minimal in terms of the number of letters. (Kronecker products usually give systems with more cells than necessary.) However, one can always apply the cell minimization procedure.

Following Szilard (1972), we say that the growth in a DOL-system S is malignant if there is no polynomial p(n) such that $f_S(n) \leq p(n)$, for all n. The following theorem is easily obtained from the results of Szilard (1972).

Theorem 34: There is an algorithm for deciding whether or not the growth in a DOL-system is malignant.

Whether or not the growth is malignant is determined by the difference equation (12) and its initial conditions. As we pointed out in Section 2g, it may happen that the same productions give rise to both malignant and "normal" growth, f or suitable choices of the axiom. Of course, it may also happen that the growth is malignant, no matter how we choose the axiom, and also that the growth is normal no matter how we choose the axiom.

b) Context-dependent DL-systems

We will now consider the case where the rewritting may depend on the context. The productions are now of the form

$$(b, a, c) \rightarrow w , \quad b, a, c \in \Sigma , \quad w \in \Sigma^* ,$$

meaning that an occurrence of the letter a lying between b and c is rewritten as w . If this occurrence of a is the first or last letter of the word under scan, the missing context is provided by a fixed letter g , so-called input from the environment.

Formally, a deterministic context-dependent Lindenmayer system or, shortly, a D2L-system is an ordered quadruple $S = (\Sigma, v, g, \delta)$, where Σ and v are as in the definition of a DOL-system, $g \in \Sigma$ and δ is a mapping of the Cartesian power Σ^3 into Σ^* . If δ is a mapping into Σ^+ , the system is termed propagating or a PD2L-system.

We now define a mapping δ' of Σ^* into Σ^* . For $w = a_1 \ldots a_n$, where $n \geq 2$ and each a_i is a letter,

$$\delta'(w) = \delta(g, a_1, a_2)\delta(a_1, a_2, a_3) \ldots \delta(a_{n-2}, a_{n-1}, a_n)\delta(a_{n-1}, a_n, g) .$$

(Juxtaposition on the right side denotes the catenation of words.) For $w = a_1 \in \Sigma$, $\delta'(w) = \delta(g, a_1, g)$. Finally, $\delta'(\lambda) = \lambda$. The language generated by S is now defined by

$$L(S) = \{(\delta')^n(v) \mid \quad n \geq 0\}.$$

and its growth function by

$$f_S(n) = |(\delta')^n(v)| , \quad n \geq 0 .$$

A D2L-system is a D1L-system if and only if one of the following conditions holds: (i) for all letters a, b, c, d, $\delta(a,b,c) = \delta(a,b,d)$, or (ii) for all letters a, b, c, d, $\delta(a,b,c) = \delta(d,b,c)$. Thus, the numbers o, 1, 2 in the definition of L-systems mean, respectively that rewriting happens in a context-free, one-sided context-sensitive or two-sided context-sensitive manner. (As regards cells in filamentous organisms, the three alternatives mean, respectively, that individual cells do not communicate, or a cell may communicate with its neighbour which either is always the one on the left or always the one on the rigth or, finally, that a cell may communicate with both of its neighbours.)

As an example, consider the PD2L-system

$$S = (\{a, b, c, g\},\ ba,\ g,\ \delta)\ ,$$

where δ is defined by

$$\begin{aligned}
\delta(b, a, x) &= b, && \text{for all } x \neq c,\\
\delta(x, a, c) &= c, && \text{for all } x \neq b,\\
\delta(x, b, g) &= ac, && \text{for all } x,\\
\delta(x, b, y) &= a, && \text{for all } x \text{ and } y \text{ such that } y \neq g\ ,\\
\delta(a, c, x) &= a, && \text{for all } x,\\
\delta(g, c, x) &= b, && \text{for all } x,\\
\delta(x, y, z) &= y, && \text{otherwise.}
\end{aligned}$$

The sequence of words $(\delta')^n(ba)$ is

ba, ab, aac, aca, caa, baa, aba, aab, aaac, aaca, acaa, caaa, baaa, abaa, aaba, aaab, aaaac, aaaca, ... ,

and the first values of the growth function

2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, ...

This example can be given the following interpretation. A filamentous organism grows only at its tail. Whenever growth has taken place, a message goes to the head which, in turn, sends back an instruction for another piece of growth. The more the organism grows, the more time it takes for these messages to get through.

By definition, the family of growth functions of contextdependent DL-systems includes the family of growth functions of informationless DL-systems. By the previous example and Corollary 27, we obtain the following theorem.

Theorem 35: There is a deterministic context-dependent Lindenmayer system whose growth function is not realizable by any DOL-system.

Our example for Theorem 35 is a PD2L-system but it can easily be replaced by a PD1L-system. In fact, Gabor Herman (personal communication) has constructed the following very slowly propagating PD1L-system, where the lengths of the sequences of equal values grow exponentially. The axiom is ad, the input from the environment is g and the productions are

$$(g,a) \to c,\ \ (c,a) \to b,\ \ (c,b) \to c,\ \ (c,d) \to ad,\ \ (x,c) \to a,\ \ \text{for all } x\,.$$

Rewriting depends always on the left neighbour only and, thus, the right neighbour is missing from the left sides of the productions. (For instance,the first production means that an initial occurence of a is rewritten as c.) For all combinations not listed above, rewriting preserves the original letter. The first words in the sequence are now

ad, cd, aad, cad, abd, cbd, acd, caad, abad, cbad, acad, cabd, abbd, cbbd, acbd, cacd, abaad, ...

Note that growth can take place only after the messenger c has reached d. This, in turn, can happen for words $cb^i d$ only. In the above sequence, the distance between two words of this form grows exponentially.

Thus, the class of growth functions of PD1L-systems (resp. D1L-systems) properly includes the class of growth functions of PDOL-systems (resp. DOL-systems). It is an open problem whether or not there exists a PD2L-system (resp. D2L-system) whose growth function cannot be realized by any PD1L-system (resp. D1L-system). This problem can be further extended to concern D(m,n)L-systems, i.e., systems where the rewriting of each letter depends on m og its left neighbours and on n of its rigth neighbours. It has been shown by Rozenberg (1971, 2) that the families of languages generated by such systems form an infinite hierarchy. This does not imply that the families of growth functions also form an infinite hierarchy. Another open problem is to give a decision method for the growth equivalence problem of deterministic context-dependent Lindenmayer systems, perhaps only for a subclass of them such as

PD1L-systems. No algorithm is known for deciding whether or not the growth in a context-dependent Lindenmayer system is malignant.

Comparing finite probabilistic and deterministic automata, it is well known that the former save states, i.e., there is a probabilistic automaton with two states which, for any k, accepts a language not acceptable by any deterministic automaton with fewer than k states but acceptable by a deterministic automaton with k states. A similar phenomenon is observed when comparing the growth functions of context-dependent and informationless L-systems. In the statement of the following theorem, a semi-PD1L-system means a PD1L-system without the axiom.

Theorem 36: There is a semi-PD1L-system S with three letters (including the input from the environment) such that, for each $k \geq 2$, there is an axiom v_k and a PDOL-system T_k with k letters which satisfy both of the following conditions: (i) the growth function of T_k cannot be realized by any PDOL-system with fewer than k letters, (ii) The growth function of T_k equals the growth function of S_k, the PD1L-system obtained from S by adding the axiom v_k.

Proof: Define $S = (\{a, b, c\}, \ . \ , a, \delta)$, where for all letters x,

$$\delta(b, a, x) = b, \quad \delta(a, b, x) = a, \quad \delta(b, c, x) = aa$$

and $\delta(x, y, z) = y$, otherwise. Furthermore, for each $k \geq 2$, define

$$v_k = ba^{k-2}c,$$

$$T_k = (\{a_1, \ldots, a_k\}, \ a_1{}^{k-1}a_2, \ \delta_k),$$

$$\delta_k(a_1) = a_1, \ \delta_k(a_k) = a_1 a_1, \ \delta_k(a_i) = a_{i-1}, \text{ for } 2 \leq i \leq k-1 .$$

Then the following function f is the growth function of both T_k and S_k:

$$f(n) = \begin{Bmatrix} k & \text{for } n \leqq k-2, \\ k+1 & \text{for } n > k-2. \end{Bmatrix}$$

Condition (i) is satisfied because in any PDOL-system realizing f the axiom must contain at least two distinct letter and, for all $i \leqq k-2$, the ith word must contain a letter which is not present in the jth word, for any $j < i$.

c) Growth relations of DTOL- and OL-systems

In systems considered so far, there is a unique sequence of words beginning with the axiom. We now consider cases where this condition is not satisfied and, thus, we obtain a growth relation rather than a growth function.

A deterministic informationless Lindenmayer system with tables or, shortly, a DTOL-system is an ordered triple $S = (\Sigma, v, T)$, where Σ and v are as in the definition of a DOL-system and T is a finite nonempty collection of mappings t such that (Σ, v, t) is a DOL-system for every $t \in T$. For each DOL-system (Σ, v, t) thus obtained, we define the matrices π, $A(t)$ and η as in Section 3a.

The growth relation R_S of S is the binary relation defined as follows. For any $m, n \geqq 0$, $R_S(m,n)$ holds if and only if either $m=0$ and $n=\pi\eta$, or else $m>0$ and there are elements $t_1, \ldots, t_m$, or T such that

$$\pi A(t_1) \ldots A(t_m)\eta = n .$$

Two DTOL-systems S (with matrices π, $A(t)$, η) and S' (with matrices π', $A'(t)$, η') are strongly growth equivalent if there is a one-to-one correspondence between the set of matrices $A(t)$ and the set of matrices $A'(t)$ such that, for any $m \geqq 0$ and $t_1, \ldots, t_m$,

$$\pi A(t_1) \ldots A(t_m)\eta = \pi' A'(t_1') \ldots A'(t_m')\eta' ,$$

where $A'(t'_i)$ is the matrix corresponding to $A(t_i)$. They are weakly growth equivalent if $R_S = R_{S'}$.

Intuitively, in a DTOL-system any element of T (so-called "tables") may be applied to the word under scan but different tables may not be mixed. The language generated by the system consists of all words obtained from the axiom in this fashion. A DOL-system can be viewed as a special case of DTOL-system with only one table. If there are more than one tables, many words may be derived from the axiom in m steps and, consequently, we have a growth relation rather than a growth function. By definition, strong growth equivalence of two systems implies that the systems have the same number of tables, i. e., the same degree of synchronization in the terminology of Rozenberg (1971, 1) In weak growth equivalence, only the lengths of the words are taken into account, not the number of different ways in which words of given length may be derived.

The theory of integral sequential word functions is directly applicable to strong growth equivalence but not to weak growth equivalence. The results are summarized in the following theorem. The notions in the statement of the theorem are defined exactly as for DOL-systems, with the convention that equivalence means always strong growth equivalence. The theorem is obtained from Theorems 2 and 13 and Corollaries 14 and 21 in the same way as Theorems 29-32. It is to be emphasized that because only strong growth equivalence is considered, in each of the results one considers a family of DTOL-systems with the same degree of synchronization.

Theorem 37: There is an algorithm for finding all growth equivalent axioms for any DTOL-system. The growth equivalence problem for DTOL-systems is solvabel. The cell minimization problem for DTOL-systems is solvabel, and so is the problem of realizing a given growth with a given number of cells. For any DTOL-system S and integer k , there is only a finite number of DTOL-systems growth equivalent to S and having k letters in their alphabet.

Finally, we consider OL-systems. A OL-system is defined as a DOL-system except that now δ is a mapping into the set of all nonempty finite subsets of Σ^* . One step in the rewriting process consists in replacing each letter a by som word in $\delta(a)$. Different occurrences of the same letter may be replaced by different words in $\delta(a)$ and, therefore, matrix approach will not be directly applicable. The growth relation R_S of a OL-system S is defined as follows. For any $m, n \geq 0$, $R_S(m, n)$ holds if and only if either $m=0$ and n is the length of the axiom, or else $m>0$ and a word of length n can be obtained from the axiom as the result of m steps of rewriting. Two OL-systems or a DTOL-system and a OL-system are weakly growth equivalent if they have the same growth relation.

There are Ol-languages which are not DTOL-languages, eg. the language $\{a^{2n} \mid n \geq 0\}$ is generated by the OL-ystem S with the axiom aa and productions $a \to aa$ and $a \to \lambda$ but is not generated by any DTOL-system. However, the DTOL-system S_1 with the axiom $a_1 a_2$ and tables t such that $t(a_i) = w_i$, $i = 1, 2$, and the words w_i , independently of i , assume the values $a_1{}^2$, $a_2{}^2$, $a_1 a_2$ and λ is weakly growth equivalent to S . The same holds true also in general. The idea in the proof of the following theorem is the same as in the example: introduce new letters in such a way that if two occurrences of the same letter are rewritten differently according to the OL-system then in the DTOL-system they are replaced by two different letters a_1 and a_2 .

Theorem 38: For any OL-system, there is a weakly growth equivalent DTOL-system.

Proof: Let the given OL-system be $S = (\Sigma, v, \delta)$. Without loss of generality, we assume that the following condition is satisfied for each letter a in Σ : all letters occrurring in some of the words in $\delta(a)$ are distinct among themselves and also different from a . (For if this is not the case originally, then we replace each a in Σ with sufficiently many new letters $a_1, \ldots\ldots, a_k$, referred to as descendants of a . The new set of productions consists of all productions obtained in the following way. The left side is a descendant of some letter a . The right side is obtained from a word in $\delta(a)$ by replacing every letter vith one of its descendants in such a way that the new system satisfies the required condition. Since from the point of view of growth the descendants do not change anything, the new system is weakly growth equivalent to the original one.)

Thus, we assume that S satisfies the condition mentioned above. Let m be the maximum of the two numbers: The lenght of v and the cardinality of Σ. Consequently, there are at most m words in $\delta(a)$, for any a in Σ. (This holds true also if λ is among these words.) For each a in Σ, introduce m^2 new letters $a_1, \dots a_{m^2}$, referred to as descendants of a. Let Σ_1 be the alphabet of all the new letters thus obtained. For a word w over Σ, denote by $U(w)$ the (finite) set of words over Σ_1 which are obtained from w by replacing every letter with one of its descendants. (Different descendants.) Let $v_1 \in U(v)$ be such that different occurrences of the same letter are, in fact, replaced with different descendants. (Such a v_1 exists by the choice of the number m.) Let, finally, T be the collection of mappings t of Σ_1 into ${\Sigma^*}_1$, consisting of all mappings obtained in the following way. For each a in Σ, denote by $U_1(a)$ the union of all sets $U(w)$, where w ranges over the elements in $\delta(a)$. Consider the DTOL-system $S_1 = (\Sigma_1, v_1, T)$. We claim that S and S_1 are weakly growth equivalent.

In fact, if for some m and n we have $R_{S_1}(m,n)$ then we also have $R_S(m,n)$ because we only have to erase the indices indicating descendants to get the same growth. The converse implication follows from the subsequent observations: (i) in v_1 all letters are distinct, (ii) a step $w_1 \Rightarrow w_2$ in a derivation according to S can be simulated by a stem $w'_1 \Rightarrow w'_2$ in a derivation according to S_1 in such a way that in w'_2 the letters of w_2 are indexed to take care of the next step of the derivation. (Remember that from the growth point of view the order of letters is immaterial.) Hence, $R_S = R_{S_1}$ and Theorem 38 follows.

The language generated by the DTOL-system S with the axiom a and two tables $(a \to a^3)$ and $(a \to a^2)$ is not generated by any OL-system. In the following theorem we show that no equivalent OL-system can be obtained even if attention is restricted only to growth relations.

Theorem 39: There is no OL-system which is weakly growth equivalent to the DTOL-system S defined above. Consequently, the family of growth relations of DTOL-systems properly includes the family of growth relations of OL-systems.

Proof: Clearly, $R_S(m,n)$ holds if and only if $n = 2^{m-i} 3^i$, for some i such taht $0 \leqq i \leqq m$. Assume that there is a OL-system S_1 such that $R_S = R_{S_1}$. For each m_1, there is an $m > m_1$ such that at the mth step of the rewriting process according to S_1 it is possible to replace an occurrence of a letter a in a word w by two words w_1 and w_2 of different lengths. (Otherwise, the cardinalities of the sets

$$R_m = \{n \mid R_S(m,n)\} \quad , \quad m = 0, 1, 2, \ldots .$$

would be bounded.) Let u be greater than the greatest among the differences $|x_1| - |x_2|$, where x_1 and x_2 are the right sides of some productions of S_1 whose left sides coincide. Choose m_1 to satisfy $2^{m_1-1} > u$. Then, whenever $m > m_1$ and n_1 and n_2, $n_1 > n_2$, are such that $R_S(m, n_1)$ and $R_S(m, n_2)$ hold, we have $n_1 - n_2 > u$. A contradiction now arises because the absolute value of the difference $|w_1| - |w_2|$ is less than u. This proves our theorem.

Clearly, the growth function or growth relation of any Lindenmayer system is bounded by a function $\varphi(n) = ab^n$, where a and b are constants. Problems concerning malignant growth for systems more general than DOL-systems are left open.

Acknowledgement Most of the work in this paper was done during the Open House in Unusual Automata Theory at DAIMI of Aarhus University, Denmark, Jan. 10 - Jan. 28, 1972. The authors express their gratitude to the participants and organizers of this meeting, especially to Dr. Brian Mayoh.

REFERENCES

P. G. Doucet (1972) Growth of word length in DOL-systems, manuscript ot be published.

A. Paz (1971) Introduction to Probabilistic Automata. Academic Press.

A. Paz (1971, b) Formal series, finiteness properties and decision problems. Ann. Acad. Scient. Fennicae, Ser. A I 493.

G. Rozenberg (1971, 1) TOL-systems and languages, manuscript to be published.

G. Rozenberg (1971, 2) L-systems with interactions, manuscript to be published.

G Rozenberg and P. G. Doucet (1971) On OL-languages. Information and control 19, 302-318.

G. Rozenberg and A. Lindenmayer (1971) Deterministic developmental systems with catenative, recursive formulas, manuscript to be published.

A Salomaa (1973) Formal Languages Academic Press, to appear.

A. L. Szilard (1972) Growth functions of Lindenmayer systems. Manuscript to be published.

stichting
mathematisch
centrum

MC

REKENAFDELING

MR 138/72 SEPTEMBER

P.M.B. VITÁNYI
DOL-LANGUAGES AND A FEASIBLE SOLUTION
FOR A WORD PROBLEM

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

ABSTRACT

L-Systems are automata theoretic developmental models for filamenteous growth. In this report a subclass, the DOL-Systems, is studied by considering a classification of letters with respect to their productions. The notion of a recursive complexity structure is introduced. The properties derived are exploited, yielding a feasible algorithm for the solution of a "word problem" (i.e. the membership question) for DOL-Systems.

Necessary and sufficient conditions for the finiteness of DOL-languages are stated, and the size of a DOL-language is fixed within sharp bounds depending on the size of the alphabet and the size of classes induced by an equivalence relation on this alphabet. An ALGOL-60 implementation of the above mentioned algorithm, and a program for generating the sequence of subsequent words, are provided, both capably written by F.A.L.M. Goossens.

CONTENTS

1 INTRODUCTION

1.1 NOTATION AND PRELIMINARIES

Let $\sum$ be a finite set. Any sequence of elements of $\sum$ is called a word or string over $\sum$. If α and β are two words over $\sum$, then their concatenation is written as $\alpha\beta$. λ denotes the empty word. If $a\epsilon\sum$ then a^2 means aa, a^3 means aaa, etc. $a^0 = \lambda$. If L_1 and L_2 are two sets of words over $\sum$ then

$$L_1.L_2 = \{\alpha\beta | \alpha\epsilon L_1 \text{ and } \beta\epsilon L_2\}.$$

$\sum^*$ is the Kleenean closure of $\sum$, i.e. $\sum^* = \bigcup_{i=0}^{\infty} \sum^i$ where $\sum^0 = \{\lambda\}$ and $\sum^i = \sum^{i-1}.\sum$. $\sum^+ = \sum^* \setminus \{\lambda\}$.

If $\omega\epsilon\sum^*$ then $\gamma : \sum^* \rightarrow 2^{\sum}$ is defined by

$$\gamma(\omega) = \{a\epsilon\sum | \exists \eta,\xi\epsilon\sum^* [\omega = \eta a \xi]\}$$

$|\sum|$ denotes the number of elements of $\sum$. $|\alpha|$ denotes the length of a word α.

Let Seq $= c_0, c_1, \ldots, c_k$ be a sequence of elements of $\sum$, then $\gamma(\text{Seq}) = \{c_i \epsilon \sum | c_i$ occurs in Seq$\}$, and $|\text{Seq}| = k+1$.

A language over $\sum$ is a set $L \subseteq \sum^*$. Further notation will by and large conform to the one usual in mathematical linguistics.

Logical quantifiers: $\forall$ means "for all"

$\exists$ means "there exists at least one".

1.2 LINDENMAYER SYSTEMS

Lindenmayer or L-Systems were proposed by Lindenmayer [6] as a developmental model for filamenteous growth. They were studied formally in e.g. [1, 4 and 7].
We shall investigate some aspects of a subclass of the L-Systems: the DOL-Systems or Deterministic 0-Input Lindenmayer Systems.

Def. 1.1 A Semi-DOL-System (Semi DOL) is an ordered pair $S = <\Sigma, \delta>$ where

(i) Σ is a nonempty finite set, the alphabet of S, and an element of Σ is called a letter.

(ii) $\delta : \Sigma \to \Sigma^*$ is a total mapping, called the set of production rules, and for $\delta(a) = \alpha$ we also write $a \to \alpha$.

A Semi DOL generates words as follows:

Let $\omega = a_1 a_2 \ldots a_m \in \Sigma^+$ and

$\omega' = \alpha_1 \alpha_2 \ldots \alpha_m \in \Sigma^*$, then

ω produces or generates ω' directly, written as $\omega \Rightarrow \omega'$,

iff $\quad \forall j \in \{1,2,\ldots,m\} \; [a_j \to \alpha_j]$.

$\overset{*}{\Rightarrow}$ denotes the transitive and reflexive closure of the relation $\Rightarrow$, and $\omega \overset{*(k)}{\Longrightarrow} \omega'$ denotes a chain of length k:

$$\omega = \omega_0 \Rightarrow \omega_1 \Rightarrow \ldots \Rightarrow \omega_k = \omega'$$

If $\omega \overset{*}{\Rightarrow} \omega'$ we say ω produces, generates or derives ω', and if $\omega \overset{*(k)}{\Longrightarrow} \omega'$, then ω' is derived in k steps from ω, and $\omega \overset{*(k)}{\Longrightarrow} \omega'$ is a k-derivation of ω' from ω. $\omega \overset{+}{\Rightarrow} \omega'$ means $\omega \overset{*(k)}{\Longrightarrow} \omega'$ and $k > 0$.

We extend δ in the obvious way to δ' such that

$$\delta'(a_1 a_2 \ldots a_m) = \delta(a_1)\, \delta(a_2) \ldots \delta(a_m)$$

and omit for convenience sake the " ' " .

$$\delta^0(\omega) = \omega$$

$$\delta^i(\omega) = \delta(\delta^{i-1}(\omega)); \text{ i.e. } \omega \overset{*(i)}{\Longrightarrow} \delta^i(\omega)$$

$$\omega' \epsilon \delta^*(\omega) \text{ iff } \omega \overset{*}{\Longrightarrow} \omega'.$$

$$\gamma(\delta^*(\omega)) \overset{\text{def}}{\equiv} \bigcup_{\omega' \epsilon \delta^*(\omega)} \gamma(\omega')$$

$$\delta^i(\Sigma) \overset{\text{def}}{\equiv} \bigcup_{a \epsilon \Sigma} \delta^i(a) \text{ and } \gamma(\delta^i(\Sigma)) \overset{\text{def}}{\equiv} \bigcup_{a \epsilon \Sigma} \gamma(\delta^i(a))$$

$$\delta^*(\Sigma) \overset{\text{def}}{\equiv} \bigcup_{a \epsilon \Sigma} \delta^*(a) \text{ and } \gamma(\delta^*(\Sigma)) \overset{\text{def}}{\equiv} \bigcup_{a \epsilon \Sigma} \gamma(\delta^*(a))$$

Def. 1.2 A DOL-System (DOL) is an ordered triple

$G = <\Sigma, \delta, \sigma>$ where

(i) Σ and δ are as in def. 1.1

(ii) $\sigma \epsilon \Sigma^+$ is called the axiom of G.

Def. 1.3 The DOL-Language generated by a DOL $G = <\Sigma, \delta, \sigma>$ is defined by

$$L(G) = \{\omega \epsilon \Sigma^* \mid \sigma \overset{*}{\Longrightarrow} \omega\}, \text{ i.e. } L(G) = \delta^*(\sigma).$$

L-Systems differ from traditional grammars in the following respects:

(i) All letters of a string are rewritten simultaneously at each time step. This feature conforms to the state of affairs in natural processes which are mostly parallel as opposed to the sequential character of grammars.

(ii) Every string derived in this manner from σ belongs to L(G).

(iii) As a consequence of (i) and (ii) there is no distinction between terminal and auxiliary letters (in a sense there are no terminals).

Def. 1.4 The sequence $\xi(G)$ of words generated by $G = <\Sigma, \delta, \sigma>$ i.e.

$$\xi(G) = \sigma, \delta(\sigma), \delta^2(\sigma), \ldots$$

is called a propagation.

Example 1 $G = \langle\{a\}, \{a \to aa\}, a\rangle$

$\xi(G) = a, aa, aaaa, \ldots.$

$L(G) = \{a^{2^t} | t \geq 0\}$

Example 2 $G = \langle\{a,b\},\{a \to aba, b \to \lambda\}, a\rangle$

$\xi(G) = a, aba, abaaba, \ldots.$

$L(G) = \{(aba)^{2^t} | t \geq 0\} \cup \{a\}$

Example 3 $G = \langle\{a,b\},\{a \to b, b \to ab\}, a\rangle$

$\xi(G) = a, b, ab, bab, abbab, \ldots.$

$L(G) = \{a, b, ab, bab, abbab, \ldots.\}$

Note that the lengths of the consecutively generated words

$|a|, |b|, |ab|, |bab|, |abbab|, \ldots.$

form the main Fibonacci sequence

1, 1, 2, 3, 5, ...

Consider the DOL $G = \langle\{a,b,c\}, \{a \to aa, b \to bb\}, a\rangle$

$\xi(G) = a, aa, aaaa, \ldots.$

$L(G) = \{a^{2^t} | t \geq 0\}$

Clearly, the letter b is superfluous since it does not appear in the sequence and language produced by G.

Following Rozenberg and Lindenmayer [8] we define

Def. 1.5 A DOL $G = \langle\sum,\delta,\sigma\rangle$ is

(i) Quasi-reduced iff $\bigcup_{t \in \mathbb{N}} \gamma(\delta^t(\sigma)) = \sum$

(ii) Reduced iff $\forall j \in \mathbb{N}\ [\bigcup_{t \in \mathbb{N}} \gamma(\delta^t(\delta^j(\sigma))) = \sum]$

2 THE GENERIC POWER OF LETTERS

What type of language a DOL $G = <\Sigma,\delta,\sigma>$ generates depends on the generic or productive qualities imbued to the letters by the semi DOL $S = <\Sigma,\delta>$.

Def. 2.1 Let S $<\Sigma,\delta>$ be a semi DOL, $|\Sigma| = p$.
A letter $a\in\Sigma$ is

(i) Mortal iff $a\overset{*}{\Longrightarrow}\lambda$. $\Sigma_m = \{a|a \text{ is mortal}\}$. $p_m = |\Sigma_m|$.

(ii) Vital iff $a\notin\Sigma_m$. $\Sigma_v = \Sigma\backslash\Sigma_m$. $p_v = |\Sigma_v|$.

(a) Recursive iff $a\overset{+}{\Longrightarrow}\eta a\xi$ for some $\eta,\xi\in\Sigma^*$ *)

$\Sigma_r = \{a|a \text{ is recursive}\}$. $p_r = \Sigma_r$.

If $a\overset{+}{\Longrightarrow}\eta a\xi$ with $\eta,\xi\in\Sigma_m^*$ then letter a is monorecursive.

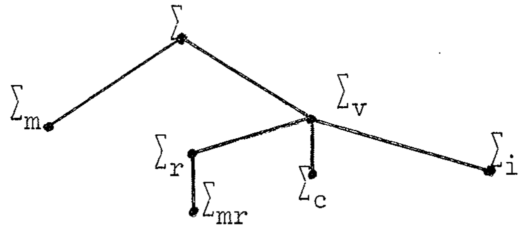
$\Sigma_{mr} = \{a|a \text{ is monorecursive}\}$. $p_{mr} = |\Sigma_{mr}|$.

(b) Recurring iff $a\notin\Sigma_r$ and there exists a letter $b\in\Sigma_r$ such that $b\overset{*}{\Longrightarrow}\eta a\xi$. $\Sigma_c = \{a\in\Sigma\backslash\Sigma_m|a \text{ is recurring}\}$. $p_c = |\Sigma_c|$.

(c) Initial iff $a\notin\Sigma_r$ and there does not exist a letter $b\in\Sigma_r$ such that $b\overset{*}{\Longrightarrow}\eta a\xi$. $\Sigma_i = \{a\in\Sigma\backslash\Sigma_m|a \text{ is initial}\}$. $p_i = |\Sigma_i|$.

NB. The distinction between recurring and initial can also be made in the case of mortal letters. Here we need this distinction only for vital letters, and we shall talk about recurring vital ($a\in\Sigma_c$) and initial vital ($a\in\Sigma_i$) letters.

The inclusion relation induces a partial ordering on $\{\Sigma, \Sigma_m, \Sigma_v, \Sigma_r, \Sigma_{mr}, \Sigma_c, \Sigma_i\}$ as follows:



*) In the sequel we shall omit "for some $\eta,\xi\in\Sigma^*$" when ever this is obviously implied.

Clearly: (i) $\sum_v \cup \sum_m = \sum$.

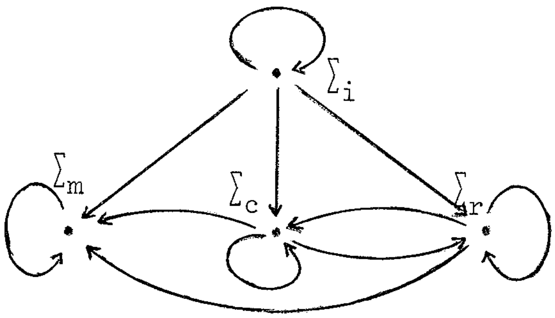
(ii) $\sum_i \cup \sum_c \cup \sum_r = \sum_v$; $\sum_{mr} \subseteq \sum_r$.

(iii) $\sum_v \cap \sum_m = \sum_i \cap \sum_c = \sum_i \cap \sum_r = \sum_c \cap \sum_r = \emptyset$.

(iv) $p_m \leq p$; $p_{mr} \leq p_r \leq p$; $p_i < p$; $p_c < p$; $p_v \leq p$.

(cf. proofs lemma's 2.1 - 2.4 and lemma 3.5).

When an arrow pointing from $\sum_k$ to $\sum_j$, $k,j \in \{m,r,c,i\}$ means that (by def. 2.1) a letter $a \in \sum_k$ may generate a letter $b \in \sum_j$, we easily see that the diagram below holds:

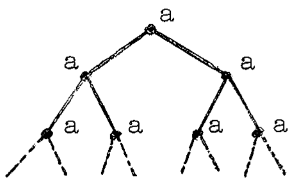


A useful heuristic device in the investigation of aspects of the generic power of a letter $a \in \Sigma$, $S = \langle \Sigma, \delta \rangle$, is the notion of the propagation tree T_a of a; related to the rule tree or derivation tree as encountered in the theory of context free languages.

Def. 2.2. Let $S = \langle \Sigma, \delta \rangle$ be a semi DOL.
The propagation tree T_a of $a \in \Sigma$ is a labeled directed tree of which the labels attached to the nodes are elements of Σ. When we designate the j-th node (from left to right starting with 1) at level k (from top to bottom starting with 0) by (k,j) and $b_i \in \Sigma$ is attached to (k,j), then node (k;j) is connected by edges with nodes (k+1,h),...,(k+1,h+n) labelled $c_{i_1}, \ldots, c_{i_n}$, respectively, iff $b_i \rightarrow c_{i_1} \ldots c_{i_n} \in \delta$. The root of T_a is the single node (0,1) at level 0 labeled with a. A branch is a connected path in T_a. We shall identify the labels with the nodes they label.

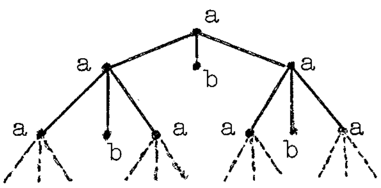
Example 1 $S = \langle \{a\}, \{a \rightarrow aa\} \rangle$

$T_a =$

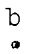


Example 2 $S = \langle \{a,b\}, \{a \rightarrow aba, b \rightarrow \lambda\} \rangle$

$T_a =$



$T_b =$



Remark

For $a \in \sum_m$ the propagation tree T_a eventually terminates, for $a \in \sum_v$ never. $a \in \sum_r$ occurs in T_a appart from the root. $a \in \sum_c$ occurs in T_b of some $b \in \sum_r$. $a \in \sum_i$ does not occur in T_b of some $b \in \sum_r$.

Def. 2.3 A pedigree of b is a sequence $l(b) = b_0, b_1, \ldots, b_t$ such that

$$\forall i \in \{0,1,2,\ldots,t\}[b_{i+1} \in \gamma(\delta(b_i)) \text{ and } b_t = b].$$

Lemma 2.1 Let $S = <\sum,\delta>$ be a semi DOL and $a \in \sum$.

$$a \in \sum_m \text{ iff } \lambda \in \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$$

i.e. iff a derives λ in no more than p_m productions.

Proof $\leftarrow$. If $a \xRightarrow{*(k)} \lambda$, $k \leq p_m$, then $a \in \sum_m$.

$\rightarrow$. Suppose $\lambda \notin \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$

Then $\forall k \in \{0,1,\ldots,p_m\}$ $[\gamma(\delta^k(a)) \cap \sum \neq \emptyset]$.

Let $b \in \gamma(\delta^{p_m}(a)) \cap \sum$, and let $l(b) = b_0, b_1, \ldots, b_{p_m}$,

$b_0 = a$ and $b_{p_m} = b$, be a pedigree of b.

Case 1 $|\gamma(l(b))| = p_m+1$

But $p_m = |\sum_m|$ and therefore

$\gamma(l(b)) \cap \sum_v \neq \emptyset$.

Hence $\bigcup_{0\leq k\leq p_m} \gamma(\delta^k(a)) \cap \sum_v \neq \emptyset$.

i.e. a derives a vital letter and hence is itself vital: $a \notin \sum_m$.

Case 2 $|\gamma(l(b))| \leq p_m$

But $|l(b)| > p_m$ and therefore

$\exists i,j \in \{0,1,\ldots,p_m\}$ $[i < j$ and $b_i = b_j]$.

Hence $\bigcup_{0\leq k\leq p_m} \gamma(\delta^k(a)) \cap \sum_r \neq \emptyset$,

i.e. a derives a recursive letter and hence is vital: $a \notin \sum_m$ □

Remark

The proof is intuitively obvious when we envisualize the propagation tree of a.

Lemma 2.2 Let $S = <\sum,\delta>$ be a semi DOL and $a\in\sum$.

$$a\in\sum_v \text{ iff } \lambda\notin \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$$

Proof $a\notin\sum_m$ iff $\lambda\notin \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$. Hence $a\in\sum\setminus\sum_m = \sum_v$ □

Lemma 2.3 Let $S = <\sum,\delta>$ be a semi DOL and $a\in\sum$.

$$a\in\sum_r \text{ iff } a\in \bigcup_{0<k\leq p_r} \gamma(\delta^k(a))$$

Proof ← . If $a\in \bigcup_{0<k\leq p_r} \gamma(\delta^k(a))$ then $a\in\sum_r$.

→ . If $a\in\sum_r$ then $a\in \bigcup_{k=1}^{\infty} \gamma(\delta^k(a))$

Let h be such that

(1) $a\in\gamma(\delta^h(a))$, and

(2) $a\notin \bigcup_{0<k<h} \gamma(\delta^k(a))$

Let $l(a) = c_0,c_1,c_2,\dots,c_{h-1},c_h$ be a pedigree of the occurence of a in (1), $c_0 = c_h = a$.

Clearly, $\forall 0\leq i\leq h\ [c_i\in\gamma(\delta^h(c_i))]$, and therefore:

(3) $\gamma(l(a)) \subseteq \sum_r$.

Suppose $h > p_r$.

Because of (1), (2) and (3) $|\{c_1,c_2,\dots,c_{h-1}\}| < p_r$.

Therefore $\exists i,j\in\{1,2,\dots,h-1\}[i<j \text{ and } c_i = c_j]$,

and we have

$a\in\gamma(\delta^{h-j}(c_j)) = \gamma(\delta^{h-j}(c_i)) \subseteq \bigcup_{0<k\leq h-j+i} \gamma(\delta^k(a))$, contradicting (2).

Hence $h \leq p_r$ □

Corollary From the proof of lemma 2.3 follows that every $a\in\sum_r$ has a pedigree

$$l(a) = a, c_1, \ldots, c_{k-1}, a \quad \text{such that}$$

$$\gamma(l(a)) \subseteq \sum_r \quad \text{and} \quad |\gamma(l(a)| = k.$$

This means that every occurence of a in some propagation tree is connected by a sequence of recursive letters without repetitions with another occurence of a.

We call such a sequence $C(a) = a, c_1, \ldots, c_{k-1}$ a connecting sequence of a. (Such a connecting sequence is a special case of a dependence path as defined by Rozenberg & Lindenmayer [8]).

Def. 2.4 Let $C(a)$ be a connecting sequence of a.

Then $k = |C(a)|$ is called a period of a.

$$K_a \overset{\text{def}}{\equiv} \{k\in\mathbb{N} \mid k \text{ is a period of } a\}.$$

Lemma 2.4 Let $S = <\sum,\delta>$ be a semi DOL and $a\in\sum$.

$$a\in\sum_c \quad \text{iff} \quad a\in \bigcup_{0<k\le p_c} \gamma(\delta^k(\sum_r)) \cap \sum_v\setminus\sum_r$$

Proof Along lines similar to the proofs of lemma's 2.1-2.3 □

Corollary $a\in\sum_i$ iff $a\notin \bigcup_{0<k\le p_c} \gamma(\delta^k(\sum_r)) \cap \sum_v\setminus\sum_r$ and $a\in\sum_v\setminus\sum_r$.

Theorem 2.5 Let $S = <\sum,\delta>$ be a semi DOL. For every $a\in\sum$ we can effectively determine

(i) Whether a is mortal, vital, or recursive, by examining

$$\bigcup_{0<k\le p} \{\delta^k(a)\} \quad \text{and} \quad \bigcup_{0<k\le p} \gamma(\delta^k(a))$$

(ii) Whether a is recurring vital or initial vital by examining

$$\bigcup_{0<k\le p_v-p_r} \gamma(\delta^k(\sum_r))$$

Proof By lemma's 2.1-2.4 and the corollary. □

Theorem 2.6 We can effectively determine whether a DOL $G = \langle \Sigma, \delta, \sigma \rangle$ is

(i) Quasi-reduced

(ii) Reduced.

Proof Hint (i) $\bigcup_{0 \le k \le p} \gamma(\delta^k(\sigma)) = \Sigma$ iff G is quasi-reduced.

(ii) $\bigcup_{0 \le k \le p} \gamma(\delta^k(\sigma)) = \bigcup_{p-p_r \le k \le 2p-p_r} \gamma(\delta^k(\sigma)) = \Sigma$ iff G is reduced.

2.1 RECURSIVE COMPLEXITY

Consider a language like

$$L(G) = \{a^{2^t} b^{2^t} c^{2^t} | t \geq 0\}$$

Clearly, a pattern like

aa...a bb...b cc...c

can only be produced by another such pattern if

$$\delta(a) \in \{a\}^*$$

$$\delta(b) \in \{b\}^*$$

$$\delta(c) \in \{c\}^*$$

And we easily see that

$$G = <\{a,b,c\},\{a \to aa,\ b \to bb,\ c \to cc\},\ abc>.$$

We can investigate DOL-languages, as sets of patterns, and the semi DOL's which give rise to them, by studying relations between recursive letters and developing a notion of recursive complexity. (This will be the subject of a subsequent report).

Def. 2.5 Let $S = <\sum,\delta>$ be a semi DOL. The Recursive Complexity Structure of S is a partially ordered set $RCS(S) = (\sum_{r/\sim},\leq)$ such that

(i) $\sum_r \subseteq \sum$ is the set of recursive letters.

(ii) Let $a,b \in \sum_r$. $a \leq b$ iff $a \in \gamma(\delta^*(b))$.

(iii) Let $a,b \in \sum_r$. $a \sim b$ iff $a \leq b$ and $b \leq a$.

Clearly, the relation $\sim$ is an equivalence relation and induces a partition on $\sum_r$ in blocks $[a]_i$, i.e.

$\sum_{r/\sim} = \{[a]_i\}$, and we define $[a]_i \leq [a]_j$ iff

$c \leq b$ for some $c \in [a]_i$ and $b \in [a]_j$.

Lemma 2.7 $[a] \supseteq \cup \{\gamma(C(a)) | C(a)$ is a connecting sequence of $a\}$.

Proof $\gamma(C(a)) \subseteq \sum_r$. (corollary lemma 2.3).

Let $b \in \gamma(C(a))$. Then $b \in \bigcup_{0 \leq k \leq p_r} \gamma(\delta^k(a))$ and $a \in \bigcup_{0 \leq k \leq p_r} \gamma(\delta^k(b))$.

Hence $b \leq a$ and $a \leq b$ □

That the converse of this lemma is not true follows from the counter example

$G = \langle\{a,b,c\},\{a \rightarrow ab,\ b \rightarrow ac,\ c \rightarrow b\},\ a\rangle$

$\cup\{\gamma(C(a))\} = \{a\} \cup \{a,b\} = \{a,b\}$.

But $c \in \gamma(\delta^2(a))$ and $a \in \gamma(\delta^2(c))$ and $c \in \sum_r$.
Hence $c \in [a]$ and $c \notin \cup\{\gamma(C(a))\}$.

Lemma 2.8 Let $S = \langle\sum,\delta\rangle$ be a semi DOL. We can effectively determine RCS(S) by examining $\bigcup_{0 \leq k \leq p} \gamma(\delta^k(a))$ for all $a \in \sum$.

Proof We prove that for $a,b \in \sum_r$ ($\sum_r$ determined by theorem 2.5):

$a \leq b$ iff $a \in \bigcup_{0 \leq k \leq p} \gamma(\delta^k(b))$.

$\leftarrow$. If $a \in \bigcup_{0 \leq k \leq p} \gamma(\delta^k(b))$ then $a \leq b$.

$\rightarrow$. If $a \leq b$ then there is a pedigree $l(a) = b,c_1,\ldots,c_{h-1},a$.

Suppose $h > p$. Clearly $l(a)$ contains a repetition of letters and there is an $l'(a) = b,d_1,\ldots,d_{h'-1},a$ with $h' < h$.
By iteration of this argument there is an

$l'(a) = b,e_1,\ldots,e_{k-1},a$ such that $k \leq p$.

Hence $a \in \bigcup_{0 \leq k \leq p} \gamma(\delta^k(b))$ □

In section 3.2 we shall prove that if $L(G)$ is finite then for all $a,b \in \sum_r$ if $a \leq b$ then $a \sim b$, i.e. the RCS consists of incomparable classes.

3 DOL LANGUAGES AND A WORD PROBLEM

Consider the following "word problem", given a semi DOL $S = <\sum,\delta>$ and two words ω_1 and ω_2 over $\sum$. Does there exist an algorithm which decides whether or not $\omega_1 \overset{*}{\Longrightarrow} \omega_2$. In another version the problem is posed in [3] and called the membership question for DOL's: given a DOL $G = <\sum,\delta,\sigma>$ and a word ω_τ is it decidable whether $\omega_\tau \in L(G)$. Doucet [op. cit.] proves, independent and preliminary to the research reported here, that this question is decidable *), essentially by showing that:

(i) It is decidable whether L(G) is finite or infinite.

(ii) If L(G) is infinite then $|\delta^p(\delta^i(\sigma))| > |\delta^i(\sigma)|$ and therefore we can decide the question by generating finitely many successive strings starting with σ.

(iii) If L(G) is finite, the question is decided by writing out the whole of L(G) where

$$|L(G)| \leq p^{(p-1)MK^p+M}$$

where $p = |\sum|$

$$K = \max_{a\in\sum} \{|\alpha| \;\; |a \rightarrow \alpha\}$$

$$M = \max_{\omega\in L(G)} \{|\omega|_v \;\; | \;\; |\omega|_v \text{ is the number of occurences of vital letters in } \omega\}$$

(iii) suffers the same defect most decision procedures do, viz. it is not <u>feasible</u>. The a priori bound on the computation length is not proportionate to our present (and future) means of computation. Even a very conservative estimate with p = 5, K = 2 and M = 5 gives us

$$|L(G)| \leq 5^{4\times5\times2^5+5} = 5^{645},$$

*) The decidability of this word problem also is a corollary of the inclusion of the DOL Languages in the context sensitive languages [7].

which, for all practical purposes means the same as no a priori bound. There is a profound difference between most mathematical decision procedures and feasible algorithms which can be executed on a computer and answer reasonable questions in a reasonable time. Nobody is satisfied by a Turing machine computation of $10^{10} + 10^{10}$ which takes $\mathcal{O}(10^{20})$ steps (when the TM has a one letter alphabet).

In section 3.2 we devise a feasible heuristic algorithm which has been implemented in an ALGOL program and delivers answers (to reasonable questions) in a matter of seconds.

3.1 FINITE AND INFINITE DOL LANGUAGES

We investigate some properties of DOL's which also form prerequisites of the proposed algorithm.

Let $|\omega|_k$ denote the number of occurences in ω of letters $a\in\sum_k$ where $k\in\{m,v,r,c,i,mr\}$.

<u>lemma 3.1</u> Let $S = <\sum,\delta>$ be a semi DOL and $\omega_1,\omega_2\in\sum^*$.

If $|\omega_1|_v > |\omega_2|_v$ then $\omega_1 \overset{*}{\not\Longrightarrow} \omega_2$.

<u>Proof</u> Since $\gamma(\delta(a)) \cap \sum_v \neq \emptyset$ for $a\in\sum_v$.

$\forall t\in\mathbb{N}$ $[$if $|\omega_1|_v > |\omega_2|_v$ then $|\delta^t(\omega_1)|_v > |\omega_2|_v]$ □

It is easy to see that initial vital letters can only occur in $\sigma, \delta(\sigma),\ldots, \delta^{p_i-1}(\sigma)$; and recurring vital letters not derived from recursive letters can only occur in

$\sigma, \delta(\sigma),\ldots,\delta^{p_i+p_c-1}(\sigma)$.

<u>Lemma 3.2</u> Let $G = <\sum,\delta,\sigma>$ be a DOL.

$\forall t \geq p_i + p_c$ $[$if $b\in\gamma(\delta^t(\sigma)) \cap \sum_v$ then $\gamma(l(b)) \cap \sum_r \neq \emptyset]$

where $l(b) = b_0,b_1,\ldots,b_{t-1},b_t$, such that $b_0\in\gamma(\sigma)$, $b_t = b$.

(Every vital letter in $\delta^t(\sigma)$, $t \geq p_i + p_c$, has been derived from a recursive letter in $\delta^{t'}(\sigma)$, $0 \leq t' < p_i + p_c$).

<u>Proof</u> If $b\in\sum_v$ then $\gamma(l(b)) \subseteq \sum_v$.

Suppose the lemma is not true, i.e. $\gamma(l(b)) \subseteq \sum_i \cup \sum_c$.

But then $|l(b)| = t+1 > p_i + p_c$ while $|\gamma(l(b))| \leq p_i + p_c$.

Hence $l(b)$ contains a repetition of a letter and $\gamma(l(b)) \cap \sum_r \neq \emptyset$, which contradicts the assumption. □

<u>Corollary</u> For all $t \geq p_i + p_c$ holds:

if $b \epsilon \gamma(\delta^t(\sigma)) \cap \sum_v$ then there is a $c \epsilon \gamma(\delta^t(\sigma)) \cap \sum_r$ such that $b \epsilon \gamma(\delta^*(c))$.

Or: $\gamma(\delta^t(\sigma)) \cap \sum_v \subseteq \gamma(\delta^*(\gamma(\delta^t(\sigma)) \cap \sum_r))$.

Hint: by lemma 3.2, repeated application of the corollary of lemma 2.3 and by lemma 2.7.

<u>Def. 3.1</u> A DOL $G' = <\sum', \delta', \sigma'>$ is the <u>positive</u> <u>k-displacement</u> of $G = <\sum, \delta, \sigma>$ if $\sigma' = \delta^k(\sigma)$ and $\sum' \subseteq \sum$, $\delta' \subseteq \delta$ such that G' is quasi-reduced. G is a <u>negative k-displacement</u> of G .

<u>lemma 3.3</u> The positive p-p_r displacement $G' = <\sum', \delta', \sigma'>$ of $G = <\sum, \delta, \sigma>$ is reduced.

<u>Proof</u> By the corollary of lemma 3.2 and by lemma 2.1 □

<u>lemma 3.4</u> Let $S = <\sum, \delta>$ be a semi DOL. If $a \epsilon \sum_{mr}$ then $[a] = \gamma(C(a)) \subseteq \sum_{mr}$, where $C(a)$ is the unique connecting sequence of a. Moreover, $\delta^*(a) \subseteq \sum_m^* [a] \sum_m^*$.

<u>Proof</u> Let $C(a) = b_0, b_1, \ldots, b_{k-1}$ be a connecting sequence of a, $b_0 = a$, and let $b_k = a$.

Suppose $c \epsilon \gamma(\delta^*(a)) \cap \sum_v \backslash \gamma(C(a))$.

Then $c \epsilon \gamma(\delta^h(a))$ and $b_p \epsilon \gamma(\delta^h(a))$ for some h and $p \equiv h \bmod (k)$.

Therefore $|\delta^{h+k-p}(a)|_v = 2$. But from def. 2.1 follows $\delta^{h+k-p}(a) = \eta a \xi \epsilon \sum_m^* \sum_{mr} \sum_m^*$, which contradicts the assumption.

Hence $C(a)$ is the unique connecting sequence of a; $\gamma(C(a)) \supseteq [a]$; and $\delta^*(a) \subseteq \sum_m^* \gamma(C(a)) \sum_m^* \subseteq \sum_m^* \sum_{mr} \sum_m^*$.

By lemma 2.7 also $\gamma(C(a)) \subseteq [a]$ and therefore $\gamma(C(a)) = [a]$ □

Lemma 3.5 Let $S = <\sum,\delta>$ be a semi DOL.

(i) If $a\epsilon\sum_r$ then $a\epsilon \bigcup_{0<k\leq p_r} \gamma(\delta^k(a)) \cap \sum_r$

(ii) If $a\epsilon\sum_c$ then $\bigcup_{0<k\leq p_c} \gamma(\delta^k(a)) \cap \sum_r \neq \emptyset$

(iii) If $a\epsilon\sum_i$ then $\bigcup_{0<k\leq p_i+p_c} \gamma(\delta^k(a)) \cap \sum_r \neq \emptyset$

Proof (i) follows from lemma 2.3.

(ii) Let $a\epsilon\sum_c$. Then there is a $b\epsilon\sum_v$ such that

$$b\epsilon\gamma(\delta^{p_c}(a)) \cap \sum_v.$$

Let $l(b) = b_0,b_1,\ldots,b_{p_c}$, $b_0 = a$, $b_{p_c} = b$, be a pedigree of b.

Since $a\epsilon\sum_c$, $\gamma(l(b)) \subseteq \sum_c \cup \sum_r$.

Suppose $\gamma(l(b)) \subseteq \sum_c$; then $|\gamma(l(b))| \leq p_c$.

But $|l(b)| > p_c$ and hence there is a repetition of a letter in $l(b)$ which contradicts the assumption.

Hence $\gamma(l(b)) \cap \sum_r \neq \emptyset$ which gives us lemma 3.5 (ii).

(iii) Analogous to (ii) with $\sum_c \cup \sum_i$ and p_c+p_i substituted for $\sum_c$ and p_c □

Remark Lemma 3.5 tells us that every vital letter derives a repetition of a recursive letter within p_v steps.

Lemma 3.6 Let $S = <\sum,\delta>$ be a semi DOL.

(i) If $a\epsilon\sum_r\backslash\sum_{mr}$, i.e. $|\delta^k(a)|_v = x > 1$ where $k = \min K_a$,

then $\forall n\epsilon\mathbb{N}\ [|\delta^{nk}(a)|_v > nx-n]$

(ii) If $a\epsilon\sum_{mr}$, i.e. $|\delta^k(a)|_v = 1$ where $K_a = \{k\}$,

then $\forall t\epsilon\mathbb{N}\ [|\delta^t(a)|_v = |\delta^t(a)|_r = 1]$

Proof (i) By induction on n.

$n = 0$. $|\delta^0(a)|_v = |a|_v = 1 > 0$.

Suppose the assumption is true for n.

$$|\delta^{(n+1)k}(a)|_v = |\delta^k(\delta^{nk}(a))|_v \geq$$

$$|\delta^{nk}(a)|_v - 1 + |\delta^k(a)|_v >$$

$$n\,x - n - 1 + x = (n+1)x - (n+1)$$

(ii) By lemma 3.4 □

Theorem 3.7 L(G) is finite iff $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \subseteq \sum_{mr}$.

Proof →. L(G) is finite.

Suppose $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v\backslash\sum_{mr} \neq \emptyset$.

Let $a_i \in \gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v\backslash\sum_{mr}$.

Case 1 $a_i \in \sum_r\backslash\sum_{mr}$, i.e. $|\delta^{k_i}(a_i)|_v = x_i > 1$, where $k_i = \min K_{a_i}$.

$|\delta^{n*k_i}(a_i)|_v > nx_i - n$ (lemma 3.6),

and for all $b \in \mathbb{N}$

$$|\delta^{b*k_i}(a_i)| \geq |\delta^{b*k_i}(a_i)|_v > b\,x_i - b \geq b$$

Hence L(G) is infinite: contradiction.

Case 2 $a_i \in \sum_c$. Then by the corollary of lemma 3.2 there is a $b \in \gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_r$ such that $a_i \in \gamma(\delta^*(b))$.

By lemma 3.4 $b \in \sum_r\backslash\sum_{mr}$; and by case 1 L(G) is infinite which contradicts the assumption.

Case 3 $a \in \sum_i$. By lemma 3.2 $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_i = \emptyset$.

From case 1 - case 3 follows $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \subseteq \sum_{mr}$.

←. $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \subseteq \sum_{mr}$.

Let $|\delta^{p_i+p_c}(\sigma)|_{mr} = m$. By lemma 3.4

$$\forall t \geq 0\ [|\delta^t(\delta^{p_i+p_c}(\sigma))|_v = |\delta^t(\delta^{p_i+p_c}(\sigma))|_r = m].$$

Denote the i-th occurence of a monorecursive letter in $\delta^t(\sigma)$, $t \geq p_i+p_c$, by $a_i(t)$ and its period by k_i.

Since

(1) $\forall t,t' \geq p_i+p_c \quad \forall i\in\{1,2,\ldots,m\}$ [if $t \equiv t' \bmod(k_i)$ then

$$a_i(t) = a_i(t')]$$

we have

(2) $a_1(t)a_2(t)\ldots a_m(t) = a_1(t+u)a_2(t+u)\ldots a_m(t+u)$ where

$u = \text{l.c.m.}(k_1,k_2,\ldots,k_m)$ and $t \geq p_i+p_c$.

By (2) and lemma 2.1, for all $t \geq p_i+p_c$ and all $\eta_1,\eta_2,\ldots,\eta_{m+1},\xi_1,\xi_2,\ldots,\xi_{m+1}\in\sum_m^*$ holds:

$$\delta^{p_m}(\eta_1\ a_1(t)\ \eta_2\ a_2(t)\ \eta_3\ldots\eta_m\ a_m(t)\ \eta_{m+1}) =$$
$$\delta^{p_m}(a_1(t)\ a_2(t)\ \ldots\ a_m(t)) =$$
$$\delta^{p_m}(a_1(t+u)\ a_2(t+u)\ \ldots\ a_m(t+u)) =$$
$$\delta^{p_m}(\xi_1\ a_1(t+u)\ \xi_2\ a_2(t+u)\ \xi_3\ \ldots\ \xi_m\ a_m(t+u)\ \xi_{m+1})$$

In particular:

$\delta^{p_i+p_c+p_m}(\sigma) = \delta^{p_i+p_c+p_m+u}(\sigma)$ and hence

$|L(G)| \leq p-p_r + \text{l.c.m.}(k_1,k_2,\ldots,k_m) \qquad \Box$

<u>Corollary</u> Let G be quasi-reduced. L(G) is finite iff $\sum_v = \sum_i \cup \sum_{mr}$.

$\leftarrow$. If $\sum_v = \sum_i \cup \sum_{mr}$, by the previous arguments L(G) is finite.

$\rightarrow$. Suppose $\overset{\infty}{\underset{k=0}{\cup}}\gamma(\delta^k(\sigma)) \cap (\sum_c \cup \ \sum_r\backslash\sum_{mr}) \neq \emptyset$.

<u>case 1</u> For some $t \geq 0 \quad b\in\gamma(\delta^t(\sigma)) \cap \sum_r\backslash\sum_{mr}$. By the previous arguments L(G) is infinite.

<u>case 2</u> For some $t \geq 0 \quad b\in\gamma(\delta^t(\sigma)) \cap \sum_c$. Since G is quasi-reduced, by the def. of $\sum_c$ and lemma 3.4

$\gamma(\delta^{t'}(\sigma)) \cap \sum_r \setminus \sum_{mr} \neq \emptyset$ for some t', which gives us case 1.

Hence if L(G) is finite the assumption is false, i.e. $\sum_c \cup \sum_r \setminus \sum_{mr} = \emptyset$ □

Clearly, if $a,b \in \sum_{mr}$ and $a \leq b$ then $a \sim b$.
Hence, if L(G) is finite then RCS(S) consists of pairwise incomparable classes of monorecursive letters. The converse is trivially true. Therefore, the family of finite DOL-languages is contained in the family of DOL-languages of which the RCS consists of pairwise incomparable elements.

$G = \langle\{a,b\},\{a \to aa,\ b \to b\},\ ab\rangle$ yields
$L(G) = \{a^{2^t} b \mid t \geq 0\}$ and the RCS consists of pairwise incomparable elements. Hence the containment is proper.

Lemma 3.8 L(G) is infinite iff $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \setminus \sum_{mr} \neq \emptyset$

Proof By theorem 3.7 □

Theorem 3.9 L(G) is finite iff $|\delta^{p_i+2p_c+p_r}(\sigma)|_v = |\delta^{p_i+p_c}(\sigma)|_v$.

Proof →. L(G) is finite. Since $\sum_v \cap \gamma(\delta^{p_i+p_c}(\sigma)) \subseteq \sum_{mr}$

$$|\delta^{p_i+2p_c+p_r}(\sigma)|_v = |\delta^{p_i+p_c}(\sigma)|_v$$

←. $|\delta^{p_i+2p_c+p_r}(\sigma)|_v = |\delta^{p_i+p_c}(\sigma)|_v$.

Clearly,
$\forall a_i \in \sum_v$ [if $a_i \in \delta^{p_i+p_c}(\sigma)$ then $|\delta^{p_c+p_r}(a_i)|_v = 1$]

case 1 $a_i \in \sum_r \setminus \sum_{mr}$.

$|\delta^{p_c+p_r}(a_i)|_v \geq |\delta^{k_i}(a_i)|_v > 1$ where $k_i = \min K_{a_i}$:

contradiction.

case 2 $a_i \in \sum_c$. By the corollary of lemma 3.2 and lemma 3.4 this reduces to case 1.

case 3 $a_i \epsilon \sum_i$: contrary to lemma 3.2.

Since cases 1 - 3 cannot occur, $a_i \epsilon \sum_{mr}$ and the proof follows by theorem 3.7 □

Corollary L(G) is infinite iff $|\delta^{p_i+2p_c+p_r}(\sigma)|_v > |\delta^{p_i+p_c}(\sigma)|_v$.

Remark Up to now we have given several criteria for determining whether L(G) is finite or infinite.

(i) If $\sum_c \cup \sum_r \setminus \sum_{mr} \neq \emptyset$ and G is quasi-reduced, then L(G) is infinite.

(ii) If $\gamma(\delta^t(\sigma)) \cap \sum_v \subseteq \sum_{mr}$, $t \geq p_i+p_c$, then L(G) is finite.

(iii) If $|\delta^t(\sigma)|_v = |\delta^{t'}(\sigma)|_v$, with $t-t' \geq p_r+p_c$ and $t' \geq p_i+p_c$, then L(G) is finite.

Relevant to the solution of the word problem are the following observations.

Let L(G) be infinite.

$$\exists a_i \epsilon \gamma(\delta^{p_i+p_c}(\sigma))\ \forall n \epsilon \mathbb{N}\ [|\delta^{n*k_i}(a_i)|_v > n]$$

If we take $n = |\omega_\tau|_v$, then after $p_i+p_c+n*k_i$ productions, surely, we know whether $\sigma \overset{*}{\Longrightarrow} \omega_\tau$.

(N.B. clearly, n is a very poor lower bound on $|\delta^{n*k_i}(a_i)|_v$).

Let L(G) be finite.

$$\exists x,u \epsilon \mathbb{N}\ [\delta^x(\sigma) = \delta^{x+u}(\sigma)],\ \text{i.e.}$$

$$L(G) = \{\delta^t(\sigma) | 0 \leq t < x+u\}.$$

If $\omega_\tau \notin L(G)$ then $\sigma \overset{*}{\not\Longrightarrow} \omega_\tau$.

Lemma 3.10 Let $S = <\sum,\delta>$ be a semi DOL and $a \epsilon \sum_{mr}$.

$$\delta^{p_m+t}(a) = \delta^{p_m+t'}(a) \text{ for } t' \equiv t \bmod (k)$$

$$\delta^{p_m+t}(a) \neq \delta^{p_m+t'}(a) \text{ for } t' \not\equiv t \bmod (k)$$

where $t,t' \epsilon \mathbb{N}$ and k is the period of a.

Proof Let $C(a) = b_0,b_1,\ldots,b_{k-1}$, $b_o=a$, be the unique connecting sequence of a.

(1) $\gamma(C(a)) \subseteq \sum_{mr}$ (lemma 3.4)

(2) $\delta^t(a) \in \sum_m^* \sum_{mr} \sum_m^*$ (lemma 3.4)

(3) In C(a), $b_i \neq b_j$ for $0 \leq i < j < k$ (by definition).

(4) $\delta^t(a) = \eta b_{t \bmod(k)} \xi \in \sum_m^* \sum_{mr} \sum_m^*$ (by definition of C(a) and (1) and (2)).

By (1)-(4), $\delta^t(a) \neq \delta^{t'}(a)$ for $t \not\equiv t' \bmod(k)$, $t,t' \geq 0$. More in particular:

(5) $\delta^{p_m+t}(a) \neq \delta^{p_m+t'}(a)$ for $t \not\equiv t' \bmod(k)$, $t,t' \geq 0$.

Since $\delta^{p_m}(\eta) = \lambda$ for all $\eta \in \sum_m^*$ and (4) we have:

$$\delta^{p_m}(\delta^t(a)) = \delta^{p_m}(\eta\ b_{t \bmod(k)}\ \xi) =$$

$$\delta^{p_m}(\eta'\ b_{t' \bmod(k)}\ \xi') = \delta^{p_m}(\delta^{t'}(a))$$

for $t \equiv t' \bmod(k)$, $t,t' \geq 0$, and $\delta^{t'}(a) = \eta'\ b_{t' \bmod(k)}\ \xi'$, $\eta',\xi' \in \sum_m^*$.

Hence

(6) $\delta^{p_m+t}(a) = \delta^{p_m+t'}(a)$ for $t \equiv t' \bmod(k)$, $t,t' \geq 0$.

From (5) and (6) the lemma follows. □

Lemma 3.11 Let $S = \langle\Sigma,\delta\rangle$ be a semi DOL and $a \in \sum_{mr}$,

$$\delta^{p_m+t}(a) \neq \delta^{p_m+t'}(a)\mu \quad \text{with } \mu \in \sum^+$$

for all $t,t' \in \mathbb{N}$.

Proof By (1)-(4) and (6) of the previous proof. □

3.2. THE WORD PROBLEM

The derived theorems yield an algorithm to decide the word problem. According to the proof of lemma 3.2 all vital $a_i \in \gamma(\delta^{p_i+p_c}(\sigma))$ are derived from previous occurrences of recursive letters and will recur again.

By comparing $|\delta^{p_i+p_c}(\sigma)|_v$ and $|\delta^{p_i+2p_c+p_r}(\sigma)|_v$ we know whether L(G) is finite or infinite (theorem 3.9). If L(G) is infinite we generate along and compare $\delta^t(\sigma)$ and ω_τ until either $\delta^t(\sigma) = \omega_\tau$ or $|\delta^t(\sigma)|_v > |\omega_\tau|_v$.

If L(G) is finite all vital $a_i \in \delta^{p_i+p_c}(\sigma)$ are monorecursive (theorem 3.7).

Let $\delta^{p_i+p_c}(\sigma) = \eta_1\, a_1\, \eta_2\, a_2 \dots \eta_m\, a_m\, \eta_{m+1}$ with $\eta_1,\dots,\eta_{m+1} \in \sum_m^*$ and $a_1,\dots,a_m \in \sum_{mr}$.

Since $\delta^{p_m}(\eta) = \lambda$ for all $\eta \in \sum_m^*$ we have

$$(1)\quad \delta^{p_m}(\delta^{p_i+p_c}(\sigma)) = \delta^{p_m}(\eta_1)\, \delta^{p_m}(a_1) \dots \delta^{p_m}(a_m)\, \delta^{p_m}(\eta_{m+1}) = \delta^{p_m}(a_1)\, \delta^{p_m}(a_2) \dots \delta^{p_m}(a_m).$$

By lemma 3.10 (let k_i be the period of a_i):

$$(2)\quad \forall a_i \in \textstyle\sum_{mr}\ \forall t,t' \geq 0\ [\text{if } t \equiv t' \bmod(k_i) \text{ then } \delta^{p_m+t}(a_i) = \delta^{p_m+t'}(a_i)],$$

and by lemma 3.11

$$(3)\quad \forall a_i \in \textstyle\sum_{mr}\ \forall t,t' \geq 0\ \forall \mu \in \sum^+\ [\delta^{p_m+t}(a_i) \neq \delta^{p_m+t'}(a_i)\mu].$$

(1)-(3) reduces the problem, for finite L(G), to the following: do there exist $t_1, t_2, \dots, t_m \in \mathbb{N}$ such that

$$\delta^{p_m+t_1}(a_1)\, \delta^{p_m+t_2}(a_2) \dots \delta^{p_m+t_m}(a_m) = \omega_\tau$$

and if so, does there exist a

$$u \equiv t_i \bmod(k_i) \qquad 1 \leq i \leq m \qquad \text{(cf. (2))}.$$

If u exists then $\sigma \xrightarrow{*(p-p_r+u)} \omega_\tau$. Because of (3) $t_1,t_2,\ldots,t_m$ are unique.

Theorem 3.14. (Generalized Chinese Remainder Theorem, cf. Dickson [2]; also Knuth [5, p. 256]).

Let $k_1,\ldots,k_m$ be positive integers and let $t_1,\ldots,t_m$ be any integers. There is exactly one integer u which satisfies the conditions

$$0 \leq u < \text{l.c.m.}(k_1,\ldots,k_m)$$

$$u \equiv t_i \bmod(k_i) \qquad (1 \leq i \leq m)$$

iff $$t_i \equiv t_j \bmod (\text{g.c.d}\ (k_i,k_j)) \qquad (1 \leq i < j \leq m).$$

Corollary. A solution for $u \equiv t_i \bmod(k_i)$ $(1\leq i\leq m)$ yields $u < \text{l.c.m.}\ (k_1,\ldots,k_m)$, when m denotes the number of monorecursive letters in $\delta^{p_i+p_c}(\sigma)$.

Clearly, $k_i = |[a_i]|$ and if $a_j \in [a_i]$ then $k_j = k_i$ (cf. lemma 3.4)

Hence $u < \text{l.c.m}\ (|[a]_1|,\ldots,|[a]_q|)$ where

$$\{[a]_1,\ldots,[a]_q\} = \Sigma_r/\sim \qquad \text{(G is quasi-reduced)}.$$

We conclude that, if we know that L(G) is finite, by examining the propagations of the different letters in σ for maximal $p_i + p_c + p_m + p_r = p$ steps we know whether or not

$$\omega_\tau = \delta^{p-p_r+u}(\sigma)$$

where $u < \text{l.c.m}\ (k_1,\ldots,k_m) = \text{l.c.m.}\ (|[a]_1|,\ldots,|[a]_q|)$.

If we have to decide first whether L(G) is finite or not we need $p + p_c + p_r$ generations.

3.2.1. THE ALGORITHM

We present the algorithm written in pseudo ALGOL so as to make it at once more unambiguous and comprehensible.

```
¢ Algorithm solves the word problem for DOL's ¢

begin  procedure compare ($\sigma,\omega_\tau$);
          begin if $|\sigma|_v > |\omega_\tau|_v$ then
                begin print  (⌈no solution⌉); goto exit end
                else if $\sigma = \omega_\tau$ then
                begin print (⌈solution found⌉); goto exit end;
          end;

 phase 0:  ¢ classify all $a_i \in \sum$; by examining $\bigcup_{0<k\leq p} \gamma(\delta^k(a_i))$ and
           $\bigcup_{0<k\leq p} \delta^k(a_i)$, whether they belong to $\sum_m$, $\sum_r$, or $\sum_c \cup \sum_i$.
           If $a_i \in \sum_r$ then determine its smallest period $k_i$ ¢

 phase 1:  compare ($\sigma,\omega_\tau$);

 phase 2:  for i:= 1 step 1 until p do
           begin $\sigma := \delta(\sigma)$; compare ($\sigma,\omega_\tau$)
           end; vital b:= $|\sigma|_v$;

 phase 3:  for i:= 1 step 1 until $p_v$ do
           begin $\sigma := \delta(\sigma)$; compare ($\sigma,\omega_\tau$)
           end; vital end:= $|\sigma|_v$;
           if vital end = vital b then goto phase 5;

 phase 4:  ¢ L(G) is infinite ¢
           $\sigma := \delta(\sigma)$; compare ($\sigma,\omega_\tau$);
           goto phase 4;

 phase 5:  ¢ L(G) is finite ¢
           i:= 1;
   next:   for j:= 1 step 1 until $k_i$ do
           ¢ $k_i$ is the period of $a_i$ ¢
```

```
            if $\omega_\tau = \delta^j(\delta^{p_m}(a_i))\eta$ then
            begin $t_i$ := j;   $\omega_\tau$ := $\eta$;   i:= i+1;
                  if $\omega_\tau \neq \lambda \wedge i \leq m$ then goto next else
                  if $\omega_\tau = \lambda \wedge i = m+1$ then goto phase 6
            end;
            print (⌈no solution⌉); goto exit;

phase 6:    for i:= 1 step 1 until m-1 do
            begin for j:= i+1 step 1 until m do
                  if $t_i \not\equiv t_j \bmod (\text{g.c.d.}\ (k_i,k_j))$
                  then begin print (⌈no solution⌉); goto exit end
            end;
            print (⌈solution found⌉);

exit:

end
```

Def. 3.2. Let $G = \langle\Sigma,\delta,\sigma\rangle$ be a DOL. The growth function $F: \mathbb{N} \to \mathbb{N}$ is defined by $F(t) = |\delta^t(\sigma)|$. [Szilard, 1971; paper by Salomaa & Paz in preparation].

The following speed up of the algorithm, especially when L(G) is infinite, was suggested by A. Paz.

Use the growth function of the DOL to determine the indexes of the (finitely many if L(G) is infinite) words in $\xi(G)$ which have a length equal to $|\omega_\tau|$. Then approach these words rapidly by generating

$$\begin{aligned}
\forall a \in \Sigma : \quad \delta^2(a) &= b_1 \dots b_m \\
\delta^4(a) &= \delta^2(b_1) \dots \delta^2(b_m) \\
&\vdots \\
\delta^{2^t}(a) &= c_1 \dots c_n \\
\delta^{2^{t+1}}(a) &= \delta^{2^t}(c_1) \dots \delta^{2^t}(c_n)
\end{aligned}$$

In this fashion we approach with exponential speed a word of large index without having to generate the intermediate words.

3.3. THE SIZE OF FINITE DOL LANGUAGES

Lemma 3.15. Let L(G) be a finite DOL language generated by $G = <\sum,\delta,\sigma>$.

$$\forall j \geq p_i+p_c \ [\max_{i\in\mathbb{N}} \{|\delta^i(\sigma)|_v\} = |\delta^j(\sigma)|_v = |\delta^j(\sigma)|_r].$$

Proof By lemma 3.4 and theorem 3.7. □

Theorem 3.16. Let L(G) be a finite DOL language generated by $G = <\sum,\delta,\sigma>$.

$$u \leq L(G) \leq u+p-p_r$$

where $u = \text{l.c.m.}\ (k_1,k_2,\dots,k_m)$ and $k_1,k_2,\dots,k_m$ are the periods of the monorecursive $a_1,a_2,\dots,a_m$ in $\delta^{p_i+p_c}(\sigma)$.

Proof Denote the i-th occurrence of a monorecursive letter in $\delta^t(\sigma)$, $t \geq p_i+p_c$, by $a_i(t)$. Let $|\delta^t(\sigma)|_{mr} = m$ and k_i be the period of a_i.

$$(1) \quad \forall t,t' \geq p_i+p_c \quad \forall i \in \{1,2,\dots,m\} \ [\text{if } t \equiv t' \bmod(k_i) \text{ then } a_i(t) = a_i(t') \text{ else } a_i(t) \neq a_i(t')].$$

Therefore:

$$(2) \quad \forall t \geq p_i+p_c \quad \forall j < \text{l.c.m.}(k_1,\dots,k_m) \ [a_1(t)\ a_2(t) \dots a_m(t) \neq a_1(t+j)\ a_2(t+j) \dots a_m(t+j)]$$

Hence

$$(3) \quad |L(G)| \geq \text{l.c.m.}\ (k_1,k_2,\dots,k_m) = u.$$

By the proof of theorem 3.7

$$(4) \quad |L(G)| \leq p-p_r+\text{l.c.m.}\ (k_1,k_2,\dots,k_m).$$

From (3) and (4) the lemma follows. □

Corollary. Since $k_i = |[a_i]|$ for $a_i \in \sum_{mr}$,

$$u = \text{l.c.m.}\ (k_1,k_2,\ldots,k_m) = \text{l.c.m.}\ (|[a]_1|,|[a]_2|,\ldots,|[a]_q|)$$

where $\{[a]_1,[a]_2,\ldots,[a]_q\} = \sum_{r/\sim}$ if G is quasi-reduced.

Remark. $$\begin{aligned}|L(G)| &\leq p - p_r + \text{l.c.m.}\ (k_1,\ldots,k_s)\\ &\leq p - p_r + p_r^s\\ &\leq p(1+p^{n-1})\\ &\leq p(1+p^{m-1})\end{aligned}$$

where s is the number of monorecursive letters with different periods in $\delta^{p_i+p_c}(\sigma)$ (in $\sum_{mr}$ if G is quasi-reduced), n is the number of different monorecursive letters in $\delta^{p_i+p_c}(\sigma)$, and m is the number of occurrences of monorecursive letters in $\delta^{p_i+p_c}(\sigma)$.

For the numeric example given in the introduction to section 3 we find: (m=5, p=5, K=2)

$$|L(G)| \leq 5(1+5^4) = 3130$$

which upper bound may be minimized by taking the different periods of $a_1,a_2,\ldots,a_m$ into account.

The reduction on the size of upper bound on $|L(G)|$ we have reached:

$$\frac{p^{(p-1)MK^p+M}}{p+p^m} = \frac{p^{(p-1)MK^p+M}}{p+p^M} \approx \mathcal{O}(p^{(p-1)MK^p}).$$

Strangely enough, it appears that K, i.e. the max. length of $\delta(a)$, has no influence on the size of L(G).

We are now in the position to tackle the following problems. Let $G = \langle\sum,\delta,\sigma\rangle$ be a DOL

(i) What is the minimal size of $\sum$ such that $|L(G)| = n$

(ii) What is the minimal size of $\sum$ such that $|L(G)| \geq n$

(iii) What is the maximal size of $L(G)$ when $|\sum| = n$.

(iv) What is the maximal size of $L(G)$ when $|\sum| \leq n$.

Let $$f_1, f_2, f_3, f_4 \colon \mathbb{N} \to \mathbb{N}$$

be functions which map n onto the asked sizes in (i)-(iv).

$$f_1(n) = \min\{ \sum_{i=1}^{m} k_i+d \mid m \in \mathbb{N};\ k_1,\ldots,k_m \in \mathbb{N} \text{ are pairwise prime; } d \in \mathbb{N} \text{ and } \prod_{i=1}^{m} k_i+d = n\}.$$

$$f_2(n) = \min\{ \sum_{i=1}^{m} k_i+d \mid m \in \mathbb{N};\ k_1,\ldots,k_m \in \mathbb{N} \text{ are pairwise prime; } d \in \mathbb{N} \text{ and } \prod_{i=1}^{m} k_i+d \geq n\}.$$

$$f_3(n) = \max\{ \prod_{i=1}^{m} k_i+d \mid m \in \mathbb{N};\ k_1,\ldots,k_m \in \mathbb{N} \text{ are pairwise prime; } d \in \mathbb{N} \text{ and } \sum_{i=1}^{m} k_i+d = n\}.$$

$$f_4(n) = \max\{ \prod_{i=1}^{m} k_i+d \mid m \in \mathbb{N};\ k_1,\ldots,k_m \in \mathbb{N} \text{ are pairwise prime; } d \in \mathbb{N} \text{ and } \sum_{i=1}^{m} k_i+d \leq n\}.$$

Open problems: investigate f_1, f_2, f_3 and f_4.

AKNOWLEDGEMENT

I wish to thank Dr. J.W. de Bakker, Mathematisch Centrum, for valuable discussions and criticism which resulted in an improved presentation, and Mr. F.A.L.M. Goossens, SARA, who did the programming.

REFERENCES

[1] D. van Dalen; A note on some systems of Lindenmayer, Math. Syst. Theory 5 (1971), 128-140.

[2] L.E. Dickson, History of the theory of numbers #2, Washington, Carnegie Institute, 1920.

[3] P.G. Doucet, On the membership question in some Lindenmayer sytems, Indag. Math. 34 (1972), 45-52.

[4] G.T. Herman, The computing ability of a developmental model for filamenteous organisms, J. Theoret. Biol. 25 (1969), 421-435.

[5] D.E. Knuth, Seminumerical algorithms, Reading, Massachusetts, Addison-Wesley, 1969.

[6] A. Lindenmayer, Mathematical models for cellular interaction in development I & II, J. Theoret. Biol. 18 (1968), 280-315.

[7] G. Rozenberg & P.G. Doucet, On OL-languages, Inform. Contr. 19 (1971), 302-318.

[8] G. Rozenberg & A. Lindenmayer, Developmental systems with locally catenative formulas, Acta Informatica (submitted for publication).

[9] A.L. Szilard, Growth functions of Lindenmayer systems, Tech. Rept. #4, Dept. Comp. Sc. Univ. of Western Ontario, London, Ontario, 1971.

APPENDIX

The programs are used in batch processing mode. Input to the program is presented on the same medium as the program itself. Output appears on the assigned peripheral.

PROGRAM #1
ARCHITECTURE

Description of input format: syntax

```
¢ spaces, tabulations, and carriage returns are skipped ¢

<letter>::= ¢ all characters available except "=" and "/" ¢
<nonzero string>::= <letter>|<letter><nonzero string>
<string>::= <empty>|<nonzero string>
<production rule>::= <letter> => <string>
<grammar>::= <production rule>/|<production rule>/<grammar>
<grammar declaration>::= g/<grammar>/
<beginword declaration>::= b/<nonzero string>/
<endword declaration>::= e/<string>/
<follow up job>::= job = b/<beginword declaration>|
                   job = e/<endword declaration>|
                   job = be/<beginword declaration><endword declaration>
<job>::= job = g/<grammar declaration><beginword declaration>
         <endword declaration>
<long job>::= <job>|<long job><follow up job>
<multiple job>::= <long job> job = /|<long job><multiple job>
```

Description of input format: semantics

To begin with, information concerning the nature of the input is presented. "job = g" signifies: "a new set of production rules (grammar), a new axiom (beginword) and a new targetword (endword) follow".
"g/" identifies the subsequent grammar, written in the obvious way with "=" followed by ">" acting as a production arrow. The grammar is terminated by an additional "/". "b/" identifies the subsequent beginword, i.e. a <nonzero string> terminated by an additional "/".

"e/" identifies the subsequent endword, i.e. a string terminated by an additional "/".

When we use the same grammar to test several beginwords and enwords the <long job> job = / is appropriate, e.g.:

```
job = g/ ¢ expect a grammar, beginword and endword ¢
      g/<grammar>/
      b/<beginword>/
      e/<endword>/
job = b/ ¢ expect a new beginword ¢
      b/<beginword>/
job = e/ ¢ expect a new endword ¢
      e/<endword>/
job = be/ ¢ expect a new beginword and endword ¢
      b/<beginword>/
      e/<endword>/
job = /
```

When several grammars have to be tested in the same run the <multiple job> is used: when one <long job> is finished, "job = g/" is encountered and the program is ready for a new grammar, beginword and endword. The end of the fodder is indicated by "job = /".

<u>Example</u>. $S_1 = <\{a,b\},\{a \to ab, b \to bb\}>$

$a \overset{*}{\Longrightarrow} abbbbb$? $a \overset{*}{\Longrightarrow} abbbb$?

$S_2 = <\{a\},\{a \to aa\}>$

$aa \overset{*}{\Longrightarrow} aaaa$? $a \overset{*}{\Longrightarrow} aaaaa$?

$a \overset{*}{\Longrightarrow} \lambda$?

Input:

```
        job = g/
        g/a =>   ab/b =>   bb//
        b/a/
        e/abbbbb/
        job = e/
        e/abbbb/
        job = g/
        g/a => aa//
        b/aa/
        e/aaaa/
        job = be/
        b/a/
        e/aaaaa/
        job = e/
        e//
        job = /
```

Description of the output format

The program processes one <job> or <follow up job> at a time. First the corresponding input is printed, then the jobnumber, the number of generations which were needed to reach a conclusion, the phase of the algorithm in which the conclusion was reached (cf. 3.3) and the conclusion itself. More precisely:

```
<idgit>::= 1|2|3|4|5|6|7|8|9
<digit>::= 0|<idgit>
<number>::= <idgit>|<number><digit>
<phase number>::= 1|2|3|4|5|6| 5<digit><digit><digit><digit><idgit>
<solution>::= solution found | no solution
<print input>::= <job>|<follow up job>
<job output>::= <print input>
                jobnumber: <number>
                number of generations: <number>
                phase: <phase number>
                <solution>
<multiple job output>::= <job output> job = /|
                         <job output><multiple job output>
```

The <jobnumber> is numbered consecutively from 1 to n (when the <multiple job> contains n <job>'s and <follow up job>'s).

phase number: i signifies "in phase i (of algorithm 3.3) a conclusion was reached".
phase number: 5 ... i signifies "in the finite case (phase 5) the i-th monorecursive letter of $\delta^p(\sigma)$ did not produce a prefix of the (reduced) endword ω_τ".
number of generations: n indicates $\delta^n(\sigma) = \omega_\tau$ or $|\delta^n(\sigma)|_V > |\omega_\tau|_V$ or, only in the finite case, $n = 2p-p_m$.
<solution> is "solution found" if $\sigma \overset{*}{\Longrightarrow} \omega_\tau$ and "no solution" if $\sigma \overset{*}{\not\Longrightarrow} \omega_\tau$

Example. (output of first input <job> example)

```
job = g/
g/a  =>  ab/b  =>  bb//
b/a/
e/abbbbb/
jobnumber: 1
number of generations: 3
phase: 3
no solution
```

Error messages

error job control input ← subsequent to "job =" one of the following symbols is missing: "g", "b", "e", or "/".

error in input ← (i) input format incorrect, e.g. grammar inputted after begin- or endword.
(ii) insufficient input, e.g. no grammar, begin- or endword.
(iii) the identification symbol in front of grammar, begin- and endword is not a "g", "b" and "e".

no transition sign ← a production rule without "=" has been encountered, symbol in grammar not defined ← a symbol occurs in the righthandside of a production rule for which no production rule is given, i.e. δ is not total.

symbol in word not in grammar ← a symbol occurs in the begin- or end-word, for which no production rule is given.

array memory overflow ← the production being executed overwrites indispensable memory, e.g. production rules.

program error ← either program or algorithm (or computer) is defective.

program end ← normal program termination.

ORGANIZATION

The program is written for use on the EL-X8 at the Mathematical Centre. The characterset used is that of the MC-flexowritercode. Transput is according to the ALGOL-60 compiler of the MC and only two procedures make use of it: procedure error and procedure nextsymbol.
C.f.: D. Grune, Handleiding Milli systeem voor de EL-X8, LR 1.1, Mathematisch Centrum, 1971.

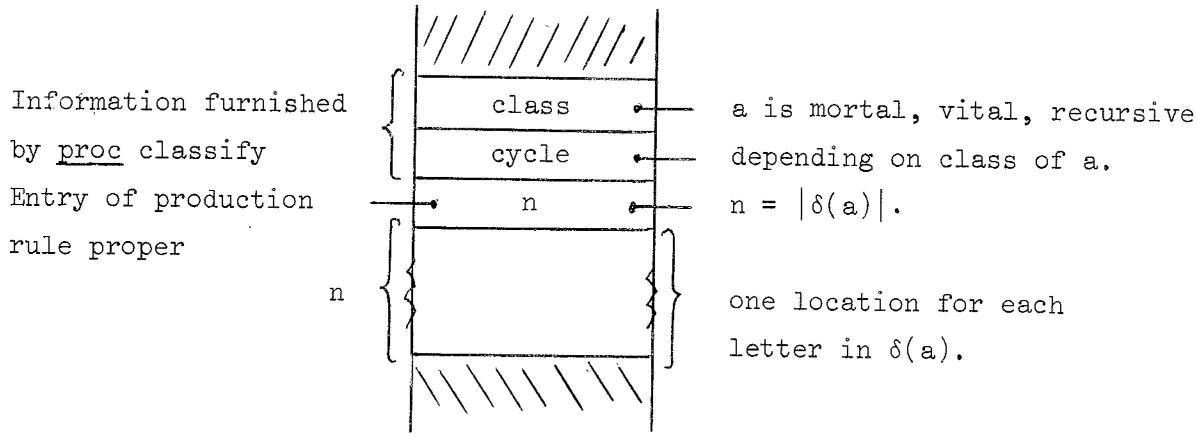
To promote efficient use of memory, the production rules, beginword, endword, the concurrently produced word, the previously produced word, and, if necessary, information for application of the Chinese Remainder Theorem are stored in one array called "array" (see figure).

production rules	endword	memory used for word generation	beginword

Low core   high core

"array"

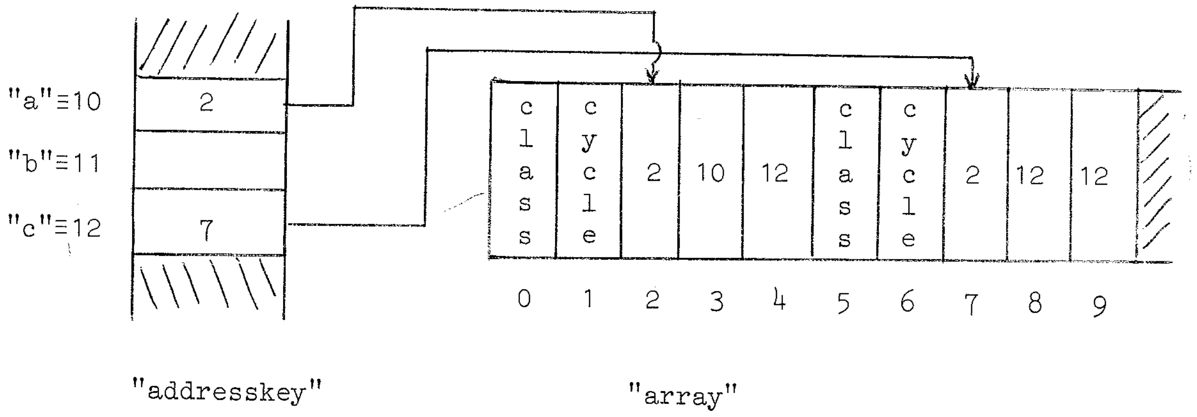
The production rules are stored as follows (the figure depicts the storage of a → $\delta(a)$)



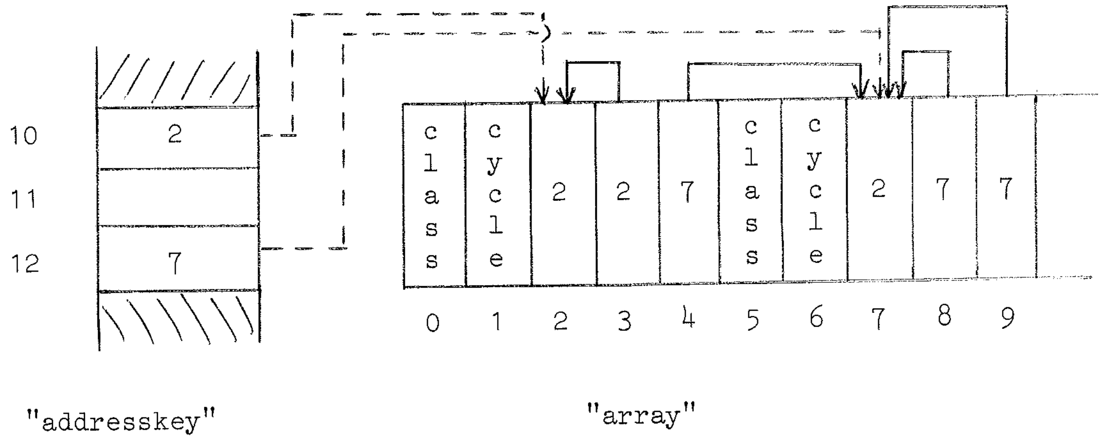
If a $\in \sum_r$ cycle:= min K_a.
If a $\in \sum_m$ cycle:= number of generations needed to derive λ.
If a $\in \sum_v$ cycle:= $|\sum|$.

To construct the correct pointers between the different entries in the table of production rules an additional array "addresskey" is used. "addresskey" has one location for each character in the characterset (in our case 127), and only uses those which are defined in the production rules.

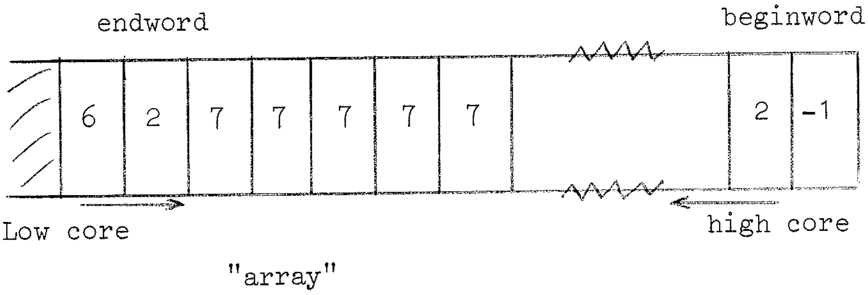
Example. G = <{a,c},{a→ac,c→cc},a>



Subsequent to the reading of the production rules, the pointers in "array" to "addresskey" are replaced by corresponding pointers to "array" itself (assemblage).

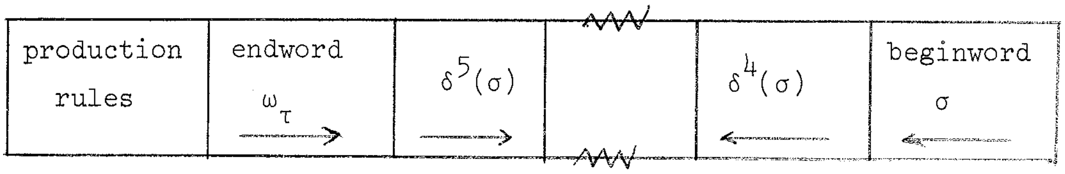


All words are stored in a similar way. The entry point contains the word length. Subsequent elements of the array contain pointers to the corresponding production rule. When the word is stored at high core, the word length is taken negative otherwise positive.
Let the beginword be "a" and the endword "accccc".



Productions are executed as follows. The first production will transform the beginword σ into the next word $\delta(\sigma)$ stored at low core from the endword upwards. $\delta^2(\sigma)$ will be stored at high core from the beginword downwards. $\delta^3(\sigma)$ overwrites $\delta(\sigma)$ at low core etc.

State of "array" after five productions.



"array"

Productions from individual letters, are, if needed, executed in a similar fashion.

PROGRAM #2

The program generates a finite propagation of a given DOL $G = <\Sigma, \delta, \sigma>$ e.g.

$$\xi(G) = \sigma, \delta(\sigma), \delta^2(\sigma), \ldots, \delta^k(\sigma).$$

Only those $\delta^i(\sigma)$ $(0 \leq i \leq k)$ are printed which are specified in the input.

ARCHITECTURE

The input format is similar to that of program #1 with the following alterations:

job = gp/ ¢ "p" stands for "print generated words as specified in print command". When "p" is omitted, no word except the beginword is printed. ¢

e/<number>/ ¢ <number> replaces <string> in <endword declaration> of program #1, and determines the number of productions the program executes. ¢

¢ followed by the printcommand ¢
p/f<number>l<number>s<number>/
which means:

```
for i:= 0 step 1 until F-1, F step S until k-L-1,
k-L step 1 until k do print (δ^i(σ));
```

where F = <number> following f
L = <number> following l
S = <number> following s

f<number> omitted: F:= 0
l<number> omitted: L:= 0
s<number> omitted: S:= 1

Error messages. Similar to ptogram #1, but for "program error", and in addition:

no separator after countcommand	←	"/" omitted after "e/k/"
error in printspecification	←	"p/f<number>l<number>s<number>/" is not correct.

```
A2317R.32,PAUL VITANYI

'BEGIN' 'COMMENT' 2317R, AUGUST 1971, PAUL VITANYI AND FRANK GOOSSENS;
        'INTEGER' BEGIN OF MEMORY,END OF MEMORY,FIRSTKEY,LASTKEY;
                  BEGIN OF MEMORY:=1;
                  END OF MEMORY:=30000;
                FIRSTKEY:=0;
                  LASTKEY:=127;
        'BEGIN' 'INTEGER' 'ARRAY' ADDRESSKEY[FIRSTKEY:LASTKEY]
        ,ARRAY[BEGIN OF MEMORY : END OF MEMORY]
        ;
        'INTEGER' SEPARATOR,EQUALSIGN,TAB,TWNR,SPACE
                  ,B,G,E
                 ,CYCLE,  CLASS,INFOGRAM
                  ,JOB,COUNT,PHASE
                  ,P,PM,PVR,VITALP,VITALBW,VITALEND
                  ,BW,EW,EWNOS,BOM,EOM
                  ,MORTAL,RECURSIVE,VITAL
                  ,I,NBW,EWN,EWHELP,BEGINPAIR,LW,LETTER,J,PL,M,BOMMIN,D
                  ;
        'BOOLEAN' SOLUTION,GRAM,BEGIN,END
                  ;

        'PROCEDURE' INITIALIZE;
        'BEGIN' JOB:=MORTAL:=0;
                SEPARATOR:=67;EQUALSIGN:=70;TAB:=118;TWNR:=119;
                SPACE:=93;
                B:=11;G:=16 ;E:=14;
                CYCLE:=-1;CLASS:=-2;INFOGRAM:=2:
                RECURSIVE:=1;VITAL:=-1;
        'END' INITIALIZE;

        'PROCEDURE' ERROR(STRING);'STRING' STRING;
        'BEGIN' CARRIAGE(2);PRINTTEXT(STRING);NLCR;
                PRINTTEXT('('JOBNUMBER:')');ABSFIXT(2,0,JOB);EXIT
         'END' ERROR;

        'PROCEDURE' READ INPUT;
        'BEGIN' 'INTEGER' CODE,DUMMY;
                'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ EQUALSIGN 'DO';
                'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ SEPARATOR 'DO'
                'IF' CODE=G 'THEN' GRAM:=BEGIN:=END:='FALSE' 'ELSE'
                'IF' CODE=B 'THEN' BEGIN:='FALSE' 'ELSE'
                'IF' CODE=E 'THEN' END:='FALSE' 'ELSE'
               ERROR('('ERROR JOB CONTROL  INPUT')');
                'IF' GRAM ∧ BEGIN ∧ END 'THEN' ERROR('('PROGRAM END')');
                'FOR' DUMMY:=0 'WHILE' ¬GRAM ∨ ¬BEGIN ∨ ¬END 'DO'
                'BEGIN' CODE:=NEXTSYMBOL;
                'IF' ¬GRAM ∧ CODE=G 'THEN'
                'BEGIN' READ GRAMMAR;GRAM:='TRUE' 'END' 'ELSE'
                'IF' ¬BEGIN ∧ CODE=B ∧ GRAM 'THEN'
                'BEGIN' GENERATE BEGINWORD;BEGIN:='TRUE' 'END' 'ELSE'
                'IF' ¬END ∧ CODE=E ∧ GRAM 'THEN'
                'BEGIN' GENERATE ENDWORD;END:='TRUE' 'END' 'ELSE'
               ERROR('('ERROR IN INPUT')');
                'END'
       'END' READ INPUT;
```

```
'PROCEDURE' READ GRAMMAR;
'BEGIN' 'INTEGER' SYMBOL,SYM,HELP,I,J;
      'FOR' I:=FIRSTKEY 'STEP' 1 'UNTIL' LASTKEY 'DO'ADDRESSKEY[I]:=0;
      BOM:=BEGIN OF MEMORY; EOM:= END OF MEMORY;
      P:=0; NEXTSYMBOL;
      'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
      'FOR' SYMBOL:=NEXTSYMBOL 'WHILE' SYMBOL‡SEPARATOR'DO'
      'BEGIN' 'IF' NEXTSYMBOL‡EQUALSIGN 'THEN' ERROR('('NO TRANSITION SIGN')');
            NEXTSYMBOL;
            'COMMENT' ARROWSIGN OF TRANSITIONSIGN IS READ;
            BOM:=BOM+INFOGRAM;
            ARRAY[BOM+CYCLE]:=ARRAY[BOM+CLASS]:=ARRAY[BOM]:=0;
            HELP:=ADDRESSKEY[SYMBOL]:=BOM;BOM:=BOM+1;
            'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ‡ SEPARATOR 'DO'
            'BEGIN' ARRAY[BOM]:=SYM:BOM:=BOM+1 'END';
            ARRAY[HELP]:=BOM-HELP-1:
            P:=P+1;
            'COMMENT' GRAMMARRULE COUNTER;
      'END';
EW:=BOM;
      J:=1;
      'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
      'BEGIN' J:=J+INFOGRAM;
            HELP:=ARRAY[J];
            'FOR' SYM:=1 'STEP' 1 'UNTIL' HELP 'DO'
            'BEGIN' SYMBOL:=ADDRESSKEY[ARRAY[J+SYM]];
                  'IF' SYMBOL=0 'THEN' ERROR('('SYMBOL IN GRAMMAR NOT DEFINED')')
                  'ELSE' ARRAY[J+SYM]:=SYMBOL
            'END';
            J:=J+HELP+1
      'END';
      CLASSIFY(P,PM)
'END' READ GRAMMAR;

'PROCEDURE' READ WORD(BADDRESS,DIRECTION,EADDRESS);
'VALUE'      DIRECTION;'INTEGER' BADDRESS,DIRECTION,EADDRESS;
'BEGIN' 'INTEGER' SYM,HELP;
      BADDRESS:=EADDRESS;
      EADDRESS:=EADDRESS+DIRECTION;
      NEXTSYMBOL;
      'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
      'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ‡ SEPARATOR 'DO'
      'BEGIN' HELP:=ADDRESSKEY[SYM];
            'IF' HELP=0 'THEN' ERROR('('SYMBOL IN WORDNOT IN GRAMMAR')') 'ELSE'
            ARRAY[EADDRESS]:=HELP;
            EADDRESS:=EADDRESS+DIRECTION
      'END';
      ARRAY[BADDRESS]:=EADDRESS-BADDRESS-DIRECTION
'END' READ WORD;

'PROCEDURE' GENERATE BEGINWORD;
'BEGIN' EOM:=END OF MEMORY;
      READ WORD(BW,-1,EOM);
'END' GENERATE BEGINWORD;

'PROCEDURE' GENERATE ENDWORD;
'BEGIN' BOM:=EW;
      READ WORD(EW,1,BOM);
      EWNOS:=ARRAY[EW];
'END' GENERATE ENDWORD;
```

```
'INTEGER' 'PROCEDURE' NEXTSYMBOL;
'BEGIN' 'INTEGER' SYM;
NEXT:   SYM:=RESYM;PRSYM(SYM);
        'IF' SYM=SPACE ∨ SYM=TAB ∨ SYM=TWNR 'THEN' 'GOTO' NEXT
;NEXTSYMBOL:=SYM
'END' NEXTSYMBOL;

'INTEGER' 'PROCEDURE' CLASS SYMBOLS(ADDRESS,KIND);
'VALUE' ADDRESS,KIND;'INTEGER' ADDRESS,KIND;
'BEGIN' ''INTEGER' I,N,HELP;
        N:=ARRAY[ADDRESS]}
        HELP:=0;
        'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
        'IF' ARRAY[ARRAY[ADDRESS+I]+CLASS]=KIND 'THEN'
        HELP:=HELP+1;
        CLASS SYMBOLS:=HELP
'END' CLASS SYMBOLS;

'INTEGER' 'PROCEDURE' GCD(A,B);'VALUE' A,B;'INTEGER' A,B;
'BEGIN' 'INTEGER' W;
AGAIN:  'IF' B≠ 0 'THEN'
        'BEGIN' W:=A-A÷B*B;
                A:=B; B:=W; 'GOTO' AGAIN
        'END';
        GCD:= ABS(A)
'END' GCD;

'BOOLEAN' 'PROCEDURE' COMPPREFIX(WORD,EW,EWN);'VALUE' WORD,EW,EWN;'INTEGER' WORD,EW,EWN;
'BEGIN' 'INTEGER' I,M;
        COMPPREFIX:='FALSE';
        M:=ARRAY[WORD];
        'IF' EWN < ABS(M) 'THEN' 'GOTO' READY;
        'FOR' I:=1 'STEP' 1 'UNTIL' M,-1 'STEP' -1 'UNTIL'M 'DO'
        'IF' ARRAY[WORD+I]≠ ARRAY[EW+ ABS(I)] 'THEN' 'GOTO' READY;
        COMPPREFIX:='TRUE';
READY:
'END' COMPPREFIX;

'PROCEDURE' CLASSIFY(P,PM);'VALUE' P;'INTEGER' P,PM;
'BEGIN' 'INTEGER' I,J,NI,ADDRESS,SYMBOL,CIRCLE,KIND,N;
        PM:=0;
        'FOR' J:=FIRSTKEY 'STEP' 1 'UNTIL' LASTKEY 'DO'
        'BEGIN' ADDRESS:=SYMBOL:=ADDRESSKEY[J];CIRCLE:=0;
                'IF' SYMBOL ≠0 'THEN'
                'BEGIN'
                'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
                'BEGIN' 'IF' I > 1 'THEN' GENERATE(ADDRESS);
                        N:=ARRAY[ADDRESS];
                        'IF' N=0 'THEN'
                        'BEGIN' KIND:=0:PM:=PM+1;'GOTO' READY'END' 'ELSE'
                        'FOR' NI:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
                        'IF' ARRAY[ADDRESS+NI]=SYMBOL 'THEN'
                        'BEGIN' KIND:=1;CIRCLE:=I;'GOTO' READY 'END'
                'END';
                KIND:=-1;
        READY:  ARRAY[SYMBOL+CYCLE]:=CIRCLE;
                ARRAY[SYMBOL+CLASS]:=KIND
                'END'
        'END'
```

```
         'END' CLASSIFY;

         'PROCEDURE' GENERATE(ADDRESS);'INTEGER' ADDRESS;
         'BEGIN' 'INTEGER' N,STEP,NEW,K,I,R,M,J:
                 N:=ARRAY[ADDRESS];
                 'IF' N < 0 'THEN'
                 'BEGIN' STEP:=1;NEW:=BOM 'END' 'ELSE'
                 'BEGIN' STEP:=-1;NEW:=EOM 'END';
                 K:=0;
                 'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
                 'BEGIN' R:=ARRAY[ADDRESS+I];
                         M:=ARRAY[R];
                         'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO'
                         'BEGIN' K:=K+STEP;
                                 ARRAY[NEW+K]:=ARRAY[R+J]
                         'END'
                 'END';
                 'IF' NEW+K < BOM ∨ NEW+K > EOM 'THEN'  ERROR('('ARRAY MEMORY OVERFLOW')');
                 ARRAY[NEW]:=K;
                 ADDRESS:=NEW
         'END' GENERATE;

         'PROCEDURE' COMPARE(ADDRESS);'VALUE' ADDRESS;'INTEGER' ADDRESS:
         'BEGIN' 'INTEGER' N,I,STEP;
                 N:=ARRAY[ADDRESS];
                    VITALBW:=ABS(N)-CLASS SYMBOLS(BW,MORTAL);
                 'IF' VITALEND < VITALBW 'THEN' 'GOTO' END OF JOB
  ;
                 'IF' EWNOS= ABS(N) 'THEN'
                 'BEGIN' STEP:='IF' N < 0 'THEN' -1 'ELSE' 1;
                         'FOR' I:=1 'STEP' 1 'UNTIL' EWNOS 'DO'
                         'IF' ARRAY[EW+I]≠ ARRAY[ADDRESS+STEP*I] 'THEN'
                         'GOTO' READY;
                         SOLUTION:='TRUE';'GOTO' END OF JOB
                 'END';
         READY:
         'END' COMPARE;


         INITIALIZE;
START:   PHASE:=1; SOLUTION:='FALSE';JOB:=JOB+1;
         READ INPUT;
         COUNT:=0;
         VITALEND:=EWNOS- CLASS SYMBOLS(EW,MORTAL);
         COMPARE(BW);
         PHASE:=2;
         'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
         'BEGIN' GENERATE(BW);
                 COUNT:=COUNT+1;
                 COMPARE(BW)
         'END';
         PHASE:=3;
         VITALP:=VITALBW;
         PVR:=P-PM;
         'FOR' I:=1 'STEP' 1 'UNTIL' PVR 'DO'
         'BEGIN' GENERATE(BW);
                 COUNT:=COUNT+1;
                 COMPARE(BW)
         'END';
         NBW:=ARRAY[BW];
```

```
     'IF' VITALP=VITALBW'THEN' 'GOTO' VARYING 'ELSE'
     'IF' VITALP > VITALBW'THEN' ERROR('('PROGRAM ERROR')');
     PHASE:=4;
PRODUCE:
          GENERATE(BW);
          COUNT:=COUNT+1;
     COMPARE(BW);
     'GOTO' PRODUCE;
VARYING:PHASE:=5;
     EWHELP:=EW;EWN:=EWNOS;
     'IF' NBW < 0 'THEN' EOM:=EOM+NBW-1 'ELSE'
     BOM:=BOM+NBW+1;
     BEGINPAIR:=BOM;
     'FOR' I:=1 'STEP' 1 'UNTIL' NBW ,-1 'STEP' -1 'UNTIL' NBW 'DO'
     'BEGIN' LW:=LETTER:=ARRAY[BW+I];
          'IF' ARRAY[LETTER+CLASS]= RECURSIVE 'THEN'
          'BEGIN' PHASE:=50000+ ABS(I);
               'FOR' J:=1 'STEP' 1 'UNTIL' PM 'DO'
               GENERATE(LW);
               PL:=ARRAY[LETTER+CYCLE]:
               'FOR' J:=1 'STEP' 1 'UNTIL' PL 'DO'
               'BEGIN' GENERATE(LW);
                    'IF' COMPPREFIX(LW,EWHELP,EWN) 'THEN'
                    'BEGIN' M:= ABS(ARRAY[LW]);
                         EWHELP:=EWHELP+M;
                         EWN:=EWN-M;
                         ARRAY[BOM]:=COUNT+PM+J;
                         ARRAY[BOM+1]:=PL;
                         BOM:=BOM+2;
                         'GOTO' NEXTLETTER
                    'END'
               'END';
               'GOTO' END OF JOB
          'END';
     NEXTLETTER:
     'END';
     PHASE:=6;
     BOMMIN:=BOM-2;
     'FOR' I:=BEGINPAIR 'STEP' 2 'UNTIL' BOMMIN 'DO'
     'FOR' J:=I+2 'STEP' 2 'UNTIL' BOMMIN 'DO'
     'BEGIN' D:= GCD(ARRAY[I+1],ARRAY[J+1]);
          'IF' REMAINDER(ARRAY[I],D) = REMAINDER(ARRAY[J],D) 'THEN' 'ELSE' 'GOTO' END OF JOB
     'END';
     SOLUTION:='TRUE';
END OF JOB: NLCR;PRINTTEXT('('JOBNUMBER:')');ABSFIXT(2,0,JOB);
     NLCR;PRINTTEXT('('NUMBER OF GENERATIONS:')');ABSFIXT(5,0,COUNT);
     NLCP;PRINTTEXT('('PHASE:')');ABSFIXT(5,0,PHASE);NLCR;
     ''F' SOLUTION 'THEN' PRINTTEXT('(' SOLUTION FOUND.')') 'ELSE' PRINTTEXT('(' NO SOLUTION.')');
      BW:=END OF MEMORY;
     EOM:=END OF MEMORY+ARRAY[END OF MEMORY]-1;
     BOM:=EW+ARRAY[EW]+1;
     CARRIAGE(10);
     'GOTO' START
'END'
'END'
```

```
040872-172      A 2317R.32 PAULVITANYI              6

JOB=G/
G/ S=>PQ/ P=>V*V*/ V=>TYGER/ *=>,/ Q=>WX/ W=>BURNING/ X=>BRIGHT/ U=>12/ 1=>345/
2=>648/ 3=>IN/ 4=>THE/ 5=>FORESTS/ 6=>OF/ 8=>NIGH/ H=>0/0=>9/ 9=>T /
,=> / A=> /B=> /C=> /D=> /E=> /F=> /G=> /I=> /L=> /M=> /N=> /O=> /R=> /T=> /
Y=> /?=> / /
B/S/
E/   TYGER, TYGER, BURNING BRIGHT /
JOBNUMBER:  1
NUMBER OF GENERATIONS:     3
PHASE:      2
 SOLUTION FOUND.
```

Job 1-3. The Tyger

Tyger! Tyger! burning bright
in the forests of the night
what immortal hand or eye
could frame thy fearful symmetry?

William Blake in: Songs of Experience.

Presumably, the produced solutions reflect the intention of the poet.

```
JOB=E/
E/ IN THE FORESTS OF THE NIGHT /
JOBNUMBER:  2
NUMBER OF GENERATIONS:     6
PHASE:      2
 SOLUTION FOUND.
```

```
JOB=E/
E/ WHAT  IMMORTAL HAND OR EYE
COULD FRAME.THY FEARFUL SYMMETRY ? /
JOBNUMBER:  3
NUMBER OF GENERATIONS:    18
PHASE:      2
 NO SOLUTION.
```

```
JOB=G/
G/ A=>B/ B=>AB //
B/ A/
E/ BAB/
JOBNUMBER:  4
```

N.B. Program #2 has a printout of consecutively generated strings for all DOL's used as an example in Program #1.

```
NUMBER OF GENERATIONS:      3
PHASE:     3
 SOLUTION FOUND.
```

Job 4-6. The lengths of the consecutively generated words

$\sigma, \delta(\sigma), \delta^2(\sigma), \ldots$

form the main Fibonacci series i.e.

1,1,2,3,5,8,...

(cf. 1.2 example 3).

```
JOB=E/
E/ BABBAB/
JOBNUMBER:  5
NUMBER OF GENERATIONS:      5
PHASE:     4
 NO SOLUTION.
```

```
JOB=E/
E/ ABBABBABABBAB/
JOBNUMBER:  6
NUMBER OF GENERATIONS:      6
PHASE:     4
 SOLUTION FOUND.
```

```
JOB=G/
G/ A=>BC/ B=>KD/ C=> EK/ D=>GB/ E=>CF/ F=>IH/ G=>HI/ H=>DE/ I=>K/ K=>K/ /
B/A/
E/ KHIKDEKIHK/
JOBNUMBER:  7
NUMBER OF GENERATIONS:      4
PHASE:     2
 SOLUTION FOUND.
```

Job 7-9. The DOL generates a sequence which can be interpreted as the development of the marginal meristem of a compound leaf. The system generates longer and longer strings on the leaf margin, with portions corresponding to lobes, and each lobe eventually splitting into three new lobes. We interpret the string by assigning cells in state k to non-growing portions (notches) on the leaf margin occupying positions between adjacent lobes or leaflets and assigning cells in all other states to growing portions of the margin.
(Rozenberg & Lindenmayer [8])

```
JOB=E/
E/ KHIKDEDKIHK/
JOBNUMBER:  8
NUMBER OF GENERATIONS:      5
PHASE:      2
 NO SOLUTION.
```

```
JOB=E/
E/ KHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHK/
JOBNUMBER:  9
NUMBER OF GENERATIONS:      7
PHASE:      2
 SOLUTION FOUND.
```

```
JOB=G/
G/A=>BAC/ C=>CC/ B=>BB/ /
B/A/
E/BBBBBBBBBBBBBBBPBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
JOBNUMBER: 10
NUMBER OF GENERATIONS:      5
PHASE:      3
 NO SOLUTION.
```

```
JOB=E/
E/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
JOBNUMBER: 11
NUMBER OF GENERATIONS:      5
PHASE:      3
 SOLUTION FOUND.
```

σ	a	
$\delta(\sigma)$	bc	
$\delta^2(\sigma)$	kdek	
$\delta^3(\sigma)$	kgbcfk	
$\delta^4(\sigma)$	khikdekihk	
$\delta^5(\sigma)$	$\underbrace{\text{kdek}}\ \underbrace{\text{kgbcfk}}\ \underbrace{\text{kdek}}$	see fig. 1.
$\delta^6(\sigma)$	$\underbrace{\text{kgbcfk}}\ \underbrace{\text{khikdekihk}}\ \underbrace{\text{kgbcfk}}$	see fig. 2.
$\delta^7(\sigma)$	$\underbrace{\text{khikdekihk}}\ \underbrace{\text{kdekkgbcfkkdek}}\ \underbrace{\text{khikdekihk}}$	see fig. 3.

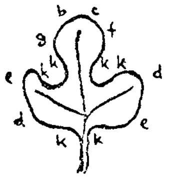


Fig. 1.

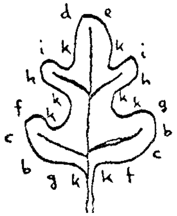


Fig. 2.

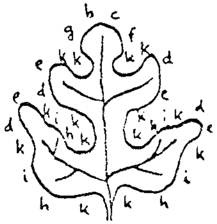
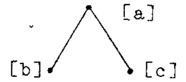


Fig. 3.

Job 10-12. The semi DOL has the following RCS:



hence L(G) is infinite (theorem 3.7 and sequel).

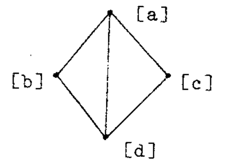
```
JOB=E/
E/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
JOBNUMBER: 12
NUMBER OF GENERATIONS:     6
PHASE:      3
 NO SOLUTION.
```

```
JOB=G/
G/ A=>ABCD/ B=>BD/ C=>CD/ D=>D//
B/A/
E/ABCDBDCDDBDDCDDDBDDDCDDDDBDDDDCDDDDDBDDDDDCDDDDDDBDDDDDDCDDDDDDDBDDDDDDDCDDDDD
DDDBDDDDDDDDCDDDDDDDDDBDDDDDDDDDCDDDDDDDDDDBDDDDDDDDDDCDDDDDDDDDDDBDDDDDDDDDDD
DCDDDDDDDDDDDDBDDDDDDDDDDDDCDDDDDDDDDDDDDD/
JOBNUMBER: 13
NUMBER OF GENERATIONS:     14
PHASE:      4
 NO SOLUTION.
```

```
JOB=G/
G/  K=>L/ L=>M/ M=>K/ 1=>2/ 2=>3/ 3=>4/ 4=>5/ 5=>6/ 6=>7/ 7=>1/ +=>-/ -=>+/
A=>B/ B=>C/ C=>D/ D=>E/ E=>A/ /
B/L3+A4/E/K2-D7/
JOBNUMBER: 14
NUMBER OF GENERATIONS:     34
PHASE:      6
 NO SOLUTION.
```

```
JOB=E/E/M3+D2/
JOBNUMBER: 15
NUMBER OF GENERATIONS:     34
PHASE:      6
 NO SOLUTION.
```

Job 13. The semi DOL has the following RCS:



hence L(G) is infinite (theorem 3.7 and sequel).

Job 14-20. The semi DOL and an axiom determine a finite language of well formed arithmetical expressions.
The RCS consists of:

[k] [1] [+] [a]
 ∘   ∘   ∘   ∘

```
040872-172      A 2317R.32 PAULVITANYI                    10                                    61

JOB=E/E/K17D-/
JOBNUMBER: 16
NUMBER OF GENERATIONS:     34
PHASE: 50003
 NO SOLUTION.


JOB=BE/B/K7+E6/E/M6-D5/
JOBNUMBER: 17
NUMBER OF GENERATIONS:     34
PHASE:      6
 SOLUTION FOUND.


JOB=E/E/K1-D7/
JOBNUMBER: 18
NUMBER OF GENERATIONS:     34
PHASE:      6
 SOLUTION FOUND.


JOB=E/E/K1-DD/
JOBNUMBER: 19
NUMBER OF GENERATIONS:     34
PHASE: 50005
 NO SOLUTION.
```

Since $L(G)$ is finite and $\sum = \sum_V$, $2 * |\sum| = 34$ generations are executed if ω_τ is not encountered in the meantime. Then the Chinese Remainder theorem is applied to obtain a solution. (cf. algorithm 3.3). Note that

$$\begin{aligned}|L(G)| &= \text{l.c.m. } (|[k]|,|[1]|,|[+]|,|[a]|)\\ &= \text{l.c.m. } (2,3,5,7)\\ &= 210.\end{aligned}$$

```
JOB=G  /G/1=>2*4/2=>2/4=>2*5/5=>6#5/6=>7/7=>8/8=>9(1)/9=>9/(=>(/)=>)/ *=>*/
#=>#//B/1/
E/2*2*9(1)#8#7#6#5/
JOBNUMBER: 20
NUMBER OF GENERATIONS:     6
PHASE:     2
 SOLUTION FOUND.
```

```
JOB=E/
E/2*2*9(2*2*9(1)#8#7#6#5)#9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)
#9(1)#8#7#6#5/
JOBNUMBER: 21
NUMBER OF GENERATIONS:    12
PHASE:     2
 SOLUTION FOUND.
```

```
JOB=/

PROGRAM END
JOBNUMBER: 22
```

Job 20-21. This DOL simulates the development of Callithamnion Roseum (a red alga), cf. A. Lindenmayer, Developmental systems without cellular interactions, their languages and grammars, J. Theoret. Biol. 30 (1971), 455-484.

The DOL takes into account the progression from transverse to oblique cell walls which is so characteristic for this system. 1-9 (except 3) symbolize cells in diverse states, left and right parenthesis indicate branches, and the two additional symbols * and # indicate the presence of transverse and oblique cell walls between cells. $\sigma = 1$. E.g. $8 \to 9(1)$ means that a cell in state 8 divides and gives rise to a basal cell in state 9 and a branch cell in state 1. $1 \to 2 * 4$ means that a cell in state 1 undergoes division and gives rise to two new cells in state 2 and 4 separated by a transverse wall.

$\sigma = 1$

$\delta(\sigma) = 2 * 4$

$\delta^2(\sigma) = 2 * 2 * 5$.

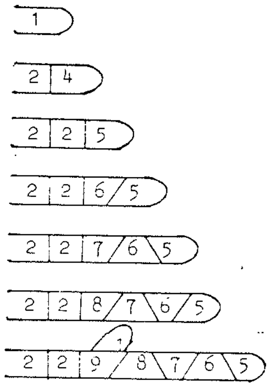
$\delta^3(\sigma) = 2 * 2 * 6 \# 5$

$\delta^4(\sigma) = 2 * 2 * 7 \# 6 \# 5$

$\delta^5(\sigma) = 2 * 2 * 8 \# 7 \# 6 \# 5$

$\delta^6(\sigma) = 2 * 2 * 9(1) \# 8 \# 7 \# 6 \# 5$

.
.
.



```
'BEGIN' 'COMMENT' 2317R, AUGUST 1971, PAUL VITANYI AND FRANK GOOSSENS;
        'INTEGER' BEGIN OF MEMORY,END OF MEMORY,FIRSTKEY,LASTKEY;
                  BEGIN OF MEMORY:=1;
                  END OF MEMORY:=30000;
                FIRSTKEY:=0;
                  LASTKEY:=127;
        'BEGIN' 'INTEGER' 'ARRAY' ADDRESSKEY[FIRSTKEY:LASTKEY]
        ,ARRAY[BEGIN OF MEMORY : END OF MEMORY]
        ;
        'INTEGER' SEPARATOR,EQUALSIGN,TAB,TWNR,SPACE
                  ,KEY
                   ,S,STEP
                   ,B,G,E
                   ,P,F,L,FROM,FIRST,LAST,JOB,COUNT,BW,BOM,EOM,I
                   ,INFOGRAM
                   ;
        'BOOLEAN' PRINTER,BEGIN,END,GRAM;

        'PROCEDURE' INITIALIZE;
        'BEGIN' JOB:=0;
                SEPARATOR:=67;EQUALSIGN:=70;TAB:=118;TWNR:=119;
                SPACE:=93;
                B:=11;G:=16 ;E:=14;
                P:=25;F:=15; L:=21;
                S:=28;
                INFOGRAM:=1;KEY:=1;FIRST:=LAST:=0;
        'END' INITIALIZE;

        'PROCEDURE' ERROR(STRING);'STRING' STRING;
        'BEGIN' CARRIAGE(2);PRINTTEXT(STRING);EXIT
         'END' ERROR;

        'PROCEDURE' READ INPUT;
        'BEGIN' 'INTEGER' CODE;
                'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ EQUALSIGN 'DO';
                'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ SEPARATOR 'DO'
                'IF' CODE=G 'THEN' GRAM:=BEGIN:=END:= 'FALSE'  'ELSE'
                'IF' CODE=B 'THEN' BEGIN:='FALSE' 'ELSE'
                'IF' CODE=E 'THEN' END:='FALSE' 'ELSE'
                'IF' CODE=P 'THEN' PRINTER:='FALSE' 'ELSE'
                ERROR('('ERROR JOB CONTROL  INPUT')');
                'IF' GRAM ∧ BEGIN ∧ END ∧ PRINTER 'THEN'
                ERROR('('PROGRAM END')');
                'FOR' CODE:=CODE 'WHILE' ¬GRAM ∨ ¬BEGIN ∨ ¬END ∨ ¬PRINTER 'DO'
                'BEGIN' CODE:=NEXTSYMBOL;
                'IF' ¬GRAM ∧ CODE=G 'THEN'
                'BEGIN' READ GRAMMAR;GRAM:='TRUE' 'END' 'ELSE'
                'IF' ¬BEGIN ∧ CODE=B ∧ GRAM 'THEN'
                'BEGIN' GENERATE BEGINWORD;BEGIN:='TRUE' 'END' 'ELSE'
                'IF' ¬END ∧ CODE=E ∧ GRAM 'THEN'
                'BEGIN' READ COUNT;END:='TRUE' 'END' 'ELSE'
                'IF' ¬PRINTER ∧ CODE= P 'THEN'
                'BEGIN' READ PRINTCOMMANDS;PRINTER:='TRUE' 'END' 'ELSE'
                ERROR('('ERROR IN INPUT')');
                'END'
        'END' READ INPUT;
```

```
'PROCEDURE' READ GRAMMAR;
'BEGIN' 'INTEGER' SYMBOL,SYM,HELP,I,J,P;
        'FOR' I:=FIRSTKEY 'STEP' 1 'UNTIL' LASTKEY 'DO'ADDRESSKEY[I]:=0;
        BOM:=BEGIN OF MEMORY; EOM:= END OF MEMORY;
        P:=0; NEXTSYMBOL;
        'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
        'FOR' SYMBOL:=NEXTSYMBOL 'WHILE' SYMBOL≠SEPARATOR'DO'
        'BEGIN' 'IF' NEXTSYMBOL≠EQUALSIGN 'THEN' ERROR('('NO TRANSITION SIGN')');
                NEXTSYMBOL;
                'COMMENT' ARROWSIGN OF TRANSITIONSIGN IS READ;
                BOM:=BOM+INFOGRAM;
                ARRAY[BOM]:=0;
                ARRAY[BOM+KEY]:=SYMBOL;
                HELP:=ADDRESSKEY[SYMBOL]:=BOM;BOM:=BOM+1;
                'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ≠ SEPARATOR 'DO'
                'BEGIN' ARRAY[BOM]:=SYM;BOM:=BOM+1 'END';
                ARRAY[HELP]:=BOM-HELP-1;
                P:=P+1;
                'COMMENT' GRAMMARRULE COUNTER;
        'END';
        J:=1;
        'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
        'BEGIN' J:=J+INFOGRAM;
                HELP:=ARRAY[J];
                'FOR' SYM:=1 'STEP' 1 'UNTIL' HELP 'DO'
                'BEGIN' SYMBOL:=ADDRESSKEY[ARRAY[J+SYM]];
                        'IF' SYMBOL=0 'THEN' ERROR('('SYMBOL IN GRAMMAR NOT DEFINED')')
                        'ELSE' ARRAY[J+SYM]:=SYMBOL
                'END';
                J:=J+HELP+1
        'END';
'END' READ GRAMMAR;

'PROCEDURE' READ WORD(BADDRESS,DIRECTION,EADDRESS);
'VALUE'         DIRECTION;'INTEGER' BADDRESS,DIRECTION,EADDRESS;
'BEGIN' 'INTEGER' SYM,HELP;
        BADDRESS:=EADDRESS;
        EADDRESS:=EADDRESS+DIRECTION;
        NEXTSYMBOL;
        'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
        'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ≠ SEPARATOR 'DO'
        'BEGIN' HELP:=ADDRESSKEY[SYM];
                'IF' HELP=0 'THEN' ERROR('('SYMBOL IN WORDNOT IN GRAMMAR')') 'ELSE'
                ARRAY[EADDRESS]:=HELP;
                EADDRESS:=EADDRESS+DIRECTION
        'END';
        ARRAY[BADDRESS]:=EADDRESS-BADDRESS-DIRECTION
'END' READ WORD;

'PROCEDURE' GENERATE BEGINWORD;
'BEGIN' EOM:=END OF MEMORY;
        READ WORD(BW,-1,EOM);
'END' GENERATE BEGINWORD;

'INTEGER' 'PROCEDURE' NEXTSYMBOL;
'BEGIN' 'INTEGER' SYM;
NEXT:   SYM:=RESYM;PRSYM(SYM);
        'IF' SYM=SPACE ∨ SYM=TAB ∨ SYM=TWNR 'THEN' 'GOTO' NEXT
;NEXTSYMBOL:=SYM
```

```
 'END' NEXTSYMBOL;

 'PROCEDURE' GENERATE(ADDRESS);'INTEGER' ADDRESS;
 'BEGIN' 'INTEGER' N,STEP,NEW,K,I,R,M,J;
         N:=ARRAY[ADDRESS];
         'IF' N < 0 'THEN'
         'BEGIN' STEP:=1;NEW:=BOM 'END' 'ELSE'
         'BEGIN' STEP:=-1;NEW:=EOM 'END';
         K:=0;
         'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
         'BEGIN' R:=ARRAY[ADDRESS+I];
                 M:=ARRAY[R];
                 'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO'
                 'BEGIN' K:=K+STEP;
                         ARRAY[NEW+K]:=ARRAY[R+J]
                 'END'
         'END';
         'IF' NEW+K < BOM ∨ NEW+K > EOM 'THEN'  ERROR('('ARRAY MEMORY OVERFLOW')');
         ARRAY[NEW]:=K;
         ADDRESS:=NEW
 'END' GENERATE;

 'PROCEDURE' PRINT WORD(BW);'VALUE' BW;'INTEGER' BW;
 'BEGIN' 'INTEGER' I,N;
         N:=ARRAY[BW];
         NLCR;
         'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL'N 'DO'
         PRSYM(ARRAY[ARRAY[BW+I]+KEY]);
         NLCR;
'END'PRINT WORD;

 'PROCEDURE' READ COUNT;
 'BEGIN' 'INTEGER' SYM;
         NEXTSYMBOL;
         COUNT:=READ(SYM);
         'IF' SYM ≠ SEPARATOR 'THEN' ERROR('('NO SEPARATOR AFTER COUNTCOMMAND')')
 'END' READ COUNT;

 'PROCEDURE' READ PRINTCOMMANDS;
 'BEGIN' 'INTEGER' SYM;
         NEXTSYMBOL;
         SYM:=NEXTSYMBOL;
         'FOR' SYM:=SYM 'WHILE' SYM ≠ SEPARATOR 'DO'
         'IF' SYM=F 'THEN' FIRST:=READ(SYM) 'ELSE'
         'IF' SYM=S 'THEN' STEP:=READ(SYM) 'ELSE'
         'IF' SYM=L 'THEN' LAST:=READ(SYM) 'ELSE'
         ERROR('('ERROR IN PRINTSPECIFICATION')')
 'END' READ PRINTCOMMANDS;

 'INTEGER' 'PROCEDURE' READ(X);'INTEGER' X;
 'BEGIN' 'INTEGER' SYM;
         SYM:=0;
         'FOR' X:=NEXTSYMBOL 'WHILE' DIGIT(X) 'DO'
         SYM:=SYM*10 + X;
         READ:=SYM
 'END' READ;

 'BOOLEAN' 'PROCEDURE' DIGIT(X);'VALUE'X; 'INTEGER'X;
 DIGIT:= X > -1 ∧ X < 10;
```

```

           INITIALIZE;
 START:    JOB:=JOB+1;PRINTER:='TRUE';
                   FIRST:=LAST:=0;
           STEP:=0;
           PRINTTEXT('(' JOBNUMBER: ')');ABSFIXT(2,0,JOB);
           NLCR;
           READ INPUT;
 CARRIAGE(3);
           FROM:=COUNT-LAST;
           'FOR' I:=1 'STEP' 1 'UNTIL' COUNT 'DO'
           'BEGIN' GENERATE(BW);
                   'IF' I<FIRST+1 ∨ I>FROM ∨ I=COUNT ∨ REMAINDER(I-FIRST,STEP)=0
                   'THEN' PRINT WORD(BW)
           'END';
 CARRIAGE(10);
           EOM:=END OF MEMORY + ARRAY[END OF MEMORY] - 1;
           'GOTO' START
  'END'
 'END'
```

```
3008/2-307      A 2317R.33 PAULVITANYI                    5                                                            34

 JOBNUMBER:    1

JOB=GP/
G/ S=>PQ/ P=>V*V*/ V=>TYGER/ *=>,/ Q=>WX/ W=>BURNING/ X=>BRIGHT/ U=>12/ 1=>345/
2=>648/ 3=>IN/ 4=>THE/ 5=>FORESTS/ 6=>OF/ 8=>NIGH/ H=>0/0=>9/ 9=>T /
,=> / A=> /B=> /C=> /D=> /E=> /F=> /G=> /I=> /L=> /M=> /N=> /O=> /R=> /T=> /
Y=> /?=> / /
B/S/
E/9/
P/F9/


PQ

V*V*WX

TYGER,TYGER,BURNINGBRIGHT

120

3456489

INTHEFORESTSOFTHENIGHT

0PQPQ00

9V*V*WXV*V*WX99

TTYGER,TYGER,BURNINGBRIGHTTYGER,TYGER,BURNINGBRIGHTTT




 JOBNUMBER:    2

JOB=GP/
G/ A=>B/ B=>AB //
B/ A/
E/10/
P/ F1 S10 L4/


B

BABABBABABBABBABABBAB

ABBABBABABBABBABABBABABBABBABABBAB

BABABBABABBABBABABBABABBABBABABBABBABABBABABBABBABABBAB

ABBABBABABBABBABABBABABBABBABABBABBABABBABABBABBABABBABABBABABBABBABABBABBABABBABABBABBABABBAB
```

```
300872-307      A 2317R.33 PAULVITANYI                    6                                     34

 JOBNUMBER:   3

JOB=GP/
G/ A=>BC/ B=>KD/ C=> EK/ D=>GB/ E=>CF/ F=>IH/ G=>HI/ H=>DE/ I=>K/ K=>K/ /
B/A/
E/11/
P/L11/


BC

KDEK

KGBCFK

KHIKDEKIHK

KDEKKGBCFKKDEK

KGBCFKKHIKDEKIHKKGBCFK

KHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHK

KDEKKGBCFKKDEKKGBCFKKHIKDEKIHKKGBCFKKDEKKGBCFKKDEK

KGBCFKKHIKDEKIHKKGBCFKKHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHKKGBCFKKHIKDEKIHKKGBCFK

KHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHKKDEKKGBCFKKDEKKGBCFKKHIKDEKIHKKGBCFKKDEKKGBCFKKDEKKHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHK

KDEKKGBCFKKDEKKGBCFKKHIKDEKIHKKGBCFKKDEKKGBCFKKDEKKGBCFKKHIKDEKIHKKGBCFKKHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHKKGBCFKKHIKDEKIHKKGBCFKKDEKKGBCFKKDEKKG
BCFKKHIKDEKIHKKGBCFKKDEKKGBCFKKDEK




 JOBNUMBER:   4

JOB=GP/
G/A=>BAC/ C=>CC/ B=>BB/ /
B/A/
E/8/
P//

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
3008/2-307      A 2317R.33 PAULVITANYI                    7                                          34

 JOBNUMBER:   5

JOB=GP/
G/ A=>ABCD/ B=>BD/ C=>CD/ D=>D//
B/A/
E/10/
P/ F3 S7/


ABCD

ABCDBDCDD

ABCDBDCDDBDDCDDD

ABCDBDCDDBDDCDDDBDDDCDDDDBDDDDCDDDDDBDDDDDCDDDDDDBDDDDDDCDDDDDDDBDDDDDDDCDDDDDDDDBDDDDDDDDCDDDDDDDDDBDDDDDDDDDCDDDDDDDDDDBDDDDDDDDDDCDDDDDDDDDDD




 JOBNUMBER:   6

JOB=GP/
G/  K=>L/ L=>M/ M=>K/ 1=>2/ 2=>3/ 3=>4/ 4=>5/ 5=>6/ 6=>7/ 7=>1/ +=>-/ -=>+/
A=>B/ B=>C/ C=>D/ D=>E/ E=>A/ /
B/L3+A4/
E/210/
P/ F4 S101 L5/


M4-B5

K5+C6

L6-D7

M7+E1

L3-A4

K6+B7

L7-C1

M1+D2

K2-E3

L3+A4
```

```
3008/2-307      A 2317R.33 PAULVITANYI                  8                           34

 JOBNUMBER:    7

JOB=GP /G/1=>2*4/2=>2/4=>2*5/5=>6#5/6=>7/7=>8/8=>9(1)/9=>9/(=>(/)=>)/ *=>*/
#=>#//B/1/
E/18/
P/ F12 S3 L1/


2*4

2*2*5

2*2*6#5

2*2*7#6#5

2*2*8#7#6#5

2*2*9(1)#8#7#6#5

2*2*9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*9(1)#8#7#6#5)#9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*4)#9(1)#8#7#6#5)#9(2*2*9(1)#8#7#6#5)#9(2*2*8#7#
6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5

2*2*9(2*2*9(2*2*9(1)#8#7#6#5)#9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#
9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9
(2*2*5)#9(2*4)#9(1)#8#7#6#5)#9(2*2*9(2*4)#9(1)#8#7#6#5)#9(2*2*9(1)#8#7#6#5)#9(2*2*8#7#6#5)#9(2*2*7#6#5)#9(2*2*6#5)#9(2*2*5)#9(2*4)#9(1)#8#7#6#5




 JOBNUMBER:    8

JOB= /

PROGRAM END
```

stichting
mathematisch
centrum

MC

REKENAFDELING NR 24/72 APRIL

P.M.B. VITÁNYI

CONTEXT-VARIABLE LINDENMAYER SYSTEMS AND SOME SIMPLE REGENERATIVE STRUCTURES

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat 49, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Contents

0. Introduction

An automata theoretic model for developmental growth in filamenteous organisms has been proposed by Lindenmayer (1968). In a (k,l) L-System we rewrite every letter of a string simultaneously according to its context, consisting of the k left and l right letters. Here we shall introduce Context-Variable Lindenmayer-Systems, where a letter of a string is rewritten according to a selection of letters from that string. The criterion for the selection is an attribute of the letter concerned. These Systems will appear to be especially suited to model certain properties of "full-growthness" and "regeneration". The accompanying languages are called Context-Variable languages.

1. Context-Variable Lindenmayer Systems

Differences between Chomsky generative grammars and Lindenmayer Systems as language generators are:

(i) In the former one letter of a string is rewritten in each time step while in the latter all letters are rewritten simultaneously.

(ii) In the Chomsky approach only terminal strings are elements of the language while in L-Systems all strings derived are elements of the language, i.e. no distinction is made between terminal and non terminal letters.

The main feature that distinguishes Context-Variable L-Systems from (k,l) ones is that in Context-Variable L-Systems the relative place of the context of a letter can vary from time to time and from place to place. This feature makes the concept difficult to handle but we shall give some simple examples below. In these examples the Systems seem to strive at attaining a certain full-grown size and structure, which, however, is not terminal. Cells, i.e. letters, are changing state, dividing, and dying all the time. When we chop off a piece we observe a certain regenerative behavior.

Def. 1.1. A Context-Variable Lindenmayer System or C-V L-System is a 3-tuple $G = <\sum,\delta,\sigma>$ such that

(i) The alphabet $\sum$ is a nonempty finite set and elements of $\sum$ are called letters.

(ii) The transition function δ maps strings $x \in \sum^+$ onto strings $y \in \sum^*$ such that each element b_j of y has a superscript $\tau_j \in I^*$, i.e.

$$\delta(a_1a_2\ldots a_n) = b_1^{\tau_1}b_2^{\tau_2} \ldots b_m^{\tau_m}$$

where

$$x = a_1a_2 \ldots a_n$$

$$y = b_1^{\tau_1}b_2^{\tau_2} \ldots b_m^{\tau_m}$$

$$\tau_j = \rho_0^{(j)}\rho_1^{(j)} \ldots \rho_{n_j}^{(j)} \qquad 1 \leq j \leq m$$

with

$$a_i,b_j \in \sum$$

$$\rho_h^{(j)} \in I \qquad \text{with } 1 \leq i \leq n,\ 1 \leq j \leq m,\ 0 \leq h \leq n_j.$$

In the above definition δ is deterministic; the generalization to the non deterministic case is done in the obvious way. In this report we shall only be concerned with the deterministic case.

(iii) The axiom σ is a word over $\sum$, each letter possessing a superscript which is a string over I, i.e.

$$\sigma = a_1^{\tau_1}a_2^{\tau_2} \ldots a_m^{\tau_m}$$

where

$$\tau_j = \rho_0^{(j)}\rho_1^{(j)} \ldots \rho_{n_j}^{(j)} \in I^*$$

$$a_j \in \sum \qquad 1 \leq j \leq m.$$

We also call the axiom the initial description of the C-V L-System.

Remark 1. The superscript $\tau_j = \rho_0^{(j)}\rho_1^{(j)} \dots \rho_{n_j}^{(j)}$ selects in string $b_1^{\cdot}b_2^{\cdot} \dots b_j^{\cdot} \dots b_m^{\cdot}$ the context $h(b_j)$ according to which b_j is going to be rewritten:

$$h(b_j) = b_{j+\rho_0^{(j)}}\, b_{j+\rho_1^{(j)}} \dots b_{j+\rho_{n_j}^{(j)}} \ .$$

If $j + \rho_i^{(j)} \leq 0$ or if $j + \rho_i^{(j)} > m$ we substitute the empty word λ for $b_{j+\rho_i^{(j)}}$ in $h(b_j)$. We will henceforth assume that $\rho_0^{(j)} = 0$ and omit $\rho_0^{(j)}$ from the superscript of b_j .

The C-V L-System generates words as follows:
Let $x = a_1^{\tau_1} a_2^{\tau_2} \dots a_m^{\tau_m}$ be a string. Then x generates y directly, written as $x \Longrightarrow y$, if

$$x = a_1^{\tau_1} a_2^{\tau_2} \dots a_m^{\tau_m}$$

$$y = \alpha_1\alpha_2 \dots \alpha_m$$

and for every j, $1 \leq j \leq m$,

$$\alpha_j = \delta(a_j\, a_{j+\rho_1^{(j)}}\, a_{j+\rho_2^{(j)}} \dots a_{j+\rho_{n_j}^{(j)}})$$

with

$$\rho_1^{(j)}\rho_2^{(j)} \dots \rho_{n_j}^{(j)} = \tau_j \ .$$

$\overset{*}{\Longrightarrow}$ denotes the reflexive and transitive closure of $\Longrightarrow$ and $x \xRightarrow{*(k)} y$ denotes a chain of length k:

$$x = x_0 \Longrightarrow x_1 \Longrightarrow \dots \Longrightarrow x_k = y.$$

If $x \overset{*}{\Longrightarrow} y$ we say x produces, generates or derives y, and if $x \xRightarrow{*(k)} y$ then y is derived in k steps from x and $x \xRightarrow{*(k)} y$ is a k-derivation of y from x. A string $x = a_1^{\tau_1} a_2^{\tau_2} \dots a_m^{\tau_m}$ is called a description, and an element of x is called a cell.

<u>Def. 1.2.</u> A <u>C-V L-Language</u> is a set $L(G) \subseteq \sum^*$ where

$$L(G) = \{a_1 \ldots a_n \mid \sigma \overset{*}{\Longrightarrow} a_1^{\cdot} \ldots a_n^{\cdot}\} .$$

<u>Example 1</u>

Let

$$G = <\{a\}, \{a \rightarrow a^{-1}a^{+1}, aa \rightarrow \lambda\}, a> .$$

Then

$$a \Longrightarrow a^{-1}a^{+1} \Longrightarrow a^{-1}a^{+1}a^{-1}a^{+1} \Longrightarrow a^{-1}a^{+1}a^{-1}a^{+1}$$

$$\Longrightarrow \ldots \Longrightarrow a^{-1}a^{+1}a^{-1}a^{+1} \Longrightarrow \ldots .$$

We notice that when the description has reached a certain full-grown size it does not change any more although the individual letters certainly are not terminal or static, i.e. letters are dividing and dying all the time but the structure, complete with context relations, stays unaltered.

The language generated by this example is

$$L(G) = \{a, aa, aaaa\}.$$

Let $G(k) = <\{a\}, \{a \rightarrow a^{-k}a^{+k}, aa \rightarrow \lambda\}, a>$.

The language produced by G(k) shall be called $L^a(k)$.

Then $L^a(1) = \{a, aa, aaaa\}$.

In a similar way we obtain

$$L^a(2) = \{a, aa, aaaa\}$$

$$L^a(3) = \{a, aa, aaaa, aaaaaaaa\}$$

$$L^a(4) = L^a(3)$$

$$L^a(5) = \{a^n \mid n = 1,2,4,8,12\}$$

$$L^a(6) = L^a(5)$$

etc.

$L^a(0) = \{\lambda, a\}$

$L^a(-1) = \{\lambda, a, aa\}$

$L^a(-2) = \{a, aa, aaaa\}$

$L^a(-3) = L^a(-2)$

etc.

We describe the general form of an $L^a(k)$-language by:

Theorem 1. Let $G(k)$ and $L^a(k)$ be as above.

a) For $k > 0$ and k is even

$$L^a(k) = \{a^{2^t} \mid 0 \leq t \leq {}^2\log(k) + 1\} \cup \{a^{2k}\}.$$

For $k > 0$ and k is odd

$$L^a(k) = L^a(k+1).$$

b) For $k < -1$

$$L^a(k) = L^a(-k-1).$$

c) $L^a(0) = \{\lambda, a\}$

$L^a(-1) = \{\lambda, a, aa\}.$

Proof. By $\delta^t(a)$ we mean $\dot{a}_1\dot{a}_2 \ldots \dot{a}_n$ if

$$a \xRightarrow{*(t)} \dot{a}_1\dot{a}_2 \ldots \dot{a}_n .$$

a) For $k > 0$

(i) $t \leq {}^2\log(k)$. $|\delta^t(a)| = 2^t \leq k$. *)

*) $|x|$ denotes the length of x.

There are no cells $a_i^{\pm k}$ in $\delta^t(a)$ such that production rule $aa \to \lambda$ is to be applied. Therefore all cells divide and $|\delta^{t+1}(a)| = 2^{t+1}$.

(ii) $\quad {}^2\log(k) < t \leq {}^2\log(k) + 1$.

For all cells a_{2i}^{+k} and $a_{2^t-2i+1}^{-k}$ $(i>0)$, such that $2i+k \leq 2^t$, production rule $aa \to \lambda$ will be applied. Let $j = \max_{2i+k\leq 2^t}(i)$; then there are 2j cells in $\delta^t(a)$ such that $aa \to \lambda$ will be the applied production rule. For k is even: $2j+k = 2^t$ or $2^t-2j = k$. 2j cells disappear and k cells divide in the next production, so $|\delta^{t+1}(a)| = 2k$. For k is odd $2j+k = 2^t-1$ or $2^t-2j = k+1$. 2j cells disappear and k+1 cells divide in the next production, so $|\delta^{t+1}(a)| = 2k+2$.

(iii) $\quad t > {}^2\log(k) + 1$.

The last production gave us $|\delta^t(a)| = 2k$ (k even), so half of the cells divide and the other half disappears in the next production: $|\delta^{t+1}(a)| = 2k$. For k is odd we get $|\delta^{t+1}(a)| = 2k+2$.

b) is proven in a similar way as a).

c) follows from the productions.

<u>Corollary</u> $\quad \bigcup_{k\in I} L^a(k) = \{a^{4n} \mid n \geq 0\} \cup \{a, aa\}$.

The C-V L-Systems we have been considering all start from a single cell, and, according to the predetermined genetical instructions (i.e. δ and the specification of k),they grow at an exponential rate until the full-grown size is reached but for one move. Next the C-V L-System grows on the remainder and stays at the same size and structure, although at each generation individual cells disappear and divide. Note that there is a limited interaction all the time between the cells to achieve this goal.

We can investigate regenerative processes in these systems, by removing part of the (full-grown or growing) description. The missing part then is regrown again. When we divide a description into several parts, all of these will eventually reach a full-grown stage. This is reminiscent of the remarkable regenerative properties of flatworms. The discussed C-V L-Systems are very simple, i.e. there is no differentiation of cells. It would be interesting, to investigate similar regenerative processes in more complex C-V L-Systems, e.g. with more cellular states. Does there exist a complexity bound, e.g. expressed in the size of the alphabet (and presumably δ), above which only partial regeneration is possible?
We may qualify questions like this by distinguishing several kinds of regeneration, e.g.

(i) Starting with one cell in a special state, i.e. reproduction.

(ii) Starting from arbitrary parts of a full grown description.

(iii) Starting from arbitrary parts of a description at some stage of the growth process.

(iv) Starting from selected parts removed from the full grown description, etcetera.

Note that there is a difference between cases where we remove an end part of a full-grown description, and cases where we remove a middle part. We illustrate this with the following example (k=2).
The full grown description is:

$$a^{-2}a^{+2}a^{-2}a^{+2}.$$

Regeneration with the left-end (skin) cell removed:

$$a^{+2}a^{-2}a^{+2} \Longrightarrow a^{-2}a^{+2}a^{-2}a^{+2}.$$

The two cells right have divided, while the new leftmost cell has disappeared in the production. Regeneration with the third (middle) cell removed:

$$a^{-2}a^{+2}a^{+2} \Longrightarrow a^{-2}a^{+2}a^{-2}a^{+2}a^{-2}a^{+2} \Longrightarrow a^{-2}a^{+2}a^{-2}a^{+2} .$$

All three cells divide in the first production. In the second production only the two outermost cells divide and the others disappear: the full-grown size is reached.

We observe that the removal of different parts of the full-grown description may yield different courses for the regenerative process. The above is suggestive of biological interpretations like the surrounding of a wound by wound-tissue which is discarded after the healing process has been completed.

In the appendix we shall consider some closure (or rather non-closure) properties of $L^a(k)$ languages, so as to get an insight into what place the considered structures take with respect to the other language generating devices.

2. The Extended French Flag Problem

Usually the French Flag problem is states as follows: suppose we have a string of cells all of which are in an identical state but because of some disturbance produce the pattern of a French Flag, i.e. one third red, one third white and one third blue. Moreover, when we cut off any piece of it which is large enough it produces this pattern again.

The above is supposed to be (e.g. Herman, 1972) a meaningful statement of problems of biological regeneration. However, as we have stated before, what seems more meaningful is the design of structures which, starting from a single cell,attain a certain full-grown stage, no cell staying static, and furthermore, when we chop off a piece of this structure regrow the missing piece until the full-grown stage has been reached again.

When we discuss the French Flag in this context what we want is:

(i) One cell divides and gives rise to a full-grown French Flag of a certain size which retains the same pattern and structure while individual cells are disappearing and dividing all the time.

(ii) When we chop off a piece of the full-grown French Flag it regrows the missing piece.

We will present a C-V L-System which does (i) and (ii).

As the system has to reach a certain full-grown size, clearly the production rules depend on this size. When we want a different full-grown size we will have to find a new set of productions.
Furthermore, in the discussed system the a's serve as some kind of "head" of the structure, i.e. the front part always regenerates a new end part but an end part does not always regenerate a new front part. When part of the head is contained in it, however, it does. The biological interpretation of this phenomenon is so obvious (lizards!) that such a kind of partial regeneration has not to be justified further. We may point out that "higher" organisms which are more differentiated mostly lose regenerative properties to a certain extend which seems to be the price to be paid for a more complex structure.
(Is there a maximal number of letters above which unlimited regeneration is not possible anymore? What about other types of regeneration?)
We shall exhibit an example of a Context Variable Lindenmayer System with maximal a two neighbor context, which, starting from a single cell attains a full-grown description, i.e. the French Flag

$$a^{\circ}a^{\bullet}a^{\circ}a^{\circ}b^{\bullet}b^{\bullet}b^{\bullet}b^{\bullet}c^{\bullet}c^{\bullet}c^{\bullet}c^{\circ} \ .$$

When this French Flag is cut, the left part always regenerates completely; the right part mostly not, depending on where the cut was placed.
We will call $a^{\bullet}a^{\bullet}a^{\bullet}a^{\circ}$ the <u>head</u>, $b^{\bullet}b^{\circ}b^{\circ}b^{\bullet}$ the <u>trunk</u> and $c^{\bullet}c^{\bullet}c^{\bullet}c^{\circ}$ the <u>tail</u> of the French Flag.

$\Sigma = \{a, b, c\}$. The transition function is specified by the following rules (we only write those we need and leave the others open):

$$a \rightarrow a^{-1+1} b^{-1+1}$$

$$b \rightarrow b^{-1+1} c^{+1-1}$$

$$c \rightarrow c^{+1-1} c^{+1-1}$$

$aa \to \lambda$

$bb \to \lambda$

$cc \to c^{+1-1}c^{+1-1}$

$ab \to a^{+2+1}a^{-1+1}$

$bc \to b^{+2+1}b^{-1+1}$

$ba \to b^{-1+1}c^{+1-1}$

$cb \to c^{+1-1}c^{+1-1}$

$aaa \to a^{-1+1}a^{-2+2}$

$bbb \to b^{-1+1}b^{-2+2}$

$ccc \to \lambda$

$aab \to a^{-1+1}a^{-1+1}$

$bbc \to b^{-1+1}b^{-1+1}$

$aba \to a^{-1+1}a^{-2+2}$

$bcb \to b^{-1+1}b^{-2+2}$

$cbc \to c^{+1-1}c^{+1-1}$

$bac \to b^{+2+1}b^{-1+1}$

$ccb \to c^{+1-1}c^{+1-1}$

$bab \to \lambda$

$bcc \to \lambda$

$cbb \to \lambda$

$cba \to \lambda$

$acb \to \lambda$

Starting from axiom a we obtain the following production:

$$(1)\quad a \Longrightarrow a^{-1+1}b^{-1+1} \Longrightarrow a^{+2+1}a^{-1+1}b^{-1+1}c^{+1-1}$$

$$\Longrightarrow a^{-1+1}a^{-2+2}a^{-1+1}a^{-1+1}b^{+2+1}b^{-1+1}c^{+1-1}c^{+1-1}$$

$$\Longrightarrow a^{-1+1}a^{-2+2}a^{-1+1}a^{-1+1}b^{-1+1}b^{-2+2}b^{-1+1}b^{-1+1}c^{+1-1}c^{+1-1}c^{+1-1}c^{+1-1}$$

$$\Longrightarrow a^{-1+1}a^{-2+2}a^{-1+1}a^{-1+1}b^{-1+1}b^{-2+2}b^{-1+1}b^{-1+1}c^{+1-1}c^{+1-1}c^{+1-1}c^{+1-1}$$

$$\Longrightarrow \text{idem.}$$

We call this full-grown description FF, and observe that FF is the desired French Flag; it stays at this structure although the individual cells are dividing and dying off continuously. Note that the head grows fastest and is completed first.

Next we investigate the regenerative properties.

There are eleven places at which FF can be cut.

When we look at the left part resulting from such a cut we see:

(N.B. We will sometimes omit superscripts when no confusion can result, e.g. a^4b^{-1+1} for $a^{-1+1}a^{-2+2}a^{-1+1}a^{-1+1}b^{-1+1}$.)

$$(2.1)\quad a^{-1+1} \Longrightarrow a^{-1+1}b^{-1+1} \overset{*}{\Longrightarrow} \text{FF by (1)}$$

$$(2.2)\quad a^{-1+1}a^{-2+2} \Longrightarrow a^{-1+1}b^{-1+1} \overset{*}{\Longrightarrow} \text{FF by (1)}$$

$$(2.3)\quad a^{-1+1}a^{-2+2}a^{-1+1} \Longrightarrow a^{-1+1}b^{-1+1} \overset{*}{\Longrightarrow} \text{FF by (1)}$$

$$(2.4)\quad a^{-1+1}a^{-2+2}a^{-1+1}a^{-1+1} \Longrightarrow a^{-1+1}a^{-2+2} \overset{*}{\Longrightarrow} \text{FF by (2.2)}$$

$$(2.5)\quad a^4b^{-1+1} \Longrightarrow a^4b^{-1+1}c^{+1-1} \Longrightarrow a^4b^{+2+1}b^{-1+1}c^{+1-1}c^{+1-1} \Longrightarrow \text{FF}$$

$$(2.6)\quad a^4b^{-1+1}b^{-2+2} \Longrightarrow a^4b^{-1+1}c^{+1-1} \overset{*}{\Longrightarrow} \text{FF by (2.5)}$$

$$(2.7)\quad a^4b^{-1+1}b^{-2+2}b^{-1+1} \Longrightarrow a^4b^{-1+1}c^{+1-1} \overset{*}{\Longrightarrow} \text{FF by (2.5)}$$

(2.8) $a^4b^{-1+1}b^{-2+2}b^{-1+1}b^{-1+1} \Longrightarrow a^4b^{-1+1}b^{-2+2} \overset{*}{\Longrightarrow}$ FF by (2.6)

(2.9) $a^4b^4c^{+1-1} \Longrightarrow a^4b^4c^{+1-1}c^{+1-1} \Longrightarrow$ FF

(2.10) $a^4b^4c^{+1-1}c^{+1-1} \Longrightarrow$ FF

(2.11) $a^4b^4c^{+1-1}c^{+1-1}c^{+1-1} \Longrightarrow$ FF.

Hence all left parts regenerate completely.
The reader may verify that the full-grown descriptions reached by the right parts are according to (3.1) - (3.11) (when the cuts are placed as in (2.1) - (2.11)).

(3.1) $a^3b^4c^4 \Longrightarrow$ FF

(3.2) $a^2b^4c^4 \Longrightarrow a^2b^4c^4$

(3.3) $a\,b^4c^4 \overset{*}{\Longrightarrow}$ FF

(3.4) $b^4c^4 \Longrightarrow b^4c^4$

(3.5) $b^3c^4 \Longrightarrow b^4c^4$

(3.6) $b^2c^4 \Longrightarrow b^2c^4$

(3.7) $b\,c^4 \overset{*}{\Longrightarrow} b^4c^4$

(3.8) $c^4 \Longrightarrow c^4$

(3.9) $c^3 \Longrightarrow c^4$

(3.10) $c^2 \Longrightarrow c^4$

(3.11) $c \overset{*}{\Longrightarrow} c^4$.

We may also cut a piece out of the middle of FF. It may be verified that

(4.1) Every part of FF containing cells of the head regenerates completely to FF except parts of the form

$$a^{-1+1}a^{-1+1}\eta$$

(i) $a^{-1+1}a^{-1+1} \Longrightarrow \lambda$

(ii) $a^{-1+1}a^{-1+1}\eta \overset{*}{\Longrightarrow} a^2b^4c^4$ for $\eta \neq \lambda$.

(4.2) Every part of FF containing cells of the trunk but no head cells grows to a full-grown description b^4c^4 except parts of the form

$$b^{-1+1}b^{-1+1}\eta$$

(i) $b^{-1+1}b^{-1+1} \Longrightarrow \lambda$

(ii) $b^{-1+1}b^{-1+1}\eta \overset{*}{\Longrightarrow} b^2c^4$ for $\eta \neq \lambda$.

(4.3) Every part of FF consisting of tail cells grows to a full tail c^4, i.e. a full-grown description.

3. Open Problems

Def. 3.1. A C-V L-System $G = <\sum, \delta, \sigma>$ stabilizes at ω if ω is the full-grown description of G.

A C-V production scheme is a pair $S = <\sum, \delta>$.

Def. 3.2. A C-V production scheme $S = <\sum, \delta>$ stabilizes at $\omega \in \sum^*$ if for all $\sigma \in \sum^*$the C-V L-System $G = <\sum, \delta, \sigma>$ stabilizes at ω.

1. Given an $\omega \in \sum^*$, does there always exist a C-V production scheme that stabilizes at ω. Find an algorithm which produces such a C-V production scheme.

2. If the answer to 1 is begative in general, then characterize the class (or a sub-class) for which the answer is positive.

A sub-class as meant in 2 is e.g.

$$\{a^{4n} \mid n \geq 0\}.$$

By example 1 $G(k)$, k is an even natural number, stabilizes at a^{2k} for every axiom $\sigma \in \{a\}^*$.

3. Given $\omega \in \sum^*$, does there always exist a C-V L-System $G = \langle\sum', \delta, a\rangle$, $\sum' \supseteq \sum$ and $a \in \sum$, such that G stabilizes at ω. Give an algorithm to obtain such a G. (Can every word be generated by a C-V L-System with a one letter axiom such that the word is a full-grown description of that C-V L-System.)

4. If the answer to 3 is negative, then characterize the class (or a sub-class) for which the answer is positive.

Again, $\{a^{4n} \mid n \geq 0\}$ is such a sub-class.

5. Given $\omega \in \sum^*$, does there always exist a production scheme $S = \langle\sum', \delta\rangle$, $\sum' \supseteq \sum$, such that for all σ, $\omega = \eta\sigma\xi$, $G = \langle\sum', \delta, \sigma\rangle$ stabilizes at ω. (Is universal regeneration possible for every word?) If not, characterize the class (or a sub-class) for which the answer is positive.

$\{a^{4n} \mid n \geq 0\}$ is such a sub-class.

One criterion for the finiteness of C-V L-Languages is whether the produced description ever stabilizes.

6. Can we indicate conditions under which a C-V L-System stabilizes.

Appendix

Theorem 2. The family of $L^a(k)$ languages is not closed under (i) complementation, (ii) union, (iii) Kleenean star (*), (iv) Kleenean cross (+), (v) concatenation, (vi) intersection with regular sets; but it is closed under (vii) intersection.

Proof.

(i) $\{a\}^* \setminus L^a(1)$ contains aaa, and aaa $\notin \bigcup_{k\in I} L^a(k)$.

(ii) $L^a(6) \cup L^a(10) = \{a^n \mid n = 1,2,4,8,12,16,20\}$. From theorem 1 follows $L^a(k) \neq L^a(6) \cup L^a(10)$ for all k.

(iii) $L^a(k)^*$ contains aaa and aaa $\notin \bigcup_{k\in I} L^a(k)$.

(iv) as (iii).

(v) $L(1) \,.\, L(1) = \{a^n \mid n = 2,3,4,5,6,8\} \neq L^a(k)$ for all k.

(vi) $L^a(1) \cap \{aaaa\} = \{aaaa\} \neq L^a(k)$ for all k.

(vii) $L^a(k_1) \cap L^a(k_2) = \{a^{2^t} \mid 0 \leq t \leq \min({}^2\log(k_1), {}^2\log(k_2)) + 1\} = L^a(k)$ for $k = \max\{2^{t-1} \mid 2^t \leq \min(2k_1, 2k_2)\}$.

Lemma 3. The family of $L^a(k)$ languages is strictly contained in the family of regular languages over a one letter alphabet.

Proof. All $L^a(k)$ languages are finite.

Lemma 4. The intersection of the family of $L^a(k)$ languages with the family of OL-languages [Rozenberg & Doucet, 1971] consists of those $L^a(k)$ languages for which $L^a(k) \neq L^a(-k-1)$, viz. $\{L^a(0), L^a(-1)\}$.

Proof. Consider the following OL-Systems:

$$S_1 = \langle\{a\}, \{a \to a, a \to \lambda\}, a^h\rangle$$

then

$$L(S_1) = \{a^t \mid 0 \leq t \leq h\}$$

and

$$L(S_1) = L^a(-1) \qquad \text{for } h = 2$$

$$L(S_1) = L^a(0) \qquad \text{for } h = 1.$$

If $h > 2$ then

$$a^3 \in L(S_1) \quad \text{and} \quad a^3 \notin L^a(h) \qquad \text{for all } h.$$

$$S_2 = \langle\{a\}, \{a \to a\}, a^h\rangle$$

$$L(S_2) = \{a^h\} \text{ and } |L(S)| = 1 \neq |L^a(k)| \quad \text{for all } k.$$

$$S_3 = \langle\{a\}, \{a \to \lambda\}, a^h\rangle$$

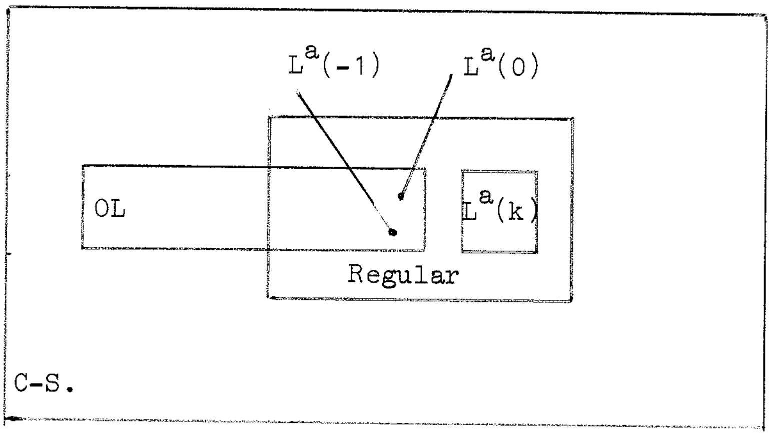
$$L(S_3) = \{a^h, \lambda\} = L^a(0) \qquad \text{for } h = 1.$$

$$L(S_3) \neq L^a(0) \qquad \text{for all } h > 1$$

and

$$|L(S_3)| = 2 \neq L^a(k) \qquad \text{for } k \neq 0.$$

All other OL-Systems over one letter alphabet have a production $a \to a^x$ where $x > 1$, and therefore generate an infinite language.



Remark. $L^a(k)$ languages are finite (containing usually more than 2 elements) and are generated in a deterministic fashion. It is not possible to generate finite languages containing more than two elements deterministically by either formal grammars or OL-Systems.

References

G.T. Herman, Polar organisms with apolar individual cells, in: IV Int. Congress for Logic, Methodology and Philosophy of Science, 1971 (in press).

A. Lindenmayer, Mathematical models for cellular interaction in development I & II, J. Theoret. Biol. 18(1968), 280-315.

G. Rozenberg & P.G. Doucet, On OL-languages, Information & Control 19(1971), 302-318.

CELIA MANUAL by Adrian Walker

GENERAL DESCRIPTION

CELIA (CEllular Linear Iterative Array simulator) is a special purpose simulator written in FORTRAN which simulates the expansion of filaments by Lindenmayer models. Information about the filament to be expanded and the control of the simulation are specified by the user with CELIA control and data cards and subprograms. A general description of an earlier version of the program can be found in Baker & Herman "Simulation of organisms using a developmantal model", International Journal of Biomedical Computing, in press. The present program incorporates some additional features.

The delta function (or expansion rules) is provided by the user in the form of either a subroutine (subroutine DELTA) or a table. If a subroutine is used, the user has the ability to change the state of any cell in the filament or even the whole filament at any time during expansion and to do some statistical survey on any attribute of the filament.

The states of cells in a filament are specified by a number of attributes. Attributes specifying the state of a cell in a filament can be real numbers or alphanumerical character strings. CELIA can handle a filament containing up to 1200 cells (with 2 attributes at most) or cells containing up to 100 attributes (with at most about 24 cells in a filament). Generally, the product of the number of cells and the number of attributes can be up to 2400. If the storage space is not large enough for the simulation, an error message will be printed out.

The I/O formats and arrangements are specified by the user. The user can also print or punch out the filament at certain moments.

The simulation is done in such a way that CELIA interprets the control statements step by step. If the statement being interpreted is a specification, then that specification will be used from that point on, and the previous specification on the same feature is overridden by the current one. Therefore, the user can change the situation from time to time.

2

CELIA CONTROL AND DATA CARDS

CELIA control and data cards (or control cards together with data cards) are prepared by the user and are read in as input during execution to control the execution of CELIA. Control cards specify the control statements, while data cards specify additional information which is necessary for the simulation. Every data card (or set of data cards) belongs to one of the user-written control statements and must appear immediately after that control statement. Not every control statement has data card(s) attached to it.

Of the 13 control statements of the current CELIA, 12 of them are actually specifications or declarations. Only one statement (EXECUTE) is executable. The command EXECUTE means expanding the filament under a specified situation. To expand filaments under different situations from time to time, the user just repeats this EXECUTE statement as many times as he wants, with some declarative statements put in between to change the situation. Note that the user need not specify everything. Declarative statements specifying the situation are necessary only when the specified information is needed and is different from that assumed by default.

Before any EXECUTE statement is interpreted, all the information concerned with that EXECUTE statement should be complete. Otherwise CELIA will use the previously specified information or assume default values (if not specified by the user at all) during the expansion of the filament. The order of the control statements specifying the situation is irrelevant except for some logical reasons. For example, input format should have been specified before CELIA reads in the original filament.

Each control statement consists of one instruction command and two data fields. They are Command, Field A, and Field B. The format of CELIA control cards is

cc 1-10	Command
cc 21-30	Field A
cc 31-80	Field B

In general, the Command field and Field B consist of character strings, Field A consists of numbers. All numbers and character strings in these fields should be left justified. Illegal statements will cause error messages, but the expansion of the filament will still go on with assumed default specifications unless there is no filament to be expanded. If the first column of a card is "*" or blank, it will be treated as a comment card.

The 8 control statements which do not need attached data cards are the following:

1. Command: NATT

This command tells CELIA how many attributes are used to specify the state of a cell.

Field A: Number of attributes specifying the state of a cell.

Field B: Not used.

Default: Field A = 1

2. Command: I-FORMAT

This statement specifies the input format for CELIA to read in (if needed) the original filament to be expanded, left and right environments, dead state specification and delta function table from data cards.

Field A: Maxium number of cells the user wants to specify in a data card. If the state of a cell is specified by more than one card, put 1 here.

Field B: FORTRAN format specifying the structure of each cell. Note that no I-format is allowed in this format specification and also that a pair of parentheses should be included.

Default: Field A = 80

Field B = "(A1)"

Example: I-FORMAT 15 $\overbrace{\text{(2A1,1X,F1.0,1X)}}^{\text{format for 1 cell}}$

means that there are at most 15 cells specified in an input card, each cell occupies 5 columns. The first two attributes are represented by A1 and the third attribute is represented by F1.0. (In this case, the number of attributes is 3.)

5

3. Command: O-FORMAT

This statement specifies the output format for CELIA to print out the filament.

Field A: Number of cells to be printed on an output line.

Field B: FORTRAN format specifying the structure of each cell appearing in output. The restrictions are the same as those of I-FORMAT.

Default: Field A = 100

Field B = "(A1)"

4. Command: P-FORMAT

This statement specifies the format for CELIA to punch out the filament.

Field A: Number of cells to be punched on a card.

Field B: The same as those of I-FORMAT and O-FORMAT.

Default: Field A = 80

Field B = "(A1)"

5. Command: O-REQUEST

This command asks CELIA to print out the filament according to O-FORMAT at the moments specified by the coded number in Field A.

Field A: 0 Print original and final filament only. Also print the dead filament if it becomes dead during expansion.

1 Print the filament only when the variable OFLAG is set by the user.

2 Print the filament only when the user-written function KPRINT(T) is equal to 1.

3 Case 1 OR case 2.

4 Print every filament during expansion.

Field B: If "NOGAP", filaments will be printed consecutively without any gap between two filaments.

If "SINGLE", a blank line will appear between two consecutive filaments.

If "DOUBLE", two blank lines will appear between two consecutive filaments.

If "ENV-XXX" ("ENV-NOGAP", "ENV-SINGLE", or "ENV-DOUBLE"), it means the same as XXX Blanks ? except that the left and right environments at that moment will also be printed out together with the cells of the filament.

Default: Field A = 0

Field B = "SINGLE"

6. Command: P-REQUEST

This command asks CELIA to punch the current filament in P-FORMAT at the moments specified by the coded number in Field A.

Field A: 0 Nothing will be punched.

1 Punch final filament only.

2 Punch dead filament only.

3 Case 1 OR case 2.

4 Punch out the filament only when the user-written function KPUNCH(T) is equal to 1.

Field B: Not used.

Default: Field A = 0

7. Command: EXECUTE

This command initiates the expansion of the original filament specified by the user. If no original filament has been specified before this command, an error message will be printed out. If no original filament has been specified between two EXECUTE statements, the final filament of the first execution will be treated as the original filament of the second execution.

Field A: Number of iterations to expand the filament while the filament is still alive.

Field B: Not used.

Default: If Field A is blank, Field A = 30.

8. Command: END

This command tells CELIA to terminate the simulation.

Field A: Not used.

Field B: Not used.

Default: END assumed when EOF card encountered.

Each of the following 5 control statements needs data card (or cards) attached immediately after it.

9. Command: ENVIR

This command specifies the left and right environments of the model.

Field A: Not used.

Field B: If "FIXED", the environments are fixed during the expansion of the filament. The data card specifying the left and right environments (left first, then right, specify consecutively on the card) should follow this control card using I-FORMAT.

If "SUBROUTINE", the environments vary according to the user-written subroutine ENVIRN(T). In the latter case, no data cards are needed.

Default: Field B = "FIXED"

No environments assumed.

10. Command: D-STATE

This command specifies the dead state of the cells.

Field A: Not used.

Field B: If "FIXED", the dead state is fixed thoughout the expansion of the filament. The specification card should be attached immediately after the D-STATE card using I-FORMAT.

If "SUBROUTINE", dead state varies according to the user-written subroutine DEADST(T). In this case, no data cards are necessary.

Default: No dead state assumed, i.e., the filament is always alive.

11. Command: D-MASK

Sometimes only some of the attributes are relevant in determining whether a cell is dead or not. The execution will be faster if CELIA knows which attributes are relevant and which are not in checking whether the filament is dead or not. The data card following this command specifies this information. If the i-th column of this data card is "1", the i-th attribute is relevant in determining the dead state. A "0" means irrelevant. If the number of attributes is more than 80, a second data card is necessary and the k-th column of the second data card is treated as the (80+k)th column.

Field A: Not used.

Field B: Not used.

Default: If D-STATE is specified, all attributes are relevant.

12. Command: TABLE

In case a delta function table is used instead of the delta function subroutine, the user should use this control card to initiate the read-in of the table.

Field A: Not used.

Field B: Not used.

If $\delta(l,m,r) = e_1 e_2 \ldots e_i$, then the data cards following this control card are prepared in the following way. First of all, divide all the entries in the delta function table into groups according to the relevancies of cells l, m, r, and the number i, then specify the table group by group. Each group consists of the following cards:

Card 1 (Indication card of the group)

C1 IL Relevance indicator for the left cell "l".
"0" means left cell is irrelevant in determining $e_1 e_2 \ldots e_i$
"1" means relevant, i.e., this cell should be matched in searching the corresponding entry.

C2 IM Relevance indicator for the middle cell "m".
The meaning is the same as IL.

C3 IR Relevance indicator for the right cell "r".
Same meaning as IL and IM.

right justified

CC 9-10 NEW Number of cells to be replaced for the cell "m", i.e., the number i.

CC 18-20 ENTRY Number of entries in this group.

Card 2 to Card (ENTRY+1)

Specifying $l,m,r,e_1,e_2, \ldots ,e_i$ corresponding to the 1st,2nd, ..., and (ENTRY)th entries in this group. Use I-FORMAT. Irrelevant cells in l,m,r do not have to be specified, i.e., just specify (IL+IM+IR+i) cells.

Example: For the group of the following two entries

$(X,m_1,r_1) = e_{11}e_{12}e_{13}$

irrelevant

$(X,m_2,r_2) = e_{21}e_{22}e_{23}$

the set of cards corresponding to this group is

C1 C10 C20

011 3 2

m_1 r_1 e_{11} e_{12} e_{13}

m_2 r_2 e_{21} e_{22} e_{23}

In order to indicate the end of the table specification and to have default action, the indication card of the last group must be IL = IM = IR = 0 and ENTRY = 1. The card following this then specifies the cells to be replaced if CELIA fails to find a matched entry after searching the whole table. With current CELIA, the maxium number of entries the user can specify is 200.

13. Command: FILAMENT

This control statement is used to specify the original filament to be expanded.

Field A: Number of cells in the original filament.

Field B: Not used.

Data cards:

Use I-FORMAT to specify all the cells of the filament consecutively. If the number of cells in the original filament is more than the number specified in Field A of I-FORMAT statement, use as many cards as needed to specify the original filament.

USER-WRITTEN SUBPROGRAMS

The subprograms described in this section are supplied by the user to control the execution of CELIA. All of them are optional. If the user does not supply them with CELIA, he will get these "unsatisfied externals" in his output core map. It is not harmful except the simulation needs some of these subprograms and the user forgot to supply them.

1. SUBROUTINE DELTA(L,M,R,N)

This subroutine specifies the delta function of the Lindenmayer model. CELIA requires this subroutine unless a delta function table is specified by the user using TABLE control card.

This subroutine should start with the following statements:

```
                   Position
                              # of new cells.
SUBROUTINE DELTA(L,M,R,N) ——> output, i.e. to be set up by body,
COMMON /DEL/ CNEW(20,100)                e.g. N = 2 means two
COMMON LINK(1200),NODE(1200),C(2600),NATT     new cells generated
COMMON /FLAG/ F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,OFLAG
    (if these flags are used in this subroutine)
INTEGER R,F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,OFLAG
REAL name1,name2,...,namei
    (if these names are not of type REAL by default)
——>DATA Mname1/1/,Mname2/2/,...,Mnamei/i/
```

Also add the following function statements immediately before the first executable statement in this subroutine.

```
name1(J) = C(J+Mname1-1)
name2(J) = C(J+Mname2-1)
      .
      .
namei(J) = C(J+Mnamei-1)
```

Where L, M and R are pointers pointing to the left neighbor, middle, and right neighbor cells respectively. name1, name2, ..., namei are the names of the first, second, ..., and the i-th attributes given by the user. i is the number of attributes specifying the state of a cell. The reason for adding these function statements is to make this subroutine more user-oriented. The user does not have to know how CELIA structures the information. He need only specify namei(J) and CELIA will give him the value of the i-th attribute of cell J. For example, if CONC is one of the user-given names standing for the concentration of some material in a cell, to obtain the concentration of the right neighbor cell, just write CONC(R).

N is the number of new cells (at most 20) to be replaced. The user should assign new cells into a two-dimensional array called CNEW. The first subscript tells CELIA which cell we refer to, the second subscript stands for the attribute of the cell concerned. Use "M" concatenated with user-defined names to denote the attributes. For example, CNEW(2,MCONC) denotes the concentration of the second cell in the new-born sequence of cells. Thus, for the instruction "If the concentration of the middle cell is less than 3.5, go to S1, otherwise return 2 cells with the concentration of the second cell equal to 10 and ...", we code it the following way

```
IF(CONC(M).LT.3.5) GO TO S1
M=2
CNEW(2,MCONC)=10.
      .
      .
```

second cell in substring generated by this δ-rule.

Note that if the attribute of a cell is represented by a number, it is of floating point mode.

The ten flags F0,F1,...,F9 are used to switch the control of CELIA execution , to the user-written subroutines S0,S1,...S9(described later) temporarily to do some statistical survey or to change the content of the current filament. Thus, if Fi is set (Fi=1) in this subroutine, control will be passed to subroutine Si after the currrnt iteration expanding the current filament is finished. The flag Fi will be reset to zero again when the execution of subroutine Si is finished.

2. SUBROUTINE ENVIRN(T)

This subroutine is required if the user puts "SUBROUTINE" in Field B of ENVIR control card. It determines the left and right environments according to how "old" the filament is, i.e., the current number of iteration T. The subroutine should start with the following statements.

```
      SUBROUTINE ENVIRN(T)
      COMMON LINK(1200),NODE(1200),C(2600),NATT
      COMMON /ENV/ LENVR,RENVR
      INTEGER T,RENVR
```

Where LENVR and RENVR are the locations of the left and right environment specifications respectively. The attributes of left environment should be put from C(LENVR) to C(LENVR+NATT-1) , where NATT is the number of attributes. Specify right environment the same way as left environment. The order of attributes should be the same as that of the cell attributes.

3. SUBROUTINE DEADST(T)

This subroutine is required if the user puts "SUBROUTINE" in Field B of D-STATE control card. It specifies the dead state according to the current number of iteration T. This subroutine should start with

```
      SUBROUTINE DEADST(T)
      COMMON /DD/ DMASK(100),DEAD(100)
      INTEGER DMASK,T
```

Put the attributes of dead state into DEAD(1) to DEAD(NATT), where NATT is the number of attributes. As that of subroutine ENVIRN, the order of these NATT attributes should be the same as that used to specify the cells in the filament. They should correspond to each other. DMASK(1) to DMASK(NATT) are the variables holding the "0"s and "1"s specified in D-MASK data card, their values can also be dynamically changed by this subroutine.

4. FUNCTION KPRINT(T) and FUNCTION KPUNCH(T)

These two functions are required when Fields A of O-REQUEST and P-REQUEST control cards are 2 and 4 respectively. If the returned values are integer 1, CELIA will print or punch the current filament. Start these function codes with

```
      FUNCTION KPRINT(T)
      INTEGER T
        and
      FUNCTION KPUNCH(T)
      INTEGER T
```

5. SUBROUTINES S0,S1,...,S9

Once the command "EXECUTE" is interpreted by CELIA and the expansion of the filament starts, the user loses control of execution until the expansion is finished, i.e., the user cannot use CELIA control statements to control the state of simulation during expansion. However, in case the user wants to change the content of the current filament or to do some survey on some attributes during expansion under some circumstances, the use can use up to ten subroutines to do things like:

(1) Change the filament, including its length and states of cells.

(2) Change the situation of expansion previously specified by CELIA control card, including the current iteration number, number of iterations to expand the filament an the conditions on which CELIA will print or punch the filament.

(3) Print out some information about the current filament.

(4) Store some information concerning the filament.

These ten user-written subroutines are S0,S1,...,S9. Th execution of these ten subroutines is controlled by ten flags F0 F1, ..., F9. If flag Fi is set, i.e., Fi=1, subroutine Si wil be executed after the current iteration expanding the curren filament is finished. If more than one flag are set, th subroutine with the least number will be executed first. Fo example, if F2 and F8 are set, S2 will be executed prior to S8 After the execution of Si, CELIA will reset Fi to zero. Althoug the setting of these flags is usually done in subroutine DELTA the user can set these flags anywhere in any user-writte subroutine as long as he is aware of the execution sequence o those subroutines. The values of these 10 flags are initialize to zero at the very beginning of the execution of CELIA. If n state-change or survey are to be made during execution, the flac need not be changed.

There are no parameters in these subroutines, but the user can have access to some information from the following common blocks:

```
COMMON /FLAG/ F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,OFLAG
COMMON /CNG/ IPRINT,IPUNCH,NMAX,K
```

where F0,F1,...,F9 are the ten flags just mentioned above. IPRINT and IPUNCH are the variables which hold the values in Fields A of O-REQUEST and P-REQUEST control cards respectively. NMAX is the value of Field A in EXECUTE control card, i.e., number of iterations for CELIA to expand the original filament. K is the current number of iteration. The user can change any one of these values whenever he wants.

In order to store, retrieve or change the state of the cells in the current filament, the following 7 subroutines are provided and can be used by the user in any user-written subroutine. The user can store as many as 10 filaments. Each stored filament is identified by a user-defined integer number. Note that the number 0 is always the identification number of the current filament under expansion.

(i) SUBROUTINE STORE(ID)

This subroutine stores the current filament in memory. ID will be the identification number of the stored filament. The current filament is not affected by this subroutine call.

(ii) SUBROUTINE RETURN(ID)

This subroutine returns the stored filament with identification number ID to available space. After this subroutine call, filament ID no longer exists.

(iii) FUNCTION LENGTH(ID)

This function returns an integer value which is the length of filament ID.

(iv) SUBROUTINE COMEIN(ID,IATT,A)

This subroutine puts the values of the IATTth attribute~~s~~ of all cells of filament ID in array A. The IATTth attribute of the Nth cell is put in the Nth element of array A.

(v) SUBROUTINE GOBACK(ID,IATT,A)

This subroutine changes the IATTth attribute of every cell in filament ID to the values specified in array A. The value of the Nth element of array A will be the IATTth attribute of the Nth cell of filament ID.

(vi) SUBROUTINE NEWFIL(ID,LENGTH)

This subroutine creates an empty filament (its cells are not specified) of length LENGTH and stores it with identification number ID. If ID=0, the current filament will be destroyed and this newly created filament will be treated as the current filament being expanded. Note that this created filament is an empty one, the user should specify the states of its cells after this subroutine call unless he is interested in an empty filament.

(vii) SUBROUTINE REPLES(ID)

This subroutine call will cause the replacement of current filament by the stored filament ID. Filament ID will be expanded from that point on. After the replacement, the original current filament no longer exists and there is no filament with ID as identification number either.

There is another subroutine which the user can use to tabulate some statistics and to print the histogram corresponding to that table. It is

```
SUBROUTINE STATIS(IDNO,ITYPE,A,N,IHISTO,BL,BU,INT)
```

The parameter IDNO is the user-given identification integer number of the table. A is the array in which N data items are stored which are to be tabulated. ITYPE is the indicator which indicates the type of data stored in array A. ITYPE=0 means the data stored in array A are real numbers and their values are continuous, i.e., their values may be any real numbers. ITYPE=1 means the data in array A are real numbers and they are discrete, i.e., each number belong to a finite set of real numbers. ITYPE=2 means the data stored in array A are character strings. If ITYPE equals 1 or 2, CELIA will tabulate the data in such a way that the data with the same value will be grouped together. If ITYPE equals 0, CELIA will divide all the data with values between BL(lower bound) and BU(upper bound) into INT(interval) groups. The table will show some statistical information about the whole array of data items and each group, such as number of items, mean, deviation, etc. If IHISTO is 1, a histogram corresponding to that table will be printed out. The parameters BU, BL and INT are not used for the cases ITYPE = 1 or 2.

It is helpful to know the sequence of execution inside each iteration if the user wants to use the above subprograms and flags effectively. The execution sequence is

```
Expand the current filament (apply subroutine DELTA or
     search delta function table).
If F0=1, execute subroutine S0.
If F1=1, execute subroutine S1.
          .
          .
If F9=1, execute subroutine S9.
Check whether it is necessary to print or punch the
     filament.
Check whether the filament is dead or not.
```

EXAMPLES OF CONTROL AND DATA CARDS

```
*    FIBONACCI NUMBERS
*      (USING DELTA FUNCTION TABLE)

TABLE
010            1              1
AB
010            2              1
BAB
000            1              1
X
O-REQUEST                     4
FILAMENT                      1
A
EXECUTE                       15
END
```

```
*    FRENCH FLAG PROBLEM
*      (USING SUBROUTINE DELTA)

NATT                     3
I-FORMAT                 20             (A1,2F1.0,1X)
O-FORMAT                 25             (A1,2F1.0,1X)
O-REQUEST                4
ENVIR                                   FIXED
G00 G00
FILAMENT                 15
I00 S00 S00 S00 S00 S00 S00 S00 S00 S00 S00 S00 S00 S00 S00
EXECUTE                  30
FILAMENT                 4
I00 S00 S00 S00
O-REQUEST                4              ENV-SINGLE
EXECUTE                  10
END
```

CARD DECK PREPARATION

(215111 - 707, CELIA) F = $\overset{50000}{55000}$ (user routines

(1) With user-written subprograms

```
    Job card (F=55000.)
    LOADER,PPLOADR.
    ATTACH,CELIA,INTERIM,CY=1.
    RUN(S)
    LOAD(LGO)
    LOAD(CELIA)
    EXECUTE(CELIA)
    7-8-9
    CELIA control and data cards
    7-8-9
    6-7-8-9
```

USER DEFINED SUBPROGS
7 - 8 - 9

(2) Without any user-written subprogram

```
    Job card (F=50000.)
    LOADER,PPLOADR.
    ATTACH,CELIA,INTERIM,CY=1.
    CELIA.
    7-8-9
    CELIA control and data cards
    7-8-9
    6-7-8-9
```