RA

stichting
mathematisch
centrum

MC

REKENAFDELING MR 138/72 SEPTEMBER

P.M.B. VITÁNYI

DOL-LANGUAGES AND A FEASIBLE SOLUTION
FOR A WORD PROBLEM

RA

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 <u>word</u> or <u>string</u> 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 $a\in\sum$ 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

$$L_1.L_2 = \{\alpha\beta \mid \alpha\in L_1 \text{ and } \beta\in L_2\}.$$

$\sum^*$ 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 $\omega\in\sum^*$ then $\gamma : \sum^* \to 2^{\sum}$ is defined by

$$\gamma(\omega) = \{a\in\sum \mid \exists \eta,\xi\in\sum^* \ [\omega = \eta a \xi]\}$$

$|\sum|$ 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 γ(Seq) = $\{c_i\in\sum \mid c_i$ occurs in Seq$\}$, and |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 $S = \langle \sum, \delta \rangle$ where

(i) $\sum$ is a nonempty finite set, the alphabet of S, and an element of $\sum$ is called a letter.

(ii) $\delta : \sum \to \sum^*$ 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 $\omega = a_1 a_2 \ldots a_m \in \sum^+$ and

$\omega' = \alpha_1 \alpha_2 \ldots \alpha_m \in \sum^*$, 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 $\omega \overset{*(k)}{\Rightarrow} \omega'$ 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 $\omega \overset{*(k)}{\Rightarrow} \omega'$, then ω' is derived in k steps from ω, and $\omega \overset{*(k)}{\Rightarrow} \omega'$ is a k-derivation of ω' from ω. $\omega \overset{+}{\Rightarrow} \omega'$ means $\omega \overset{*(k)}{\Rightarrow} \omega'$ 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 " ' " .

$\delta^0(\omega) = \omega$

$\delta^i(\omega) = \delta(\delta^{i-1}(\omega))$; i.e. $\omega \overset{*(i)}{\Longrightarrow} \delta^i(\omega)$

$\omega' \in \delta^*(\omega)$ iff $\omega \overset{*}{\Longrightarrow} \omega'$.

$\gamma(\delta^*(\omega)) \overset{\text{def}}{=\!=\!=} \bigcup_{\omega' \in \delta^*(\omega)} \gamma(\omega')$

$\delta^i(\sum) \overset{\text{def}}{=\!=\!=} \bigcup_{a \in \sum} \delta^i(a)$ and $\gamma(\delta^i(\sum)) \overset{\text{def}}{=\!=\!=} \bigcup_{a \in \sum} \gamma(\delta^i(a))$

$\delta^*(\sum) \overset{\text{def}}{=\!=\!=} \bigcup_{a \in \sum} \delta^*(a)$ and $\gamma(\delta^*(\sum)) \overset{\text{def}}{=\!=\!=} \bigcup_{a \in \sum} \gamma(\delta^*(a))$

Def. 1.2 A DOL-System (DOL) is an ordered triple

$G = <\sum, \delta, \sigma>$ where

(i) $\sum$ and δ are as in def. 1.1

(ii) $\sigma \in \sum^+$ is called the axiom of G.

Def. 1.3 The DOL-Language generated by a DOL $G = <\sum, \delta, \sigma>$ is defined by

$L(G) = \{\omega \in \sum^* \mid \sigma \overset{*}{\Longrightarrow} \omega\}$, 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 $G = <\sum, \delta, \sigma>$ i.e.

$\xi(G) = \sigma, \delta(\sigma), \delta^2(\sigma), \ldots$

is called a propagation.

Example 1 $G = \langle\{a\}, \{a \to aa\}, a\rangle$

$\xi(G) = a, aa, aaaa, \ldots.$

$L(G) = \{a^{2^t} \mid t \geq 0\}$

Example 2 $G = \langle\{a,b\},\{a \to aba, b \to \lambda\}, a\rangle$

$\xi(G) = a, aba, abaaba, \ldots.$

$L(G) = \{(aba)^{2^t} \mid t \geq 0\} \cup \{a\}$

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

$1, 1, 2, 3, 5, \ldots.$

Consider the DOL $G = \langle\{a,b,c\}, \{a \to aa, b \to bb\}, a\rangle$

$\xi(G) = a, aa, aaaa, \ldots.$

$L(G) = \{a^{2^t} \mid t \geq 0\}$

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 $G = \langle \Sigma, \delta, \sigma \rangle$ generates depends on the generic or productive qualities imbued to the letters by the semi DOL $S = \langle \Sigma, \delta \rangle$.

Def. 2.1 Let $S \langle \Sigma, \delta \rangle$ be a semi DOL, $|\Sigma| = p$.
A letter $a \in \Sigma$ is

(i) Mortal iff $a \overset{*}{\Longrightarrow} \lambda$. $\Sigma_m = \{a \mid a \text{ is mortal}\}$. $p_m = |\Sigma_m|$.

(ii) Vital iff $a \notin \Sigma_m$. $\Sigma_v = \Sigma \setminus \Sigma_m$. $p_v = |\Sigma_v|$.

(a) Recursive iff $a \overset{+}{\Longrightarrow} \eta a \xi$ for some $\eta, \xi \in \Sigma^*$ *)

$\Sigma_r = \{a \mid a \text{ is recursive}\}$. $p_r = \Sigma_r$.

If $a \overset{+}{\Longrightarrow} \eta a \xi$ with $\eta, \xi \in \Sigma_m^*$ then letter a is monorecursive.

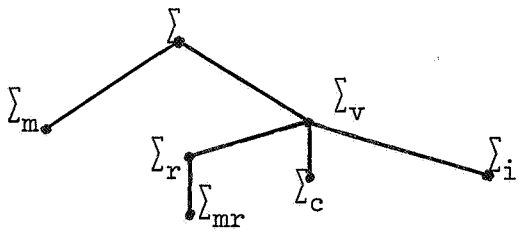
$\Sigma_{mr} = \{a \mid a \text{ is monorecursive}\}$. $p_{mr} = |\Sigma_{mr}|$.

(b) Recurring iff $a \notin \Sigma_r$ and there exists a letter $b \in \Sigma_r$ such that $b \overset{*}{\Longrightarrow} \eta a \xi$. $\Sigma_c = \{a \in \Sigma \setminus \Sigma_m \mid a \text{ is recurring}\}$. $p_c = |\Sigma_c|$.

(c) Initial iff $a \notin \Sigma_r$ and there does not exist a letter $b \in \Sigma_r$ such that $b \overset{*}{\Longrightarrow} \eta a \xi$. $\Sigma_i = \{a \in \Sigma \setminus \Sigma_m \mid a \text{ is initial}\}$. $p_i = |\Sigma_i|$.

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) $\sum_v \cup \sum_m = \sum$.

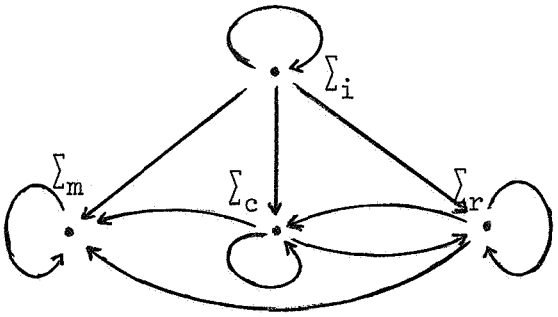
(ii) $\sum_i \cup \sum_c \cup \sum_r = \sum_v$; $\sum_{mr} \subseteq \sum_r$.

(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:

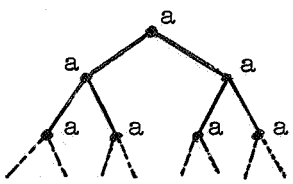


A useful heuristic device in the investigation of aspects of the generic power of a letter $a \in \sum$, $S = \langle \sum, \delta \rangle$, is the notion of the propagation tree T_a of a; related to the rule tree or derivation tree as encountered in the theory of context free languages.

Def. 2.2. Let $S = \langle \sum, \delta \rangle$ be a semi DOL.
The propagation tree T_a of $a \in \sum$ is a labeled directed tree of which the labels attached to the nodes are elements of $\sum$. 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 $b_i \in \sum$ is attached to (k,j), then node (k,j) is connected by edges with nodes $(k+1,h),\ldots,(k+1,h+n)$ labelled $c_{i_1},\ldots,c_{i_n}$, respectively, iff $b_i \to c_{i_1}\ldots c_{i_n} \in \delta$. 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.

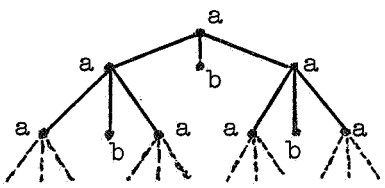
Example 1 $S = \langle \{a\}, \{a \to aa\} \rangle$

$T_a =$



Example 2 $S = \langle \{a,b\}, \{a \to aba, b \to \lambda\} \rangle$

$T_a =$



$T_b =$

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 $a \overset{*(k)}{\Longrightarrow} \lambda$, $k \leq p_m$, 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

$\exists i,j\in\{0,1,\ldots,p_m\}\ [i < j \text{ and } b_i = b_j]$.

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 $S = \langle \Sigma, \delta \rangle$ be a semi DOL and $a \in \Sigma$.

$$a \in \Sigma_v \text{ iff } \lambda \notin \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$$

Proof $a \notin \Sigma_m$ iff $\lambda \notin \bigcup_{0<k\leq p_m} \{\delta^k(a)\}$. Hence $a \in \Sigma \setminus \Sigma_m = \Sigma_v$ □

Lemma 2.3 Let $S = \langle \Sigma, \delta \rangle$ be a semi DOL and $a \in \Sigma$.

$$a \in \Sigma_r \text{ iff } a \in \bigcup_{0<k\leq p_r} \gamma(\delta^k(a))$$

Proof ← . If $a \in \bigcup_{0<k\leq p_r} \gamma(\delta^k(a))$ then $a \in \Sigma_r$.

→ . If $a \in \Sigma_r$ 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 $l(a) = c_0, c_1, c_2, \ldots, c_{h-1}, c_h$ 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) $\gamma(l(a)) \subseteq \Sigma_r$.

Suppose $h > p_r$.

Because of (1), (2) and (3) $\quad |\{c_1, c_2, \ldots, c_{h-1}\}| < p_r$.

Therefore $\quad \exists i,j \in \{1,2,\ldots,h-1\} [i<j \text{ and } c_i = c_j]$,

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.

$$K_a \overset{\text{def}}{=\!=\!=} \{k \in \mathbb{N} \mid k \text{ is a period of } a\}.$$

Lemma 2.4 Let $S = \langle \sum, \delta \rangle$ be a semi DOL and $a \in \sum$.

$$a \in \sum_c \quad \text{iff} \quad a \in \bigcup_{0<k\leq p_c} \gamma(\delta^k(\sum_r)) \cap \sum_v \backslash \sum_r$$

Proof Along lines similar to the proofs of lemma's 2.1-2.3 □

Corollary $a \in \sum_i$ iff $a \notin \bigcup_{0<k\leq p_c} \gamma(\delta^k(\sum_r)) \cap \sum_v \backslash \sum_r$ and $a \in \sum_v \backslash \sum_r$.

Theorem 2.5 Let $S = \langle \sum, \delta \rangle$ be a semi DOL. For every $a \in \sum$ we can effectively determine

(i) Whether a is mortal, vital, or recursive, by examining

$$\bigcup_{0<k\leq p} \{\delta^k(a)\} \quad \text{and} \quad \bigcup_{0<k\leq p} \gamma(\delta^k(a))$$

(ii) Whether a is recurring vital or initial vital by examining

$$\bigcup_{0<k\leq p_v - p_r} \gamma(\delta^k(\sum_r))$$

Proof By lemma's 2.1-2.4 and the corollary. □

Theorem 2.6 We can effectively determine whether a DOL $G = \langle \sum, \delta, \sigma \rangle$ is

(i) Quasi-reduced

(ii) Reduced.

Proof Hint (i) $\bigcup_{0 \leq k \leq p} \gamma(\delta^k(\sigma)) = \sum$ iff G is quasi-reduced.

(ii) $\bigcup_{0 \leq k \leq p} \gamma(\delta^k(\sigma)) = \bigcup_{p-p_r \leq k \leq 2p-p_r} \gamma(\delta^k(\sigma)) = \sum$ iff G is reduced.

2.1 RECURSIVE COMPLEXITY

Consider a language like

$$L(G) = \{a^{2^t} b^{2^t} c^{2^t} \mid t \geq 0\}$$

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

$$G = \langle\{a,b,c\},\{a \to aa,\ b \to bb,\ c \to cc\},\ abc\rangle.$$

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 $S = \langle\sum,\delta\rangle$ 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 $[a] \supseteq \cup \{\gamma(C(a)) \mid C(a)$ 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

$G = \langle \{a,b,c\}, \{a \to ab,\ b \to ac,\ c \to b\},\ a \rangle$

$\cup\{\gamma(C(a))\} = \{a\} \cup \{a,b\} = \{a,b\}$.

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 $S = \langle \sum, \delta \rangle$ 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 $G = \langle \sum, \delta, \sigma \rangle$ and a word ω_τ is it decidable whether $\omega_\tau \epsilon L(G)$. 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 $p = |\sum|$

$K = \max_{a \epsilon \sum} \{|\alpha| \; | a \rightarrow \alpha\}$

$M = \max_{\omega \epsilon L(G)} \{|\omega|_V \mid |\omega|_V$ 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 $\omega_1 \not\overset{*}{\Longrightarrow} \omega_2$.

<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 $\gamma(1(b)) \cap \sum_r \neq \emptyset$]

where $1(b) = b_0,b_1,\ldots,b_{t-1},b_t$, 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 $\gamma(1(b)) \subseteq \sum_v$.

Suppose the lemma is not true, i.e. $\gamma(1(b)) \subseteq \sum_i \cup \sum_c$.

But then $|1(b)| = t+1 > p_i + p_c$ while $|\gamma(1(b))| \leq p_i + p_c$.

Hence $1(b)$ contains a repetition of a letter and $\gamma(1(b)) \cap \sum_r \neq \emptyset$, which contradicts the assumption. □

Corollary 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.

Def. 3.1 A DOL $G' = <\sum', \delta', \sigma'>$ is the positive k-displacement 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 negative k-displacement of G .

lemma 3.3 The positive $p\text{-}p_r$ displacement $G' = <\sum', \delta', \sigma'>$ of $G = <\sum, \delta, \sigma>$ is reduced.

Proof By the corollary of lemma 3.2 and by lemma 2.1 □

lemma 3.4 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, $\delta^*(a) \subseteq \sum_m^* [a] \sum_m^*$.

Proof 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 $S = \langle \sum, \delta \rangle$ be a semi DOL.

(i) If $a \in \sum_r$ then $a \in \bigcup_{0<k\leq p_r} \gamma(\delta^k(a)) \cap \sum_r$

(ii) If $a \in \sum_c$ then $\bigcup_{0<k\leq p_c} \gamma(\delta^k(a)) \cap \sum_r \neq \emptyset$

(iii) If $a \in \sum_i$ 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 $a \in \sum_c$. Then there is a $b \in \sum_v$ such that

$$b \in \gamma(\delta^{p_c}(a)) \cap \sum_v.$$

Let $l(b) = b_0, b_1, \ldots, b_{p_c}$, $b_0 = a$, $b_{p_c} = b$, be a pedigree of b.

Since $a \in \sum_c$, $\gamma(l(b)) \subseteq \sum_c \cup \sum_r$.

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 $S = \langle \sum, \delta \rangle$ be a semi DOL.

(i) If $a \in \sum_r \setminus \sum_{mr}$, i.e. $|\delta^k(a)|_v = x > 1$ where $k = \min K_a$,

then $\forall n \in \mathbb{N}\ [|\delta^{nk}(a)|_v > nx-n]$

(ii) If $a \in \sum_{mr}$, i.e. $|\delta^k(a)|_v = 1$ where $K_a = \{k\}$,

then $\forall t \in \mathbb{N}\ [|\delta^t(a)|_v = |\delta^t(a)|_r = 1]$

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 $\gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \setminus \sum_{mr} \neq \emptyset$.

Let $a_i \in \gamma(\delta^{p_i+p_c}(\sigma)) \cap \sum_v \setminus \sum_{mr}$.

Case 1 $a_i \in \sum_r \setminus \sum_{mr}$, 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 $b \in \sum_r \setminus \sum_{mr}$; 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:

$$\delta^{p_i+p_c+p_m}(\sigma) = \delta^{p_i+p_c+p_m+u}(\sigma) \text{ and hence}$$
$$|L(G)| \leq p-p_r + \text{l.c.m.}(k_1,k_2,\ldots,k_m) \qquad \square$$

Corollary 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 $\bigcup_{k=0}^{\infty}\gamma(\delta^k(\sigma)) \cap (\sum_c \cup \sum_r\backslash\sum_{mr}) \neq \emptyset$.

case 1 For some $t \geq 0$ $b\in\gamma(\delta^t(\sigma)) \cap \sum_r\backslash\sum_{mr}$. By the previous arguments L(G) is infinite.

case 2 For some $t \geq 0$ $b\in\gamma(\delta^t(\sigma)) \cap \sum_c$. 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 $a_i \in \sum_i$: contrary to lemma 3.2.

Since cases 1 - 3 cannot occur, $a_i \in \sum_{mr}$ 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.

$$\exists a_i \in \gamma(\delta^{p_i+p_c}(\sigma)) \; \forall n \in \mathbb{N} \; [|\delta^{n*k_i}(a_i)|_v > n]$$

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.

$$\exists x,u \in \mathbb{N} \; [\delta^x(\sigma) = \delta^{x+u}(\sigma)], \text{ i.e.}$$

$$L(G) = \{\delta^t(\sigma) \mid 0 \leq t < x+u\}.$$

If $\omega_\tau \notin L(G)$ then $\sigma \overset{*}{\not\Longrightarrow} \omega_\tau$.

Lemma 3.10 Let $S = \langle \sum, \delta \rangle$ be a semi DOL and $a \in \sum_{mr}$.

$$\delta^{p_m+t}(a) = \delta^{p_m+t'}(a) \text{ for } t' \equiv t \bmod (k)$$

$$\delta^{p_m+t}(a) \neq \delta^{p_m+t'}(a) \text{ for } t' \not\equiv t \bmod (k)$$

where $t,t' \in \mathbb{N}$ 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.

$\gamma(\delta^{t'}(\sigma)) \cap \sum_r \setminus \sum_{mr} \neq \emptyset$ 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 $a_i \in \sum_i$: contrary to lemma 3.2.

Since cases 1 - 3 cannot occur, $a_i \in \sum_{mr}$ 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.

$$\exists a_i \in \gamma(\delta^{p_i+p_c}(\sigma)) \; \forall n \in \mathbb{N} \; [|\delta^{n*k_i}(a_i)|_v > n]$$

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.

$$\exists x,u \in \mathbb{N} \; [\delta^x(\sigma) = \delta^{x+u}(\sigma)], \text{ i.e.}$$

$$L(G) = \{\delta^t(\sigma) \mid 0 \leq t < x+u\}.$$

If $\omega_\tau \notin L(G)$ then $\sigma \overset{*}{\not\Longrightarrow} \omega_\tau$.

Lemma 3.10 Let $S = \langle \sum, \delta \rangle$ be a semi DOL and $a \in \sum_{mr}$.

$$\delta^{p_m+t}(a) = \delta^{p_m+t'}(a) \text{ for } t' \equiv t \bmod (k)$$

$$\delta^{p_m+t}(a) \neq \delta^{p_m+t'}(a) \text{ for } t' \not\equiv t \bmod (k)$$

where $t,t' \in \mathbb{N}$ 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 $\delta^{p_i+p_c}(\sigma) = \eta_1\, a_1\, \eta_2\, a_2 \ldots \eta_m\, a_m\, \eta_{m+1}$ with $\eta_1,\ldots,\eta_{m+1} \in \sum_m^*$ and $a_1,\ldots,a_m \in \sum_{mr}$.

Since $\delta^{p_m}(\eta) = \lambda$ for all $\eta \in \sum_m^*$ we have

$$(1)\quad \delta^{p_m}(\delta^{p_i+p_c}(\sigma)) = \delta^{p_m}(\eta_1)\,\delta^{p_m}(a_1) \ldots \delta^{p_m}(a_m)\;\delta^{p_m}(\eta_{m+1}) = \delta^{p_m}(a_1)\,\delta^{p_m}(a_2) \ldots \delta^{p_m}(a_m).$$

By lemma 3.10 (let k_i be the period of a_i):

$$(2)\quad \forall a_i \in \sum_{mr}\ \forall t,t' \geq 0\ [\text{if } t \equiv t' \bmod(k_i) \text{ then } \delta^{p_m+t}(a_i) = \delta^{p_m+t'}(a_i)],$$

and by lemma 3.11

$$(3)\quad \forall a_i \in \sum_{mr}\ \forall t,t' \geq 0\ \forall \mu \in \sum^+\ [\delta^{p_m+t}(a_i) \neq \delta^{p_m+t'}(a_i)\mu].$$

(1)-(3) reduces the problem, for finite L(G), to the following: do there exist $t_1,t_2,\ldots,t_m \in \mathbb{N}$ such that

$$\delta^{p_m+t_1}(a_1)\,\delta^{p_m+t_2}(a_2) \ldots \delta^{p_m+t_m}(a_m) = \omega_\tau$$

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

$$\{[a]_1, \ldots, [a]_q\} = \sum_r / \sim \qquad \text{(G is quasi-reduced)}.$$

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 (σ,ω_τ);
           begin if |σ|_v > |ω_τ|_v then
                 begin print (¢no solution¢); goto exit end
                 else if σ = ω_τ then
                 begin print (¢solution found¢); goto exit end;
           end;

phase 0:   ¢ classify all a_i ∈ Σ; by examining ∪_{0<k≤p} γ(δ^k(a_i)) and
           ∪_{0<k≤p} δ^k(a_i), whether they belong to Σ_m, Σ_r, or Σ_c ∪ Σ_i.
           If a_i ∈ Σ_r then determine its smallest period k_i ¢

phase 1:   compare (σ,ω_τ);

phase 2:   for i:= 1 step 1 until p do
           begin σ:= δ(σ); compare (σ,ω_τ)
           end; vital b:= |σ|_v;

phase 3:   for i:= 1 step 1 until p_v do
           begin σ:= δ(σ); compare (σ,ω_τ)
           end; vital end:= |σ|_v;
           if vital end = vital b then goto phase 5;

phase 4:   ¢ L(G) is infinite ¢
           σ:= δ(σ); compare (σ,ω_τ);
           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 ω_τ = δ^j(δ^{p_m}(a_i))η then
        begin t_i := j;  ω_τ := η;  i := i+1;
              if ω_τ ≠ λ ∧ i ≤ m then goto next else
              if ω_τ = λ ∧ 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 ∤ t_j mod (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 $G = \langle \sum, \delta, \sigma \rangle$ 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

$$\begin{aligned}
\forall a \in \textstyle\sum: \quad \delta^2(a) &= b_1 \ldots b_m \\
\delta^4(a) &= \delta^2(b_1) \ldots \delta^2(b_m) \\
&\vdots \\
\delta^{2^t}(a) &= c_1 \ldots c_n \\
\delta^{2^{t+1}}(a) &= \delta^{2^t}(c_1) \ldots \delta^{2^t}(c_n)
\end{aligned}$$

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 $G = \langle \sum, \delta, \sigma \rangle$.

$$\forall j \geq p_i + p_c \left[\max_{i \in \mathbb{N}} \{ |\delta^i(\sigma)|_v \} = |\delta^j(\sigma)|_v = |\delta^j(\sigma)|_r \right].$$

Proof By lemma 3.4 and theorem 3.7. □

Theorem 3.16. Let L(G) be a finite DOL language generated by $G = \langle \sum, \delta, \sigma \rangle$.

$$u \leq L(G) \leq u + p - p_r$$

where $u = \text{l.c.m.}\ (k_1, k_2, \ldots, k_m)$ and $k_1, k_2, \ldots, k_m$ are the periods of the monorecursive $a_1, a_2, \ldots, a_m$ 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.

$$(1) \quad \forall t, t' \geq p_i + p_c \quad \forall i \in \{1, 2, \ldots, m\} \left[\text{if } t \equiv t' \bmod(k_i) \text{ then } a_i(t) = a_i(t') \text{ else } a_i(t) \neq a_i(t') \right].$$

Therefore:

$$(2) \quad \forall t \geq p_i + p_c \quad \forall j < \text{l.c.m.}(k_1, \ldots, k_m) \quad \left[a_1(t)\, a_2(t) \ldots a_m(t) \neq a_1(t+j)\, a_2(t+j) \ldots a_m(t+j) \right]$$

Hence

$$(3) \quad |L(G)| \geq \text{l.c.m.}\ (k_1, k_2, \ldots, k_m) = u.$$

By the proof of theorem 3.7

$$(4) \quad |L(G)| \leq p - p_r + \text{l.c.m.}\ (k_1, k_2, \ldots, k_m).$$

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. $$|L(G)| \leq p - p_r + \text{l.c.m.}\ (k_1,\ldots,k_s)$$

$$\leq p - p_r + p_r^s$$

$$\leq p(1+p^{n-1})$$

$$\leq p(1+p^{m-1})$$

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 $G = <\sum,\delta,\sigma>$ 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 $$f_1, f_2, f_3, f_4 : \mathbb{N} \to \mathbb{N}$$

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 = /".

Example. $S_1 = \langle\{a,b\},\{a\to ab, b\to bb\}\rangle$

$a \overset{*}{\Longrightarrow} abbbbb$? $\quad a \overset{*}{\Longrightarrow} abbbb$?

$S_2 = \langle\{a\},\{a\to aa\}\rangle$

$aa \overset{*}{\Longrightarrow} aaaa$? $\quad a \overset{*}{\Longrightarrow} aaaaa$?

$a \overset{*}{\Longrightarrow} \lambda$?

```
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

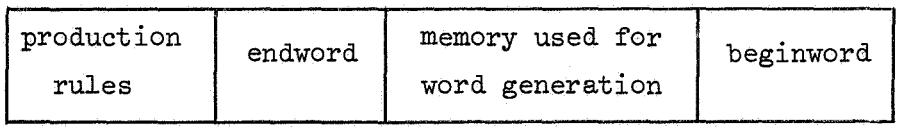
error job control input	←	subsequent to "job =" one of the following symbols is missing: "g", "b", "e", or "/".
error in input	←	(i) input format incorrect, e.g. grammar inputted after begin- or endword. (ii) insufficient input, e.g. no grammar, begin- or endword. (iii) the identification symbol in front 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.

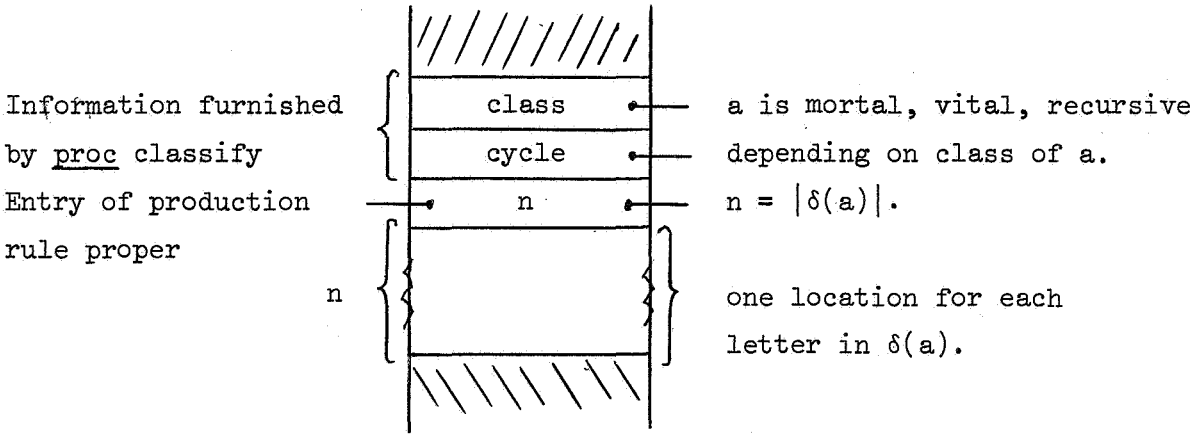
To promote efficient use of memory, the production rules, beginword, endword, the concurrently produced word, the previously produced word, and, if necessary, information for application of the Chinese Remainder Theorem are stored in one array called "array" (see figure).



Low core high core

"array"

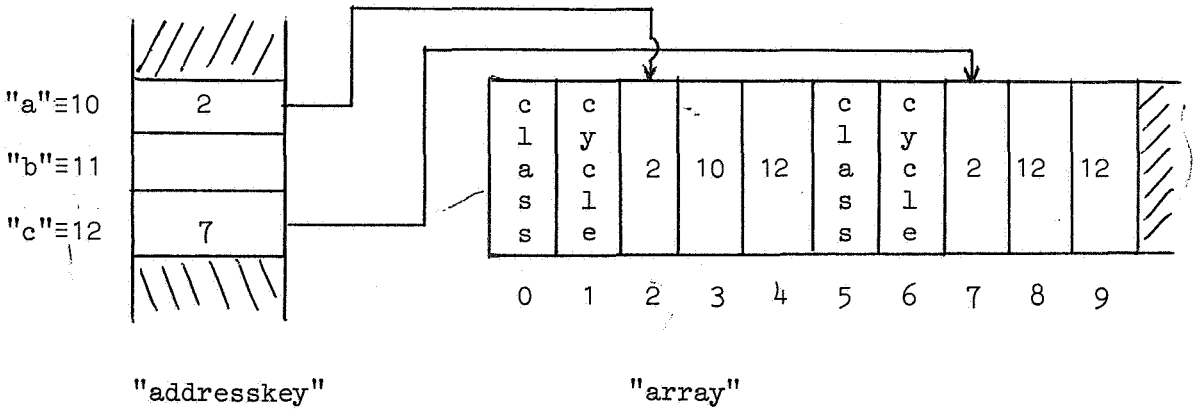
The production rules are stored as follows (the figure depicts the storage of a → δ(a))



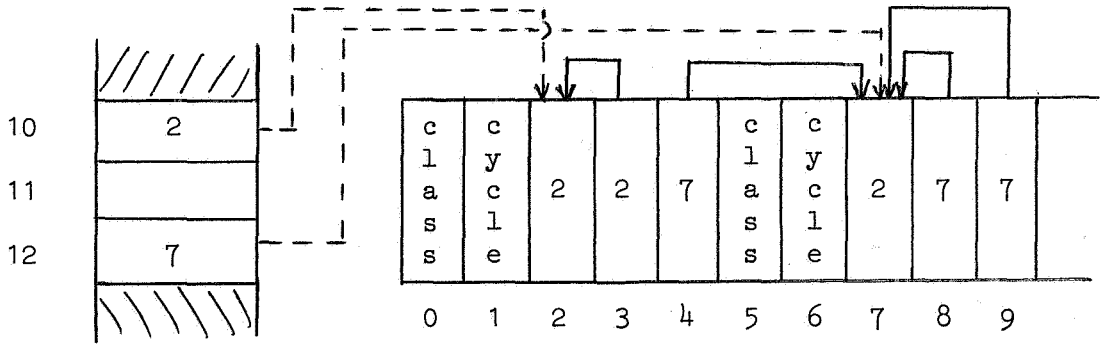
If $a \in \sum_r$ cycle:= min K_a.
If $a \in \sum_m$ cycle:= number of generations needed to derive λ.
If $a \in \sum_v$ cycle:= $|\sum|$.

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.

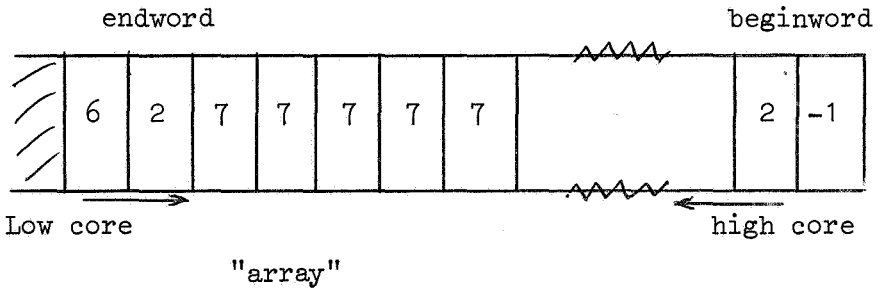
Example. G = <{a,c},{a→ac,c→cc},a>



Subsequent to the reading of the production rules, the pointers in "array" to "addresskey" are replaced by corresponding pointers to "array" itself (assemblage).

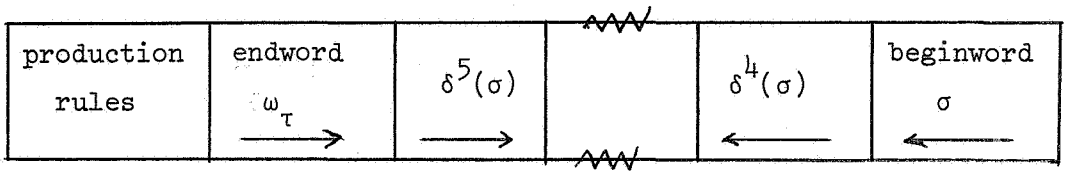


All words are stored in a similar way. The entry point contains the word length. Subsequent elements of the array contain pointers to the corresponding production rule. When the word is stored at high core, the word length is taken negative otherwise positive.
Let the beginword be "a" and the endword "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 $G = \langle \Sigma, \delta, \sigma \rangle$ 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 ($\delta^i(\sigma)$);

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;
    1         'INTEGER' BEGIN OF MEMORY,END OF MEMORY,FIRSTKEY,LASTKEY;
    2                   BEGIN OF MEMORY:=1;
    3                   END OF MEMORY:=30000;
    4                 FIRSTKEY:=0;
    5                   LASTKEY:=127;
    6         'BEGIN' 'INTEGER' 'ARRAY' ADDRESSKEY[FIRSTKEY:LASTKEY]
    7         ,ARRAY[BEGIN OF MEMORY : END OF MEMORY]
    8         ;
    9         'INTEGER' SEPARATOR,EQUALSIGN,TAB,TWNR,SPACE
   10                   ,B,G,E
   11                   ,CYCLE,  CLASS,INFOGRAM
   12                   ,JOB,COUNT,PHASE
   13                   ,P,PM,PVR,VITALP,VITALBW,VITALEND
   14                   ,BW,EW,EWNOS,BOM,EOM
   15                   ,MORTAL,RECURSIVE,VITAL
   16                   ,I,NBW,EWN,EWHELP,BEGINPAIR,LW,LETTER,J,PL,M,BOMMIN,D
   17                   ;
   18         'BOOLEAN' SOLUTION,GRAM,BEGIN,END
   19                   ;
   20
   21         'PROCEDURE' INITIALIZE;
   22         'BEGIN' JOB:=MORTAL:=0;
   23                 SEPARATOR:=67;EQUALSIGN:=70;TAB:=118;TWNR:=119;
   24                 SPACE:=93;
   25                 B:=11;G:=16 ;E:=14;
   26                 CYCLE:=-1;CLASS:=-2;INFOGRAM:=2;
   27                 RECURSIVE:=1;VITAL:=-1;
   28         'END' INITIALIZE;
   29
   30         'PROCEDURE' ERROR(STRING);'STRING' STRING;
   31         'BEGIN' CARRIAGE(2);PRINTTEXT(STRING);NLCR;
   32                 PRINTTEXT('('JOBNUMBER:')');ABSFIXT(2,0,JOB);EXIT
   33          'END' ERROR;
   34
   35         'PROCEDURE' READ INPUT;
   36         'BEGIN' 'INTEGER' CODE,DUMMY;
   37                 'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ EQUALSIGN 'DO';
   38                 'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ SEPARATOR 'DO'
   39                 'IF' CODE=G 'THEN' GRAM:=BEGIN:=END:='FALSE' 'ELSE'
   40                 'IF' CODE=B 'THEN' BEGIN:='FALSE' 'ELSE'
   41                 'IF' CODE=E 'THEN' END:='FALSE' 'ELSE'
   42                 ERROR('('ERROR JOB CONTROL  INPUT')');
   43                 'IF' GRAM ∧ BEGIN ∧ END 'THEN' ERROR('('PROGRAM END')');
   44                 'FOR' DUMMY:=0 'WHILE' ¬GRAM ∨ ¬BEGIN ∨ ¬END 'DO'
   45                 'BEGIN' CODE:=NEXTSYMBOL;
   46                 'IF' ¬GRAM ∧ CODE=G 'THEN'
   47                 'BEGIN' READ GRAMMAR;GRAM:='TRUE' 'END' 'ELSE'
   48                 'IF' ¬BEGIN ∧ CODE=B ∧ GRAM 'THEN'
   49                 'BEGIN' GENERATE BEGINWORD;BEGIN:='TRUE' 'END' 'ELSE'
   50                 'IF' ¬END ∧ CODE=E ∧ GRAM 'THEN'
   51                 'BEGIN' GENERATE ENDWORD;END:='TRUE' 'END' 'ELSE'
   52                 ERROR('('ERROR IN INPUT')');
   53                 'END'
   54         'END' READ INPUT;
   55
```

```
'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;
```

```
116
117         'INTEGER' 'PROCEDURE' NEXTSYMBOL;
118         'BEGIN' 'INTEGER' SYM;
119         NEXT:   SYM:=RESYM;PRSYM(SYM);
120                 'IF' SYM=SPACE v SYM=TAB v SYM=TWNR 'THEN' 'GOTO' NEXT
121         ;NEXTSYMBOL:=SYM
122         'END' NEXTSYMBOL;
123
124         'INTEGER' 'PROCEDURE' CLASS SYMBOLS(ADDRESS,KIND);
125         'VALUE' ADDRESS,KIND;'INTEGER' ADDRESS,KIND;
126         'BEGIN' 'INTEGER' I,N,HELP;
127                 N:=ARRAY[ADDRESS];
128                 HELP:=0;
129                 'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
130                 'IF' ARRAY[ARRAY[ADDRESS+I]+CLASS]=KIND 'THEN'
131                 HELP:=HELP+1;
132                 CLASS SYMBOLS:=HELP
133         'END' CLASS SYMBOLS;
134
135         'INTEGER' 'PROCEDURE' GCD(A,B);'VALUE' A,B;'INTEGER' A,B;
136         'BEGIN' 'INTEGER' W;
137         AGAIN:  'IF' B≠ 0 'THEN'
138                 'BEGIN' W:=A-A÷B*B;
139                         A:=B; B:=W; 'GOTO' AGAIN
140                 'END';
141                 GCD:= ABS(A)
142         'END' GCD;
143
144         'BOOLEAN' 'PROCEDURE' COMPPREFIX(WORD,EW,EWN);'VALUE' WORD,EW,EWN;'INTEGER' WORD,EW,EWN;
145         'BEGIN' 'INTEGER' I,M;
146                 COMPPREFIX:='FALSE';
147                 M:=ARRAY[WORD];
148                 'IF' EWN < ABS(M) 'THEN' 'GOTO' READY;
149                 'FOR' I:=1 'STEP' 1 'UNTIL' M,-1 'STEP' -1 'UNTIL'M 'DO'
150                 'IF' ARRAY[WORD+I]≠ ARRAY[EW+ ABS(I)] 'THEN' 'GOTO' READY;
151                 COMPPREFIX:='TRUE';
152         READY:
153         'END' COMPPREFIX;
154
155         'PROCEDURE' CLASSIFY(P,PM);'VALUE' P;'INTEGER' P,PM;
156         'BEGIN' 'INTEGER' I,J,NI,ADDRESS,SYMBOL,CIRCLE,KIND,N;
157                 PM:=0;
158                 'FOR' J:=FIRSTKEY 'STEP' 1 'UNTIL' LASTKEY 'DO'
159                 'BEGIN' ADDRESS:=SYMBOL:=ADDRESSKEY[J];CIRCLE:=0;
160                         'IF' SYMBOL ≠0 'THEN'
161                         'BEGIN'
162                         'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
163                         'BEGIN' 'IF' I > 1 'THEN' GENERATE(ADDRESS);
164                                 N:=ARRAY[ADDRESS];
165                                 'IF' N=0 'THEN'
166                                 'BEGIN' KIND:=0;PM:=PM+1;'GOTO' READY'END' 'ELSE'
167                                 'FOR' NI:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
168                                 'IF' ARRAY[ADDRESS+NI]=SYMBOL 'THEN'
169                                 'BEGIN' KIND:=1;CIRCLE:=I;'GOTO' READY 'END'
170                         'END';
171                         KIND:=-1;
172                 READY:  ARRAY[SYMBOL+CYCLE]:=CIRCLE;
173                         ARRAY[SYMBOL+CLASS]:=KIND
174                         'END'
175                 'END'
```

```
176         'END' CLASSIFY;
177
178         'PROCEDURE' GENERATE(ADDRESS);'INTEGER' ADDRESS;
179         'BEGIN' 'INTEGER' N,STEP,NEW,K,I,R,M,J;
180                 N:=ARRAY[ADDRESS];
181                 'IF' N < 0 'THEN'
182                 'BEGIN' STEP:=1;NEW:=BOM 'END' 'ELSE'
183                 'BEGIN' STEP:=-1;NEW:=EOM 'END';
184                 K:=0;
185                 'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
186                 'BEGIN' R:=ARRAY[ADDRESS+I];
187                         M:=ARRAY[R];
188                         'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO'
189                         'BEGIN' K:=K+STEP;
190                                 ARRAY[NEW+K]:=ARRAY[R+J]
191                         'END'
192                 'END';
193                 'IF' NEW+K < BOM ∨ NEW+K > EOM 'THEN'  ERROR('('ARRAY MEMORY OVERFLOW')');
194                 ARRAY[NEW]:=K;
195                 ADDRESS:=NEW
196         'END' GENERATE;
197
198         'PROCEDURE' COMPARE(ADDRESS);'VALUE' ADDRESS;'INTEGER' ADDRESS;
199         'BEGIN' 'INTEGER' N,I,STEP;
200                 N:=ARRAY[ADDRESS];
201                    VITALBW:=ABS(N)=CLASS SYMBOLS(BW,MORTAL);
202                 'IF' VITALEND < VITALBW 'THEN' 'GOTO' END OF JOB
203 ;
204                 'IF' EWNOS= ABS(N) 'THEN'
205                 'BEGIN' STEP:='IF' N < 0 'THEN' -1 'ELSE' 1;
206                         'FOR' I:=1 'STEP' 1 'UNTIL' EWNOS 'DO'
207                         'IF' ARRAY[EW+I]≠ ARRAY[ADDRESS+STEP*I] 'THEN'
208                         'GOTO' READY;
209                         SOLUTION:='TRUE';'GOTO' END OF JOB
210                 'END';
211         READY:
212         'END' COMPARE;
213
214
215         INITIALIZE;
216 START:  PHASE:=1; SOLUTION:='FALSE';JOB:=JOB+1;
217         READ INPUT;
218         COUNT:=0;
219         VITALEND:=EWNOS= CLASS SYMBOLS(EW,MORTAL);
220         COMPARE(BW);
221         PHASE:=2;
222         'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
223         'BEGIN' GENERATE(BW);
224                 COUNT:=COUNT+1;
225                 COMPARE(BW)
226         'END';
227         PHASE:=3;
228         VITALP:=VITALBW;
229         PVR:=P=PM;
230         'FOR' I:=1 'STEP' 1 'UNTIL' PVR 'DO'
231         'BEGIN' GENERATE(BW);
232                 COUNT:=COUNT+1;
233                 COMPARE(BW)
234         'END';
235         NBW:=ARRAY[BW];
```

```
236           'IF' VITALP=VITALBW'THEN' 'GOTO' VARYING 'ELSE'
237           'IF' VITALP > VITALBW'THEN' ERROR('('PROGRAM ERROR')');
238           PHASE:=4;
239   PRODUCE:
240                   GENERATE(BW);
241                   COUNT:=COUNT+1;
242           COMPARE(BW);
243           'GOTO' PRODUCE;
244   VARYING:PHASE:=5;
245           EWHELP:=EW;EWN:=EWNOS;
246           'IF' NBW < 0 'THEN' EOM:=EOM+NBW-1 'ELSE'
247           BOM:=BOM+NBW+1;
248           BEGINPAIR:=BOM;
249           'FOR' I:=1 'STEP' 1 'UNTIL' NBW ,-1 'STEP' -1 'UNTIL' NBW 'DO'
250           'BEGIN' LW:=LETTER:=ARRAY[BW+I];
251                   'IF' ARRAY[LETTER+CLASS]= RECURSIVE 'THEN'
252                   'BEGIN' PHASE:=50000+ ABS(I);
253                           'FOR' J:=1 'STEP' 1 'UNTIL' PM 'DO'
254                           GENERATE(LW);
255                           PL:=ARRAY[LETTER+CYCLE];
256                           'FOR' J:=1 'STEP' 1 'UNTIL' PL 'DO'
257                           'BEGIN' GENERATE(LW);
258                                   'IF' COMPPREFIX(LW,EWHELP,EWN) 'THEN'
259                                   'BEGIN' M:= ABS(ARRAY[LW]);
260                                           EWHELP:=EWHELP+M;
261                                           EWN:=EWN-M;
262                                           ARRAY[BOM]:=COUNT+PM+J;
263                                           ARRAY[BOM+1]:=PL;
264                                           BOM:=BOM+2;
265                                           'GOTO' NEXTLETTER
266                                   'END'
267                           'END';
268                           'GOTO' END OF JOB
269                   'END';
270           NEXTLETTER:
271           'END';
272           PHASE:=6;
273           BOMMIN:=BOM-2;
274           'FOR' I:=BEGINPAIR 'STEP' 2 'UNTIL' BOMMIN 'DO'
275           'FOR' J:=I+2 'STEP' 2 'UNTIL' BOMMIN 'DO'
276           'BEGIN' D:= GCD(ARRAY[I+1],ARRAY[J+1]);
277                   'IF' REMAINDER(ARRAY[I],D) = REMAINDER(ARRAY[J],D) 'THEN' 'ELSE' 'GOTO' END OF JOB
278           'END';
279           SOLUTION:='TRUE';
280   END OF JOB: NLCR;PRINTTEXT('('JOBNUMBER:')');ABSFIXT(2,0,JOB);
281           NLCR;PRINTTEXT('('NUMBER OF GENERATIONS:')');ABSFIXT(5,0,COUNT);
282           NLCR;PRINTTEXT('('PHASE:')');ABSFIXT(5,0,PHASE);NLCR;
283           'IF' SOLUTION 'THEN' PRINTTEXT('(' SOLUTION FOUND.')') 'ELSE' PRINTTEXT('(' NO SOLUTION.')');
284            BW:=END OF MEMORY;
285           EOM:=END OF MEMORY+ARRAY[END OF MEMORY]-1;
286           BOM:=EW+ARRAY[EW]+1;
287           CARRIAGE(10);
288           'GOTO' START
289   'END'
290   'END'
```

```
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=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
```


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.

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=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 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 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/
:/ KHIKDEDKIHK/
JOBNUMBER:  8
NUMBER OF GENERATIONS:      5
PHASE:      2
 NO SOLUTION.
```

```
JOB=E/
:/ KHIKDEKIHKKDEKKGBCFKKDEKKHIKDEKIHK/
JOBNUMBER:  9
NUMBER OF GENERATIONS:      7
PHASE:      2
 SOLUTION FOUND.
```

```
JOB=G/
G/A=>BAC/ C=>CC/ B=>BB/ /
I/A/
:/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
JOBNUMBER: 10
NUMBER OF GENERATIONS:      5
PHASE:      3
 NO SOLUTION.
```

```
JOB=E/
:/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
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	
$\delta^5(\sigma)$	kdek, kgbcfk, kdek	see fig. 1.
$\delta^6(\sigma)$	kgbcfk, khikdekihk, kgbcfk	see fig. 2.
$\delta^7(\sigma)$	khikdekihk, kdekkgbcfkkdek, khikdekihk	see fig. 3.

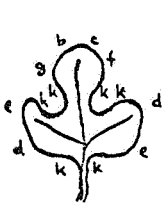


Fig. 1.

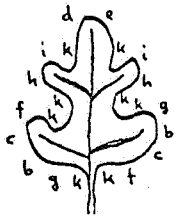


Fig. 2.

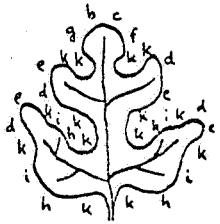
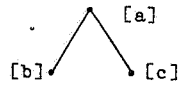


Fig. 3.

Job 10-12. The semi DOL has the following RCS:



hence L(G) is infinite (theorem 3.7 and sequel).

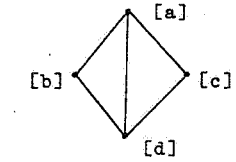
```
JOB=E/
E/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC/
JOBNUMBER: 12
NUMBER OF GENERATIONS:      6
PHASE:      3
 NO SOLUTION.
```

```
JOB=G/
G/ A=>ABCD/ B=>BD/ C=>CD/ D=>D//
B/A/
E/ABCDBDCDDBDDCDDDBDDDCDDDDBDDDDCDDDDDBDDDDDCDDDDDDBDDDDDDCDDDDDDDBDDDDDDDCDDDDDD
DDDBDDDDDDDDCDDDDDDDDDBDDDDDDDDDCDDDDDDDDDDBDDDDDDDDDDCDDDDDDDDDDDBDDDDDDDDDDD
DCDDDDDDDDDDDDDBDDDDDDDDDDDDDCDDDDDDDDDDDDDD/
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]
° ° ° °

```
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

$$|L(G)| = \text{l.c.m.}\,(|[k]|,|[1]|,|[+]|,|[a]|)$$
$$= \text{l.c.m.}\,(2,3,5,7)$$
$$= 210.$$

```
JOB=G  /G/1=>2*4/2=>2/4=>2*5/5=>6#5/6=>7/7=>8/8=>9(1)/9=>9/(=>(/)=>)/ *=>*/
#=>#//8/1/
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.

$\sigma = 1$

$\delta(\sigma) = 2 * 4$

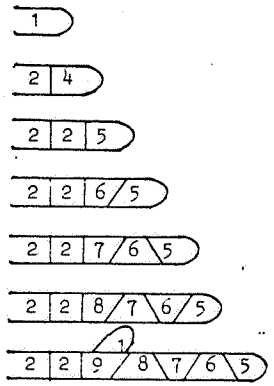
$\delta^2(\sigma) = 2 * 2 * 5$

$\delta^3(\sigma) = 2 * 2 * 6 \# 5$

$\delta^4(\sigma) = 2 * 2 * 7 \# 6 \# 5$

$\delta^5(\sigma) = 2 * 2 * 8 \# 7 \# 6 \# 5$

$\delta^6(\sigma) = 2 * 2 * 9(1) \# 8 \# 7 \# 6 \# 5$



A2317R.J3,PAUL VITANYI

```
     'BEGIN' 'COMMENT' 2317R, AUGUST 1971, PAUL VITANYI AND FRANK GOOSSENS;
   1         'INTEGER' BEGIN OF MEMORY,END OF MEMORY,FIRSTKEY,LASTKEY;
   2                   BEGIN OF MEMORY:=1;
   3                   END OF MEMORY:=30000;
   4                 FIRSTKEY:=0;
   5                   LASTKEY:=127;
   6         'BEGIN' 'INTEGER' 'ARRAY' ADDRESSKEY[FIRSTKEY:LASTKEY]
   7         ,ARRAY[BEGIN OF MEMORY : END OF MEMORY]
   8         ;
   9         'INTEGER' SEPARATOR,EQUALSIGN,TAB,TWNR,SPACE
  10                   ,KEY
  11                    ,S,STEP
  12                    ,B,G,E
  13                    ,P,F,L,FROM,FIRST,LAST,JOB,COUNT,BW,BOM,EOM,I
  14                    ,INFOGRAM
  15                    ;
  16         'BOOLEAN' PRINTER,BEGIN,END,GRAM;
  17
  18         'PROCEDURE' INITIALIZE;
  19         'BEGIN' JOB:=0;
  20                 SEPARATOR:=67;EQUALSIGN:=70;TAB:=118;TWNR:=119;
  21                 SPACE:=93;
  22                 B:=11;G:=16 ;E:=14;
  23                 P:=25;F:=15; L:=21;
  24                 S:=28;
  25                 INFOGRAM:=1;KEY:=1;FIRST:=LAST:=0;
  26         'END' INITIALIZE;
  27
  28         'PROCEDURE' ERROR(STRING);'STRING' STRING;
  29         'BEGIN' CARRIAGE(2);PRINTTEXT(STRING);EXIT
  30          'END' ERROR;
  31
  32         'PROCEDURE' READ INPUT;
  33         'BEGIN' 'INTEGER' CODE;
  34                 'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ EQUALSIGN 'DO'
  35                 'FOR' CODE:= NEXTSYMBOL 'WHILE' CODE ≠ SEPARATOR 'DO'
  36                 'IF' CODE=G 'THEN' GRAM:=BEGIN:=END:= 'FALSE'  'ELSE'
  37                 'IF' CODE=B 'THEN' BEGIN:='FALSE' 'ELSE'
  38                 'IF' CODE=E 'THEN' END:='FALSE' 'ELSE'
  39                 'IF' CODE=P 'THEN' PRINTER:=-FALSE' 'ELSE'
  40                 ERROR('('ERROR JOB CONTROL  INPUT')');
  41                 'IF' GRAM ∧ BEGIN ∧ END ∧ PRINTER 'THEN'
  42                 ERROR('('PROGRAM END')');
  43                 'FOR' CODE:=CODE 'WHILE' ¬GRAM ∨ ¬BEGIN ∨ ¬END ∨ ¬PRINTER 'DO'
  44                 'BEGIN' CODE:=NEXTSYMBOL;
  45                 'IF' ¬GRAM ∧ CODE=G 'THEN'
  46                 'BEGIN' READ GRAMMAR;GRAM:='TRUE' 'END' 'ELSE'
  47                 'IF' ¬BEGIN ∧ CODE=B ∧ GRAM 'THEN'
  48                 'BEGIN' GENERATE BEGINWORD;BEGIN:='TRUE' 'END' 'ELSE'
  49                 'IF' ¬END ∧ CODE=E ∧ GRAM 'THEN'
  50                 'BEGIN' READ COUNT;END:='TRUE' 'END' 'ELSE'
  51                 'IF' ¬PRINTER ∧ CODE= P 'THEN'
  52                 'BEGIN' READ PRINTCOMMANDS;PRINTER:='TRUE' 'END' 'ELSE'
  53                 ERROR('('ERROR IN INPUT')');
  54                 'END'
  55         'END' READ INPUT;
```

```
56
57          'PROCEDURE' READ GRAMMAR;
58          'BEGIN' 'INTEGER' SYMBOL,SYM,HELP,I,J,P;
59                  'FOR' I:=FIRSTKEY 'STEP' 1 'UNTIL' LASTKEY 'DO'ADDRESSKEY[I]:=0;
60                  BOM:=BEGIN OF MEMORY; EOM:= END OF MEMORY;
61                  P:=0; NEXTSYMBOL;
62                  'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
63                  'FOR' SYMBOL:=NEXTSYMBOL 'WHILE' SYMBOL≠SEPARATOR'DO'
64                  'BEGIN' 'IF' NEXTSYMBOL≠EQUALSIGN 'THEN' ERROR('('NO TRANSITION SIGN')');
65                          NEXTSYMBOL;
66                          'COMMENT' ARROWSIGN OF TRANSITIONSIGN IS READ;
67                          BOM:=BOM+INFOGRAM;
68                          ARRAY[BOM]:=0;
69                          ARRAY[BOM+KEY]:=SYMBOL;
70                          HELP:=ADDRESSKEY[SYMBOL]:=BOM;BOM:=BOM+1;
71                          'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ≠ SEPARATOR 'DO'
72                          'BEGIN' ARRAY[BOM]:=SYM;BOM:=BOM+1 'END';
73                          ARRAY[HELP]:=BOM-HELP-1;
74                          P:=P+1;
75                          'COMMENT' GRAMMARRULE COUNTER;
76                  'END';
77                  J:=1;
78                  'FOR' I:=1 'STEP' 1 'UNTIL' P 'DO'
79                  'BEGIN' J:=J+INFOGRAM;
80                          HELP:=ARRAY[J];
81                          'FOR' SYM:=1 'STEP' 1 'UNTIL' HELP 'DO'
82                          'BEGIN' SYMBOL:=ADDRESSKEY[ARRAY[J+SYM]];
83                                  'IF' SYMBOL=0 'THEN' ERROR('('SYMBOL IN GRAMMAR NOT DEFINED')')
84                                  'ELSE' ARRAY[J+SYM]:=SYMBOL
85                          'END';
86                          J:=J+HELP+1
87                  'END';
88          'END' READ GRAMMAR;
89
90          'PROCEDURE' READ WORD(BADDRESS,DIRECTION,EADDRESS);
91          'VALUE'         DIRECTION;'INTEGER' BADDRESS,DIRECTION,EADDRESS;
92          'BEGIN' 'INTEGER' SYM,HELP;
93                  BADDRESS:=EADDRESS;
94                  EADDRESS:=EADDRESS+DIRECTION;
95                  NEXTSYMBOL;
96                  'COMMENT' SEPARATOR OF CONTROLWORD IS READ;
97                  'FOR' SYM:=NEXTSYMBOL 'WHILE' SYM ≠ SEPARATOR 'DO'
98                  'BEGIN' HELP:=ADDRESSKEY[SYM];
99                          'IF' HELP=0 'THEN' ERROR('('SYMBOL IN WORDNOT IN GRAMMAR')') 'ELSE'
100                         ARRAY[EADDRESS]:=HELP;
101                         EADDRESS:=EADDRESS+DIRECTION
102                 'END';
103                 ARRAY[BADDRESS]:=EADDRESS-BADDRESS-DIRECTION
104         'END' READ WORD;
105
106         'PROCEDURE' GENERATE BEGINWORD;
107         'BEGIN' EOM:=END OF MEMORY;
108                 READ WORD(BW,-1,EOM);
109         'END' GENERATE BEGINWORD;
110
111         'INTEGER' 'PROCEDURE' NEXTSYMBOL;
112         'BEGIN' 'INTEGER' SYM;
113         NEXT:   SYM:=RESYM;PRSYM(SYM);
114                 'IF' SYM=SPACE ∨ SYM=TAB ∨ SYM=TWNR 'THEN' 'GOTO' NEXT
115         ;NEXTSYMBOL:=SYM
```

```
116          'END' NEXTSYMBOL;
117
118          'PROCEDURE' GENERATE(ADDRESS);'INTEGER' ADDRESS;
119          'BEGIN' 'INTEGER' N,STEP,NEW,K,I,R,M,J;
120                  N:=ARRAY[ADDRESS];
121                  'IF' N < 0 'THEN'
122                  'BEGIN' STEP:=1;NEW:=BOM 'END' 'ELSE'
123                  'BEGIN' STEP:=-1;NEW:=EOM 'END';
124                  K:=0;
125                  'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL' N 'DO'
126                  'BEGIN' R:=ARRAY[ADDRESS+I];
127                          M:=ARRAY[R];
128                          'FOR' J:=1 'STEP' 1 'UNTIL' M 'DO'
129                          'BEGIN' K:=K+STEP;
130                                  ARRAY[NEW+K]:=ARRAY[R+J]
131                          'END'
132                  'END';
133                  'IF' NEW+K < BOM ∨ NEW+K > EOM 'THEN'  ERROR('('ARRAY MEMORY OVERFLOW')');
134                  ARRAY[NEW]:=K;
135                  ADDRESS:=NEW
136          'END' GENERATE;
137
138          'PROCEDURE' PRINT WORD(BW);'VALUE' BW;'INTEGER' BW;
139          'BEGIN' 'INTEGER' I,N;
140                  N:=ARRAY[BW];
141                  NLCR;
142                  'FOR' I:=1 'STEP' 1 'UNTIL' N,-1 'STEP' -1 'UNTIL'N 'DO'
143                  PRSYM(ARRAY[ARRAY[BW+I]+KEY]);
144                  NLCR;
145          'END'PRINT WORD;
146
147          'PROCEDURE' READ COUNT;
148          'BEGIN' 'INTEGER' SYM;
149                  NEXTSYMBOL;
150                  COUNT:=READ(SYM);
151                  'IF' SYM ≠ SEPARATOR 'THEN' ERROR('('NO SEPARATOR AFTER COUNTCOMMAND')')
152          'END' READ COUNT;
153
154          'PROCEDURE' READ PRINTCOMMANDS;
155          'BEGIN' 'INTEGER' SYM;
156                  NEXTSYMBOL;
157                  SYM:=NEXTSYMBOL;
158                  'FOR' SYM:=SYM 'WHILE' SYM ≠ SEPARATOR 'DO'
159                  'IF' SYM=F 'THEN' FIRST:=READ(SYM) 'ELSE'
160                  'IF' SYM=S 'THEN' STEP:=READ(SYM) 'ELSE'
161                  'IF' SYM=L 'THEN' LAST:=READ(SYM) 'ELSE'
162                  ERROR('('ERROR IN PRINTSPECIFICATION')')
163          'END' READ PRINTCOMMANDS;
164
165          'INTEGER' 'PROCEDURE' READ(X);'INTEGER' X;
166          'BEGIN' 'INTEGER' SYM;
167                  SYM:=0;
168                  'FOR' X:=NEXTSYMBOL 'WHILE' DIGIT(X) 'DO'
169                  SYM:=SYM*10 + X;
170                  READ:=SYM
171          'END' READ;
172
173          'BOOLEAN' 'PROCEDURE' DIGIT(X);'VALUE'X; 'INTEGER'X;
174          DIGIT:= X > -1 ∧ X < 10;
175
```

```
176
177             INITIALIZE;
178     START:  JOB:=JOB+1;PRINTER:='TRUE';
179                     FIRST:=LAST:=0;
180             STEP:=0;
181             PRINTTEXT('(' JOBNUMBER: ')');ABSFIXT(2,0,JOB);
182             NLCR;
183             READ INPUT;
184     CARRIAGE(3);
185             FROM:=COUNT-LAST;
186             'FOR' I:=1 'STEP' 1 'UNTIL' COUNT 'DO'
187             'BEGIN' GENERATE(BW);
188                     'IF' I<FIRST+1 v I>FROM v I=COUNT v REMAINDER(I-FIRST,STEP)=0
189                     'THEN' PRINT WORD(BW)
190             'END';
191     CARRIAGE(10);
192             EOM:=END OF MEMORY + ARRAY[END OF MEMORY] - 1;
193             'GOTO' START
194      'END'
195     'END'
```

```
 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

OPQPQOO

9V*V*WXV*V*WX99

TTYGER,TYGER,BURNINGBRIGHTTTYGER,TYGER,BURNINGBRIGHTTT
```

```
 JOBNUMBER:   2

JOB=GP/
G/ A=>B/ B=>AB //
B/ A/
E/10/
P/ F1 S10 L4/


B

BABABBABABBABBABABBAB

ABBABBABABBABBABABBABABBABBABABBAB

BABABBABABBABBABABBABABBABABBABBABABBABABBABABBABABBAB

ABBABBABABBABBABABBABABBABBABABBABBABABBABABBABABBABABBABABBABABBABABBABABBABABBABABBAB
```

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
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

JOBNUMBER: 5

```
JOB=GP/
G/ A=>ABCD/ B=>BD/ C=>CD/ D=>D//
B/A/
E/10/
P/ F3 S7/
```

```
ABCD

ABCDBDCDD

ABCDBDCDDBDDCDDD

ABCDBDCDDBDDCDDDBDDDCDDDDBDDDDCDDDDDBDDDDDCDDDDDDBDDDDDDCDDDDDDDBDDDDDDDCDDDDDDDDBDDDDDDDDCDDDDDDDDDBDDDDDDDDDCDDDDDDDDDD
```

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
```

```
 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
```