THE UNIVERSITY OF MICHIGAN
COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Computer and Communication Sciences Department

AUTOMATON SELF-REFERENCE

Richard Alan Laing

July 1977

Logic of Computers Group
Computer and Communication Sciences Department
Technical Report No. 204

with assistance from:

National Science Foundation Grant No.
MCS76-04297

School of Advanced Technology
State University of New York (Binghamton)

ABSTRACT

AUTOMATON SELF-REFERENCE

Richard Alan Laing

Chairman: Howard Pattee

In this paper an abstract, deterministic, discrete kinematic automaton system for expressing machine computation, construction, and self-inspection has been designed. Burks's conjecture that a machine can by self-inspection obtain its own complete structural description and store the information sufficient to recover this description, within a proper part of itself is confirmed. We also exhibit a machine which though not initially equipped with a structural description of itself, can reproduce itself, thus providing a counterexample to a conjecture of von Neumann and confirming a conjecture of Arbib.

AUTOMATON SELF-REFERENCE

BY

RICHARD ALAN LAING

B. A., The University of Michigan, 1950
M. A., The University of Michigan, 1953

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Advanced Technology
in the Graduate School of the
State University of New York
at Binghamton
1977

ACKNOWLEDGEMENTS

My friends and associates among the faculty, students and staff of the School of Advanced Technology, State University of New York, Binghamton are gratefully thanked for suggestions and comments on the research reported here. Howard Pattee, Chairman of my doctoral committee, and Andrew Barto have my special thanks in this regard. I am also grateful to the other members of my committee, Dennis Geller and Michael Arbib (and to temporary member Thomas Plum). I wish to thank Dennis Geller in particular for his early, powerful, and persistent championing of this whole endeavor. Michael Arbib has my sincere gratitude for his willingness to participate on the committee despite his heavy obligations to his own research and his own university.

I would also like to thank the past and present members (senior, junior, and staff) of the Logic of Computers Group; almost all I have learned of theoretical computer science has been a consequence of my association with them. The extremely prompt and efficient attention given the preparation of this dissertation by Janet Walters, Logic Group Secretary, made possible its successful completion under unusual conditions.

Financial assistance of the National Science Foundation through Grant MCS76-04297 to the Logic of Computers Group, as well as graduate assistant financial support from the School of Advanced Technology is gratefully acknowledged. Special thanks goes to Robert Hoogstoel of the School of Advanced Technology for his expediting the complicated administrative and academic aspects of my association with the School.

TABLE OF CONTENTS

Chapter One. Introduction

1.1 Background

In this paper we consider what a discrete, deterministic system can come to know of itself. The line between what a system can come to know of itself, and what it cannot come to know, has never been precisely and completely drawn.

There are of course many important partial results, among them that a mathematical machine can read and interpret descriptions of machines, yet there is no machine which in all cases can tell what a described machine would do, (Turing, 1936), and that a machine capable of constructing other machines can construct all "passive" machines, but not necessarily all active machines (von Neumann, 1966; Moore, 1970).

A machine examining itself to discover its own complete description would seem inevitably to entail an instance of an active machine acting upon an active machine. For this reason von Neumann (1966, p. 122) conjectured, that a machine could not obtain its own complete description. He also believed that for a machine to reproduce (in the sense intended by him) it would require a complete description of itself. Thus, he further conjectured (*op. cit.*) that no system (machine or organism) *not* initially supplied from the outside with a description of itself, could reproduce. Burks (1961), on the other hand, conjectured that a machine *could* by self-inspection acquire its own complete *structural* description (and furthermore could store this information within a proper part of itself). Since a complete *structural* description would be sufficient information for construction of any (passive) machine, self-reproduction in the sense of von Neumann *would* then be possible, despite the initial absence of a complete self-description. This conjecture, that self-reproduction would be possible *sans* possession of an original complete self-description, was made explicit by Arbib (1966, p. 217).

1.2 Principal Results

The principal results of this paper are the proof of Burks' conjecture that a machine system *can* unaided obtain its own complete structural description and store within itself the information sufficient to recover the description of itself, and the consequent disproof of von Neumann's conjectures that a machine cannot obtain its own description and thus cannot reproduce if not initially equipped with a complete description (and thus a confirmation of Arbib's conjecture *contra* von Neumann).

More specifically, the main results of this paper are the following:

1) The design of an abstract, deterministic, discrete *kinematic automaton system* (KAS) for expressing machine computation, construction, and self-inspection.
2) Given any Turing machine there is a KAS machine which can simulate the computation of the Turing machine.
3) Given any Turing machine, there is a KAS machine which can simulate the computation of the given machine and moreover, such a KAS machine can in addition produce a complete structural description of itself.
4) There is a KAS machine which given the description of any

(passive) KAS machine can construct the described machine.

5) There is a KAS machine which can reproduce itself.

6) There is a KAS machine which given any (single string, passive) KAS machine can read the machine and produce a description of it.

7) There is a KAS machine which by self-inspection can obtain its own complete structural description and store the information sufficient for recovery of the description within a proper part of itself.

8) There is a KAS machine which, though not initially equipped with a description of itself, can reproduce itself.

(Of these results, 2) is a direct consequence of the design of the KAS, a design which incorporates the "programmed Turing machine" concept of Wang (1957) for which the universality of computation results have been shown; 3) is a direct translation of results by Lee (1963) and Thatcher (1963) into this new KAS format; 4) is a translation of von Neumann's result (1966) on universal construction to this KAS format; 5) follows directly from the von Neumann results on self-reproduction as modified by Thatcher (1970); 6) is an essentially new result, *viz.* a sort of universal reading of machines by machines is possible; 7) our principal result (section 4.15) is a counter-example to von Neumann's conjecture on obtaining self-descriptions by self-inspection, and a confirmation of Burks' conjecture (1961) to the contrary; 8) is a counter-example to von Neumann's conjecture (1966) on self-reproduction *sans* description, and a confirmation of Arbib's surmise (1966).)

Comment. A *machine* is composed of a finite number of primitive, activatable components drawn from a fixed finite number of primitive component types interconnected in a specific manner. A *structural description* of a machine will be any fixed finite object composed of a finite number of primitive component types which bears sufficient information to specify the type and position of each primitive component of the machine to be described.

Thus, a description of a machine could take the form of a passive "blueprint" specifying the type and location of each primitive component of the described machine, or it could take the form of a set of instructions for the identification and placement of each primitive component of the described machine. The notion of description, in its full generality, can take on many other guises. Thus, a description of a machine could be an exact duplicate of the machine, or a description of a machine could itself be an activatable machine which constructs as output the machine described, or which produces the "blueprint" giving the location and type of all primitive components of the described machine, or produces the set of instructions for the assembly of the described machine. Note also that a machine which describes or constructs any of *these* machines can in turn also be considered as a description of the original machine. Thus, for example, a machine M might be said to have as its description, a machine A which produces the constructor B of a machine C which lists the instructions for constructing M.

That a description of a machine can take so many forms, some of them quite cryptic, is important to the understanding of our central results. For the essential logical difference between our results and those of von Neumann (1966) and of Lee (1961) and Thatcher (1963, 1970) is that their construction and description processes require the initial presence, in some form, of a different structural description of each different machine which is to reproduce or to describe itself, while the principal results to be presented here do not require such a different initial description of each new machine. To distinguish our results from those of von Neumann, Lee, and Thatcher, we must convince the reader that there are no essential structural descriptions of each new proposed self-inspecting machine available in concealed form somewhere in the machine itself. We will try to secure this conviction by carefully spelling out and fixing the properties of all the sub-machines of which a machine is composed *and* then always permitting the machine to be augmented by an *arbitrary* finitely large additional submachine, which will clearly have no counterpart elsewhere in the initial machine.

(It should perhaps be explicitly pointed out that in none of these cases of the use of descriptions by machines is it intended or implied that a machine requires or possesses knowledge of the *meaning* or *use* of the primitive components described, or any information on the conventions which are assumed in the operation of the system as a whole. The capacity of a system to secure and employ *this* sort of information about itself, though of great interest and importance, is not treated by von Neumann, or by Lee or Thatcher, and is not explicitly treated in this paper.)

1.3 Organization of the Paper

In the next chapter (Chapter Two) we describe the KAS in terms of its primitive constituents, their properties, and the general organization of the constituents into activatable strings. In

Chapter Three, we show how we may compose our primitives to produce the repertoire of particular basic KAS machines and routines we shall employ in obtaining our principal results. In Chapter Four we present our major results on self-description (4.15) and reproduction (4.21) in machines; in Chapter Five we outline some additional results and indicate some directions for further research.

Chapter Two. The Kinematic Automaton System and its Primitive Constituents

2.1 Background

In this paper, the general problems raised (automaton description, construction, self-reproduction, etc.) and the strategies which might be employed in attacking these problems have their origin in von Neumann's work on machine self-reproduction (von Neumann, 1966) in a cellular automaton regular array format. As to the particular machine system in which our results will be exhibited, the KAS, it has its origin in part in von Neumann's (never completely explicated) kinematic machine system (von Neumann, 1951) and in my own notion of generalizing the concept of a Turing machine to permit machine construction and analysis (Laing, 1975).

A Turing machine is usually conceived of as an abstract, discrete, deterministic information processing device consisting of a finite-state *read-head* mounted on an indefinitely extendible *tape* marked into squares, each square capable of containing one of a finite number of tape *symbols*. The read-head may, as a function of the state it is in and the tape symbol under scan change the tape symbol (or leave it unchanged), move one square to the right or left (or remain in place), and change internal state (or remain in the same state). It is generally agreed that such an information processing system captures, in precise terms, the intuitive notion of an algorithm or effective calculation.

In this paper, the "classical" Turing machine format is extensively modified.

2.2 Kinematic Automaton System

More particularly, a KAS machine will consist of basic simple finite state automaton *primitives*, organized into *strings*. These strings will be finitely long, but may be indefinitely extended. The primitives (and consequently the strings) possess a *forward* and *backward* direction. The specific machines used in this paper consist of at most *two* separate strings of interacting primitives. In the way we shall use them, at any one time *at most one* of the two possible strings will possess an active primitive. That the active and passive roles of strings may be *exchanged* is an important feature of this machine system. (An ordinary Turing machine in this system would appear as a two string machine, one string of which consists entirely of "tape" primitives and always remains passive; see Figure 1. Figure 2 shows a KAS machine in which both strings contain non-tape primitives.) In operation, the two strings are always in sliding contact at a single point. One typical action is for the activated primitive of the active string to act upon the contacted passive primitive so as to change its state, and then (automatically) relinquish activation and contact to its own next neighbor primitive in the forward direction of the active string. (The required changing of local relationships between the contacted primitives can be taken as a basic assumed property of the system in the same manner in which implementation of the details of tape positioning in a conventional Turing machine are taken as unanalyzed basic operations. In order to make clear the actions required however, we, in Figure 3, describe in detail the changing of activation and contact.)

Figure 1. A Turing machine in the Kinematic Automaton System (KAS) format. The lower string plays the role of the finite state read-head (in the Wang program form) while the upper string plays the role of the Turing machine tape.

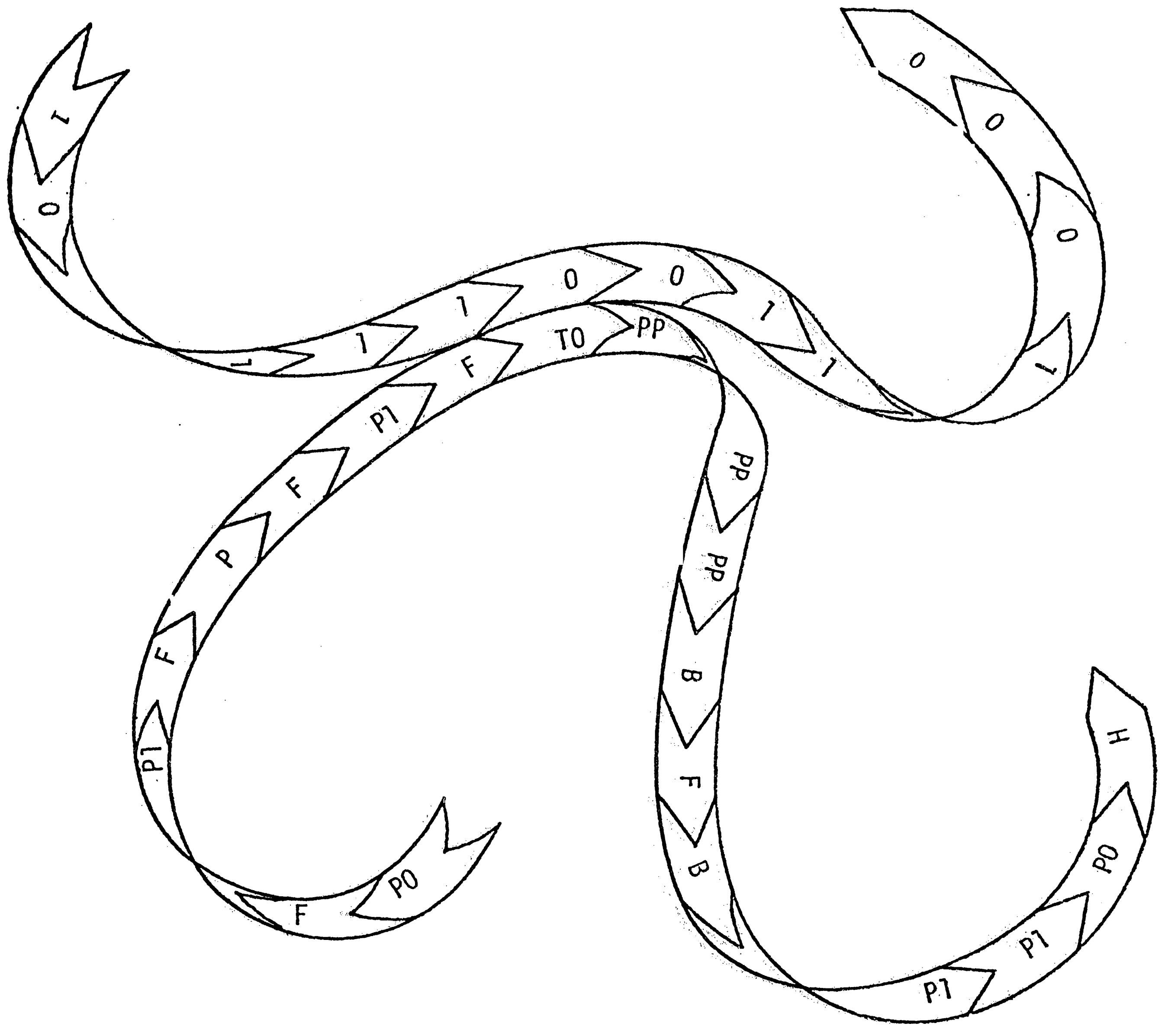
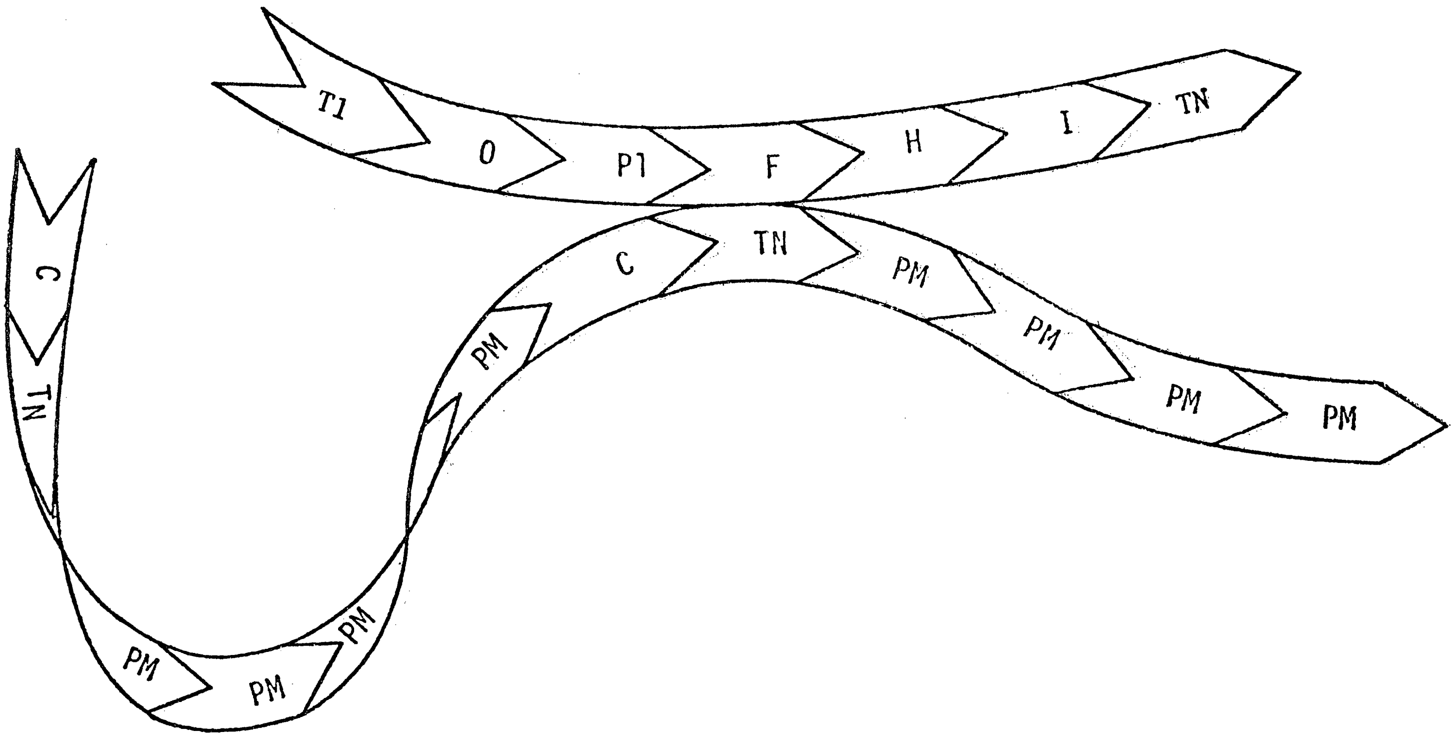

I
0
1
1
1
0
0
1
1
1
0
0
0
F
P1
F
P
F
P1
F
TO
PP
PP
PP
B
F
B
P1
P1
PO
H
F
PO

Figure 2. A KAS machine in which both strings contain activatable primitive constituents, and the roles of read-head and tape can be exchanged.


T1
O
P1
F
H
I
TN
C
TN
PM
PM
PM
PM
C
TN
PM
PM
PM
PM

2.3 Primitive Constituents

The primitives of which our machines are composed possess *simple, local* automaton properties. Employing these primitives we show how various routines, organs, etc. can be designed. We then combine these routines and organs to exhibit machines capable of sophisticated analysis and construction behaviors.

2.3.1

Three primitives N (*Null*), 0 (*Zero*), 1 (*One*) will principally be employed in the passive recording of information. (In an ordinary Turing machine, these primitives would play the part of the tape symbol alphabet.) If in the present system an N, 0, or 1 should be part of an *active* string and undergo activation, it will merely pass activation and contact to its next forward direction neighbor, thus behaving as a "no operation" primitive. The N or null primitive has some special properties. It is assumed to be a basic unit freely available in the environment. The construction process assumes that primitives can be recruited for use at the construction site (usually at the ends of passive strings) and the construction process may assume (in the absence of a specific test) that the recruited type is an N. Thus the N plays a role similar to that of a "blank" square in an ordinary Turing machine tape.

We will also adopt the convention that an N which is at the end of a string and is not specially marked, and is not in contact with a primitive of the opposing string will automatically be detached and thus be returned to the environment of the system. (See 2.3.8 for discussion of the use of this property in end locating and place holding.)

2.3.2

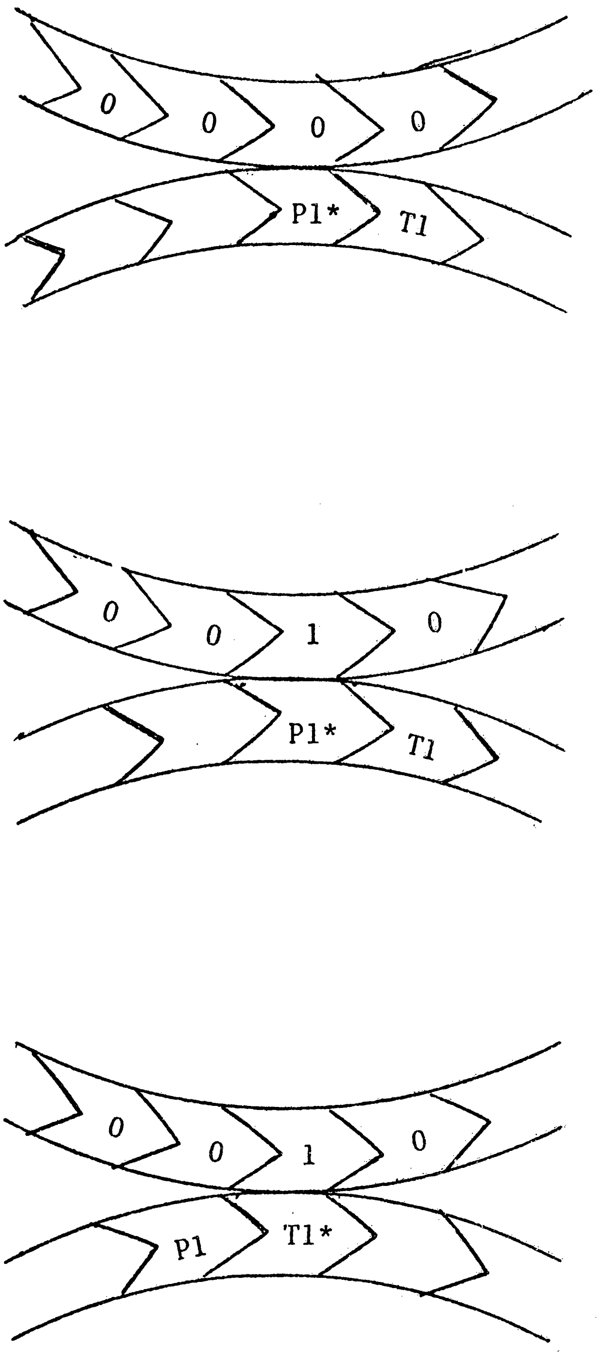
The three primitives PN (*Print Null*), PO (*Print Zero*) and P1 (*Print One*) will principally be employed in acting upon N, 0, and 1 primitives, to change their states. A PN will act upon an N, 0, or 1 to make it an N; a PO will act upon an N, 0, or 1 to make it a 0; and a P1 will act upon an N, 0, or 1 to make it a 1. After completion of this action, a PN, PO, or P1 will automatically relinquish activation and contact to its next neighbor primitive in the forward direction in its string. Thus the PN, PO, P1 primitives play a role similar to the *print* and *erase* operations of an ordinary Turing machine.

2.3.3

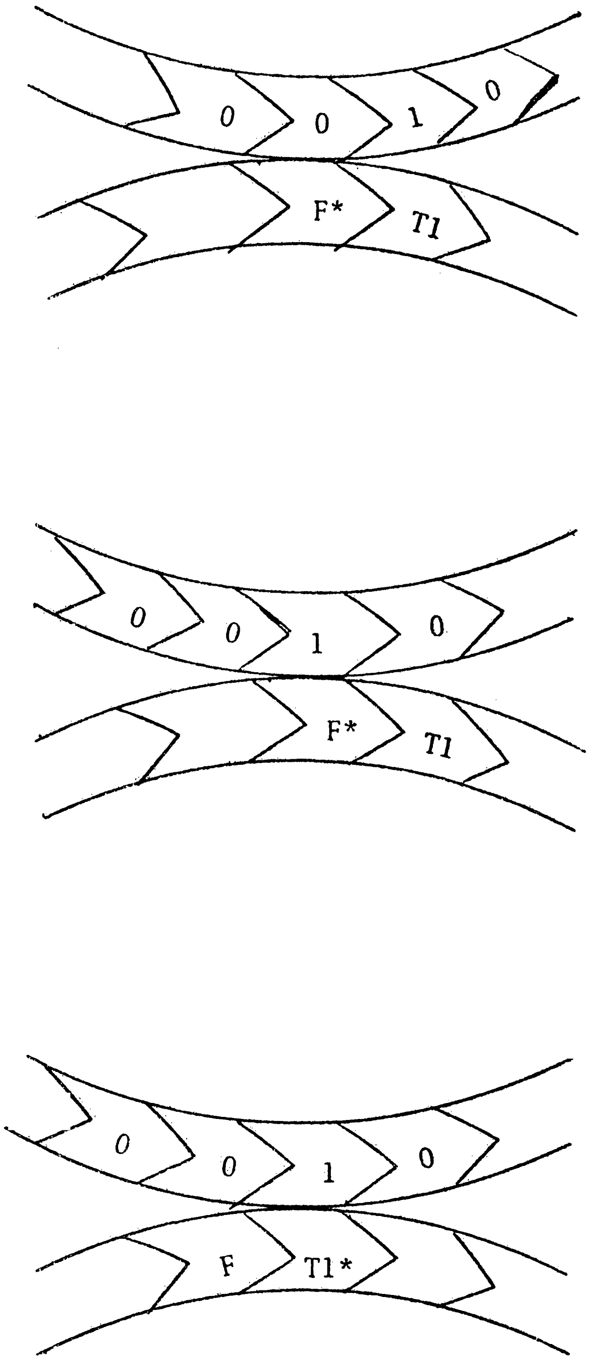
The two primitives F (*Forward*) and B (*Backward*) cause the active string to slide to the next primitive of the passive string in the forward or backward direction, respectively. If application of an F or B primitive would result in a physical disengagement by "running off the end" of the passive string, an N primitive is by convention assumed to be automatically attached to the end of the passive string. This is analogous to the automatic addition of blank squares to a Turing machine tape. The F and B primitives thus play the part of the "move right" and "move left" operations of an ordinary Turing machine. After completion of the move action, an F or B primitive automatically relinquishes activation to its next neighbor primitive in the forward direction in the active string.

Figure 3. Action between Strings Illustrated. There are two basic forms of shift of string contact and activation: one in which, after characteristic primitive action the activation and contact is shifted to the next neighbor primitive in the active string (as in the action of a P1 primitive) and one in which not only is activation and contact shifted to the next neighbor but an *adjacent* primitive of the passive string must have first been accessed (as in the action of a F primitive). i. An activated P1 in contact with a 0 changes the 0 to a 1 and then passes activation and contact with the new 1 to its next neighbor T1. ii. An activated F in contact with a passive primitive shifts contact to the next primitive of the passive string and concludes by shifting activation and contact to its own next neighbor.

i.



ii



2.3.4

The primitive H (*Halt*), if activated, terminates all activity. (Basically the H primitive is included to establish an analogy to certain related computational systems where an explicit signal of completion of the computational activity may be desirable.)

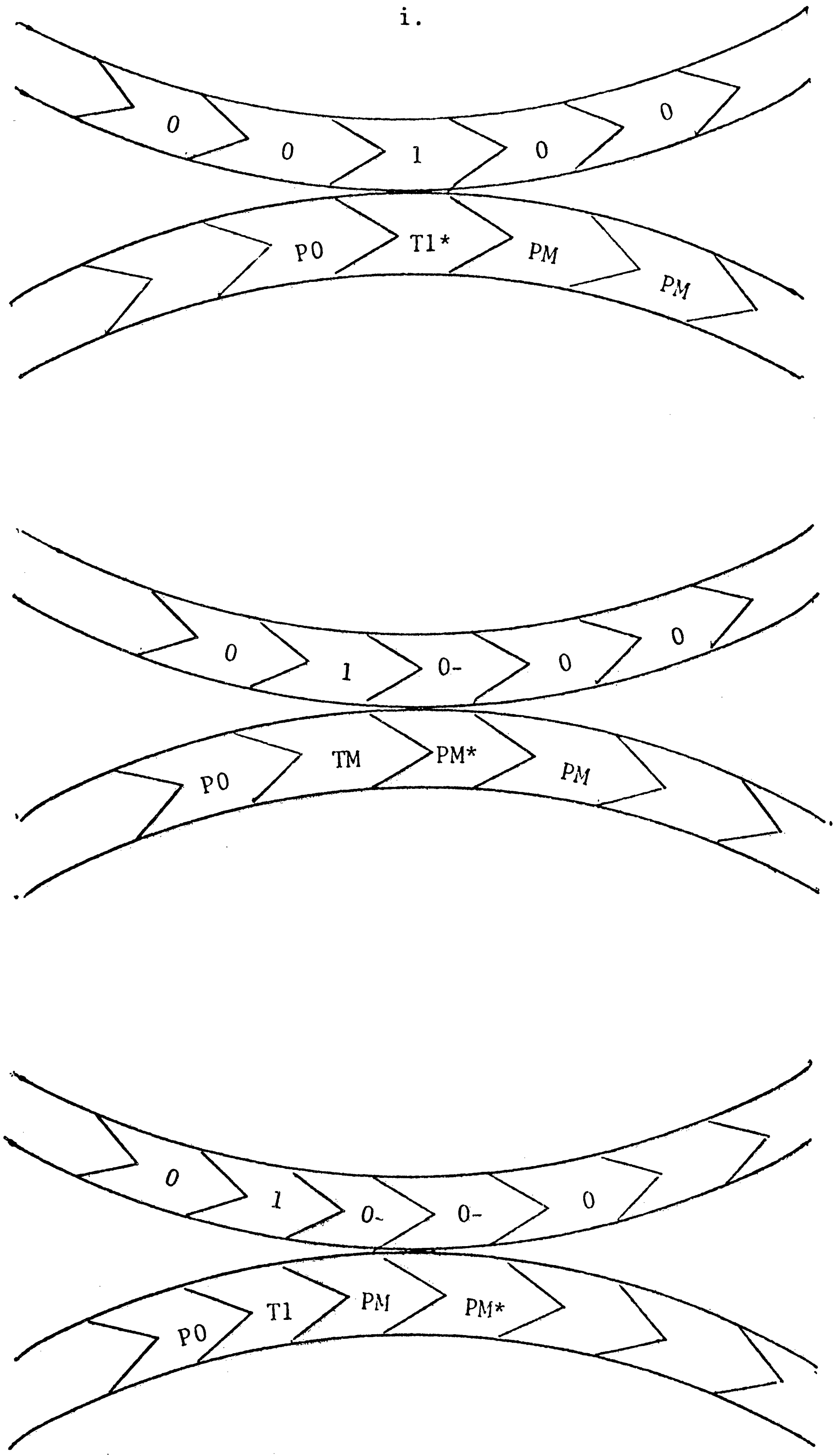
2.3.5 Transfers

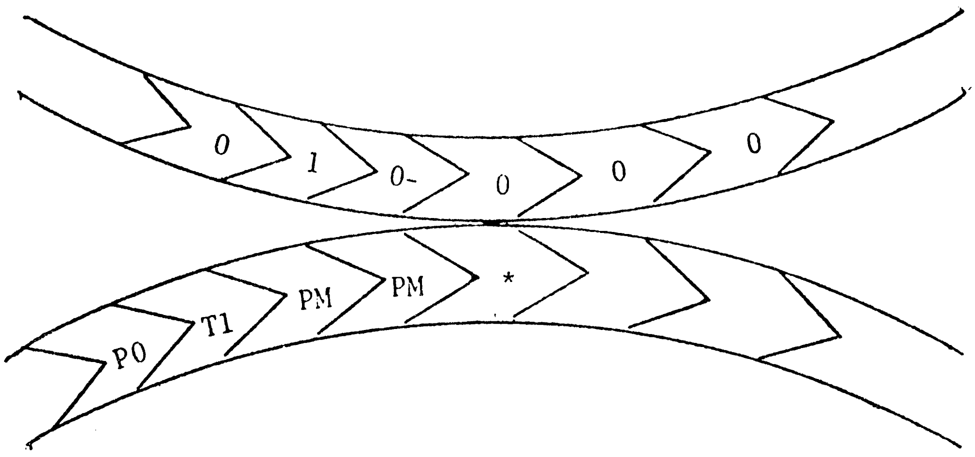
There are three *Conditional Transfer* primitives TN, T0, T1 (where contact with an N, 0, or 1, respectively, is the condition to be satisfied) and one *Unconditional Transfer* primitive T. In a machine, each of the transfer primitives will immediately be followed by a fixed finite sequence of all PP (*Print plus*) primitives or all PM (*Print minus*) primitives (to be defined below), the number of PP or PM primitives in the sequence being used to specify the location of the primitive in the active string to which activation is to be shifted. In operation, if the transfer primitive is activated and is an unconditional T, or if in the case of a TN, T0, T1 the conditio is satisfied, then the transfer primitive *specially* activates the initial primitive of its associated PP or PM string of primitives. The string of PP or PM primitives then superimposes (in the *forward* direction) upon successive primitives of the passive string a fixed finite number of special *plus* or *minus* marks. These marks in no essential way alter the basic passive primitive types upon which they are temporarily superimposed. They *do* cause ordinary (non-PP or PM) activated primitives which come in contact with them to behave in a special fashion. An active primitive in contact with a special marked passive primitive will suspend its usual actions and instead will remove one of the special superimposed marks from the passive string, move *backward* in the passive string to the next special marked passive primitive, and relinquish activation and contact to its own next neighbor (in the *forward* direction if the sequence of special marks consists of *pluses*, and in the *backward* direction if the marks are *minuses*). Thus, while detecting the special plus (or minus) marks, and removing them one by one while shifting activation in a forward (or backward) direction through the ordinary primitives of the active string, the usual active string action is suspended until all the superimposed marks have been removed from the passive string. At that time, the first active primitive ready to resume normal computational action will be a number of ordinary primitives distant from the site of the original transfer primitive and the newly activated primitive will be in contact with the passive primitive which was the occasion for the transfer in the first place. Thus, any transfer primitive need only have an appropriate length string of PM or PP primitives associated with it, to be able to shift activation to any desired ordinary primitive in the forward or backward direction on the active string.

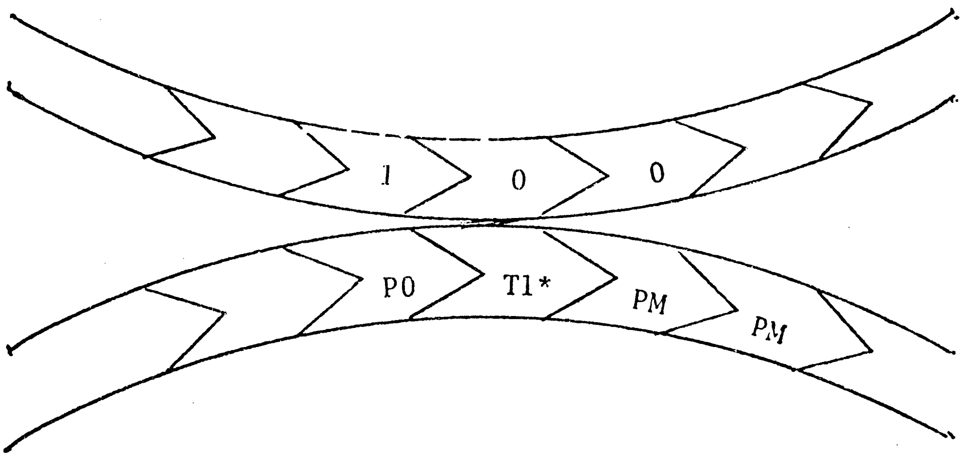
In implementing our technique of transfer it should be noted that PM or PP primitives which are not *specially* activated by a transfer primitive take no part in ordinary computational action or in transfer counting: activation and contact is merely passed through them (in the forward or backward direction) to the next ordinary primitive. It should also be pointed out that if the passive string has insufficient numbers of primitives to accomodate the complete PM or PP marking, the forward moves off the end of the passive string will result in

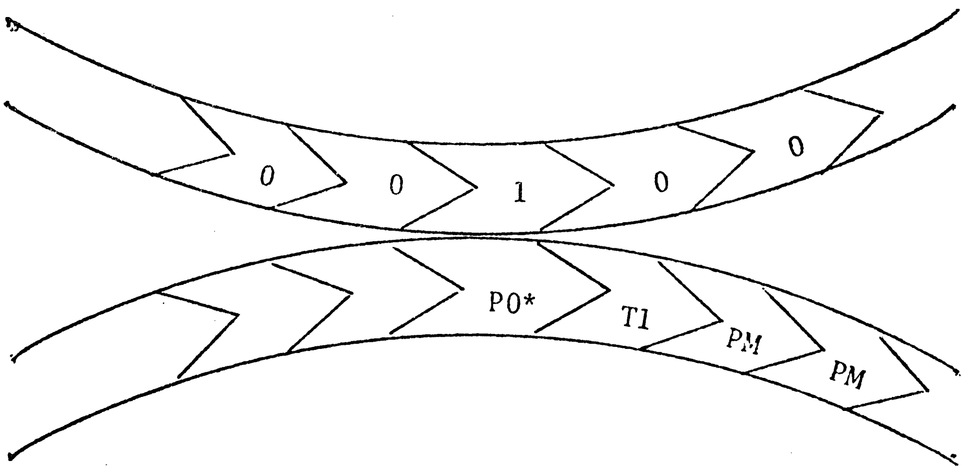
Figure 4. Illustration of the Transfer Implementation. i. A "transfer on one" primitive has its condition satisfied by contact with a one primitive of the passive string. The transfer primitive passes activation and contact to a length 2 sequence of PM (print minus) primitives with which it is associated. The PM primitives mark two successive primitives with "minuses". When activation and contact is shifted to the next (non-transfer marking) primitive, one mark is removed and activation and contact is routed *backward* completely through the entire transfer region of PM primitives, the remaining special marks not being altered. The T1 removes the (in this case) sole remaining special mark and passes activation and contact with the *next* backward direction passive primitive (the originally contacted primitive) to its back direction active string neighbor. Regular computation now proceeds with the new active primitive in contact with the passive primitive which occasioned the transfer in the first place. A transfer n places backward from the transfer point in an active string can thus be implemented by a n+1 sequence of PM primitives.

i.



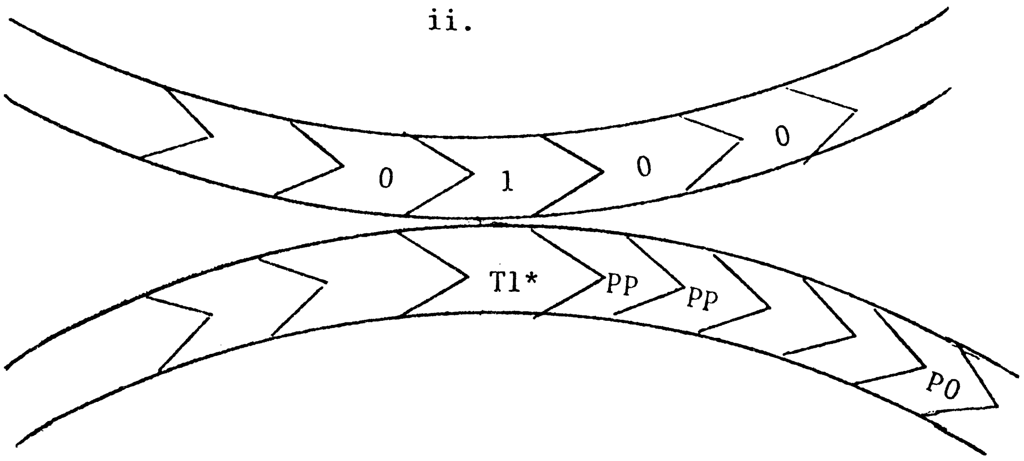

0
1
0-
0
0
0
PO
T1
PM
PM
*

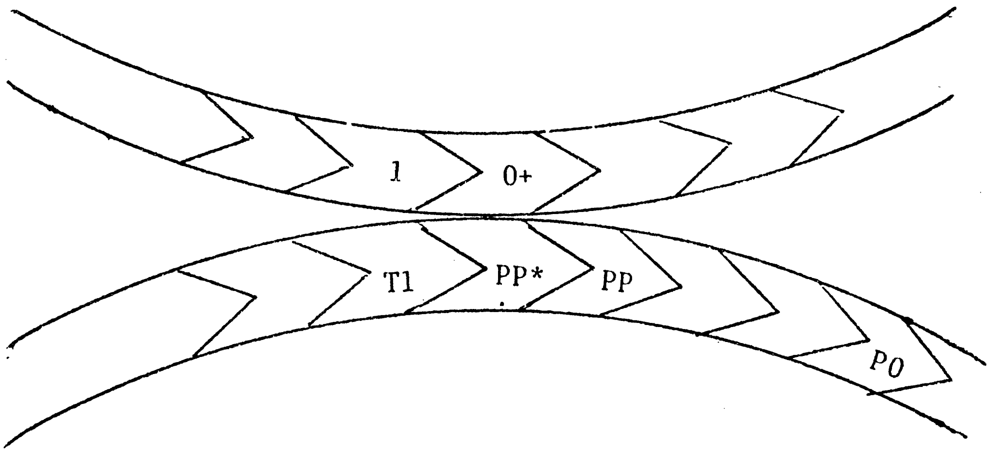

1
0
0
P0
T1*
PM
PM

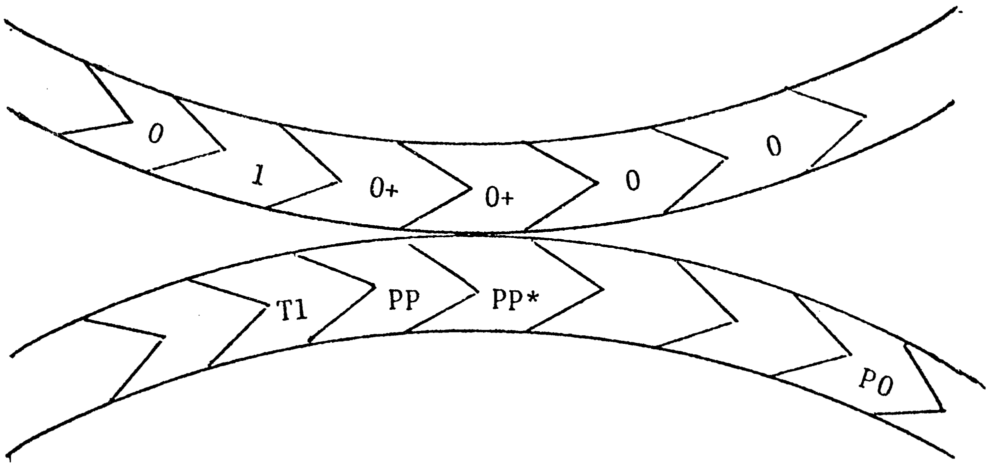

0
0
1
0
0
P0*
T1
PM
PM

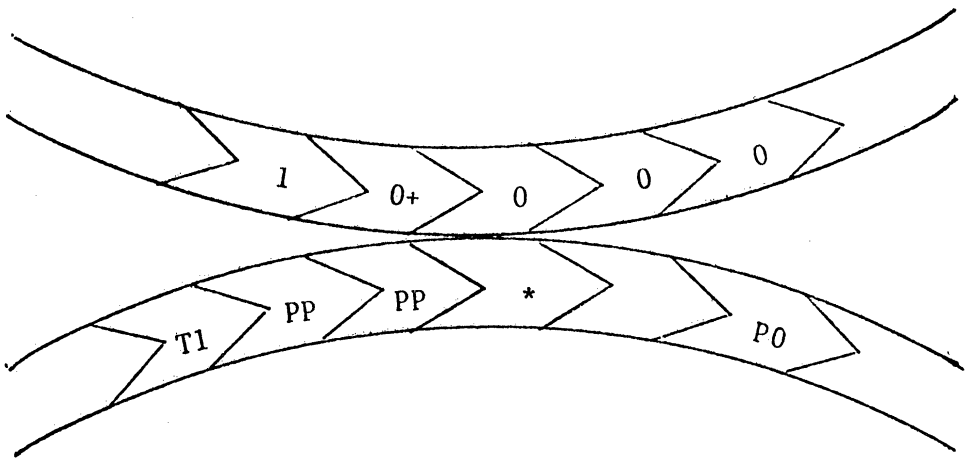
ii. A "transfer on one" primitive has its condition satisfied by contact with a passive one primitive. The transfer primitive passes activation and contact to a length two sequence of PP (print plus) shift mark primitive. These two primitives place plus marks on the next two successive passive primitives. Activation and contact is then passed on to the next forward direction primitive of the active string. Each primitive of the active string removes a plus mark, shifts contact backward in the passive string, and passes activation and contact in the forward direction in the active string. Ordinary operation resumes with an active primitive which is 3 places forward in the active string; thus activation can be transferred to a primitive n places forward in the active string by a n-1 long sequence of PP primitives (none being required if the transfer primitive and its target primitive are immediately adjacent).

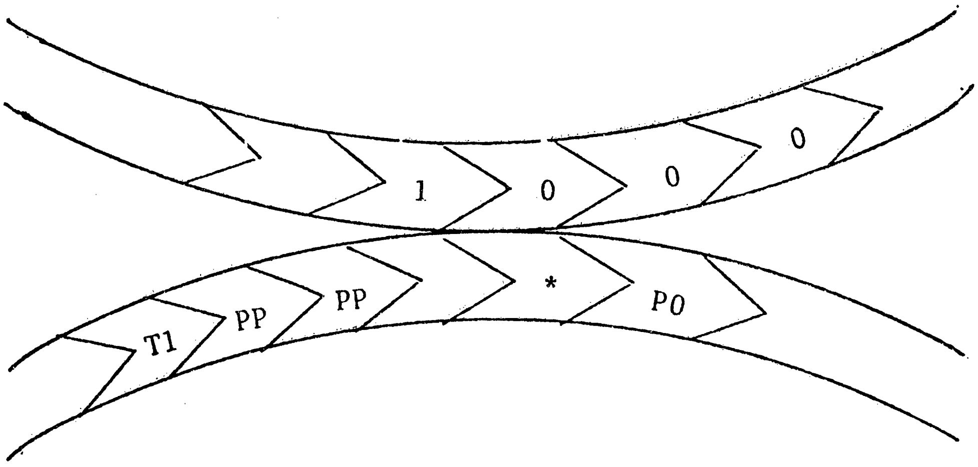
ii.

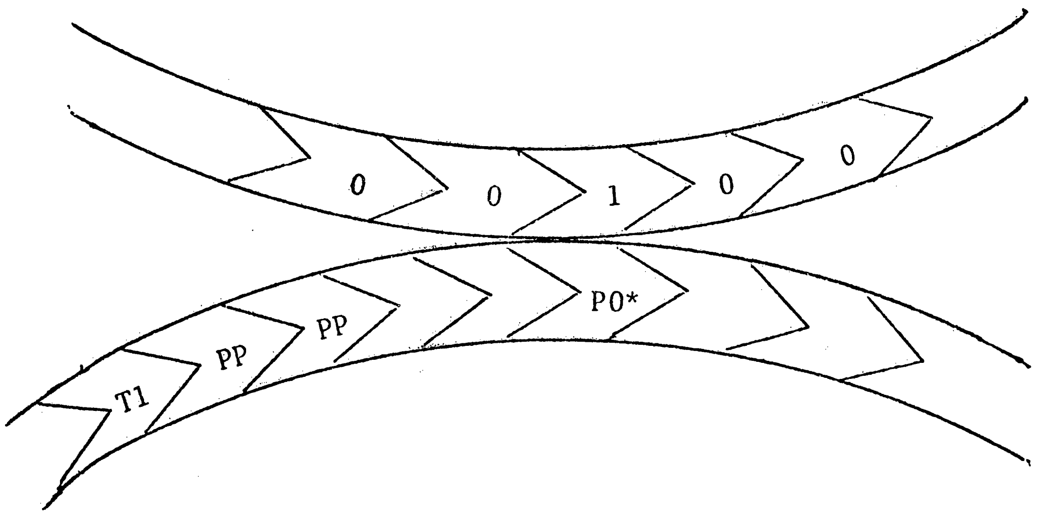







1
0+
0
0
0
T1
PP
PP
*
P0


1
0
0
0
T1
PP
PP
*
P0


0
0
1
0
0
T1
PP
PP
P0*

the recruiting and marking of additional N primitives at the end of the passive string. When the active transfer process retreats from these newly created N primitives, removing the transfer marks, the recruited Ns will be detached. (See 2.3.1) The transfer technique is illustrated in Figure 4.

This technique for transfers has been adapted from one suggested by Wagner (1967) and is also closely related to the technique of transfers designed by Arbib (1966) in which a separate third string of modules is employed in implementing the transfer shift count. Additional transfer implementation techniques for kinematic machines are discussed in Laing (1975) and Laing (1976).

2.3.6 Activation Primitive

The A or *activation* primitive will, when itself activated by a neighbor in its own string, produce an activation in the *passive* primitive with which it is in contact. (We will at times actually employ several variant kinds of activation primitives. These include a primitive which will produce an activation in the passive primitive with which it is in contact and an immediate *loss* of activation in itself. The effect then of this primitive is to produce an exchange of the active and passive roles of the strings. Another activation primitive will activate the passive primitive with which it is in contact and also bring about a *separation* between the originally active and newly activated strings; thus the effect would be to create newly independent activated strings.)

2.3.7 Conversion

In our system we will want to take recruited N primitives and convert them to any other specific primitive type; this is the basis of *construction*. In our system we will also want to take any arbitrary primitive type and be able to reduce it to an N; this is the basis of our *destruct* as well as our *analysis* procedures. (A *destruct* procedure takes any given primitive or string of primitives, and reduces it to the N status. An analysis procedure takes any given unknown primitive or primitive string and identifies the primitive types. The analysis procedure, which is at the heart of our principal results, is described in section 3.7 *et seq*; the destruct procedure is described in section 3.11 *et seq*.) The C (Conversion) primitive is at the heart of these procedures. C primitives can be used to convert any passive primitive (including even another C primitive) with which it is in contact into another (passive) primitive type. The steps by which this is carried out are set forth in Table 1. For example (beginning at the top of the table) an activated C applied to a (passive) N converts it to a 0; if a 0 is subjected to an activated C primitive, the 0 is converted to a 1, etc. Notice that the conversion sequence forms a closed loop: some sufficient, specific number of activated C applications will convert any primitive type to any other type (including returning a primitive back to its original type). (This information could also be expressed as an automaton state transition diagram in which an input of repeated contacts with activated C primitives, produces a closed cycle of automaton state changes.)

<u>Comment</u>. The conversion primitive is meant to stand for a

process (perhaps very much more complex than that set forth in Table I) by which, starting with one or a few available primitive types, the system could produce primitive constituents of the sort it itself was composed of, and which, starting with more complex entities, could reduce them to the basic type or types once more. This synthesizing and decomposing could take many forms, and what we have presented here is, designedly, especially simple.

In the von Neumann system, special signals are employed to convert a recruited quiescent cell to any other cell type; any cell type can be subjected to a sequence of special signals which will reduce it once more to the quiescent state. The von Neumann synthesis procedure is *not* backwards traceable in the strong sense in which ours is, so that it is not possible immediately to enlist von Neumann's procedure in our analysis process (spelled out in Section 3.7). It seems likely though that a regular cellular array system could be designed to possess the desired analysis as well as synthesis properties.

It is also possible to divorce the synthesis and analysis procedures. We might create desired primitives by the C conversion or the von Neumann special signal process, and *analyze* primitives in an entirely different fashion. For example, we might employ the notion that each primitive type possesses a distinct "signature", an aspect or aspects which can be read and thus permit identification. In molecular biology, the anti-codon region (Watson, 1976, p. 306) of a tRNA molecule can for example be looked upon as the signature for the attached amino acid, or the variable region in an antibody molecule for the type (or class) of antigen with which it reacts.

TABLE I

C Conversions

C(N)	→	0
C(0)	→	1
C(1)	→	PN
C(PN)	→	P0
C(P0)	→	P1
C(P1)	→	F
C(F)	→	B
C(B)	→	H
C(H)	→	T0
C(T0)	→	T1
C(T1)	→	TN
C(TN)	→	T
C(T)	→	PP
C(PP)	→	PM
C(PM)	→	A
C(A)	→	C
C(C)	→	N

In our system, we could thus agree that each primitive type has a distinct code word composed of discernible a and b characters exposed in it. Among our primitives would be Ta and Tb primitives which would detect a's and b's on the surfaces of primitives, and, by a series of transfers, "decode" the names of types, reaching a distinct region of the machine for each type decoded, a region "standing" for the knowledge of the primitive type examined.

2.3.8 Place Holders and End Markers

In our system we must make use of the notion that the system can remember what primitive in a passive string was being read, and that the system can return to that primitive after an interlude of computation elsewhere. In the Turing machine notion this usually means the use of a *special* tape symbol (by itself or superimposed on any ordinary symbol) or the use of coded blocks of the ordinary symbols. In our system we have already introduced the notion of special marks produced by the PP and PM primitives, and to which the ordinary activated primitives are sensitive. This notion could be used to implement the "mark and return" process we will require. Such a system could be made explicit by means of a fixed finite set of M (mark) TM (test for mark) and DM (delete mark) primitives. If a certain location is to be "remembered" it is given a mark by an M primitive, and location is tested for by a TM, and restored to usual status by a DM action.

Detecting the end of a passive string poses similar difficulties. We have introduced the notion that if an F or B action should threaten to bring about disengagement of the active and passive strings, a N primitive will be recruited to the end of the passive string. (This is of course analogous to the addition of blank squares at the end of the Turing machine tape.) We could make the detection of end of the passive string more explicit by having a special TE (test for end) primitive which behaved differently when the passive primitive in contact was the last (ordinary) primitive or was a *special* end marker primitive. Alternatively we could have *any* ordinary primitive behave in a special fashion in the presence of either an (ordinary) last primitive, or a special end marker primitive. Note that in the case of a special end marker primitive, it would have to be converted or detached if the passive string was to be extended from that point, and then would have to be re-attached or constructed, or recruited again to serve its end mark function (or could be assumed to do so automatically). The general point to be made is that any of several augmentations of the system would serve the purpose of implementing desired actions at the end of the passive tape.

Since however, in the results of importance in this paper, our requirements for end markers and place holders will be quite modest, we will not augment our present repertoire of primitives but will instead employ the following strategy. Since an N will be attached to the end of a passive string whenever an F or B action would carry us off the string, we can devise a "search for end" routine in which we shift in the F (or B) direction and test (by means of a TM) for the arrival of the final N. It should be pointed out that use of the newly recruited N as an end marker obliges us to constrain carefully the creation and use of N primitives *within* the string. This will pose no problem however in the cases with which we are here concerned.

In addition, as to place holders, we can, in the following fashion, again make use of the N primitive. In all the cases considered in this paper we must "remember" at most two places *within* a passive string. (Place holders are usually required when the machine must at one site sequentially read information, one primitive or one code block at a time, and must then move and locate another distinct site (possibly indefinitely far removed) where the information acquired at the first site is to be acted upon. Thus, we may have to have a place holder at each of the two distinct indefinitely separated internal sites.) We can mark an internal reading site, as we leave it, by making the last read primitive an N (and since the system can always be designed to remember a *fixed* finite amount of information, it can, upon return to this N, restore the primitive to its original status). Similarly, the site of distant activity can also be marked by an N replacing the original primitive at that point, while the system "remembers" the status of the original primitive at that site. (Notice that in the special case where the marked primitive is at the end of the string, converting it to an N will result in its detachment. In returning to this location however, seeking for the (now missing) N, the N will "automatically" be restored.)

2.3.9 Transfers of Activation Between Strings

We shall frequently have occasion to transfer activation from one of the two possible strings of our machines to the other string, thus exchanging the active and passive roles. The activation itself is produced by means of the A primitive. It is also important that the to-be-activated string receives its activation at the appropriate location; that is, the initial primitive of the proper sub-routine of that to-be-activated string must be locatable by processes within the presently activated string. It will often be the case that at most one distinct sub-routine of the to-be-activated string must be located; in some cases it can be arranged that the target sub-routine is at the "top" of the passive string or in some other distinct spatial position; in such cases the active string need only seek out the spatial location of the passive string and relinquish activation to it.

Since however we may wish to be able to activate any finite number of different sub-routines of a passive string, we will describe two general methods for carrying out the locating and activating process.

One way in which this communication of activation to specific location can be handled is to arrange that the passive string possess a series of transfer points at distinguishable locations (successive primitives at the head of the passive string, for example) and that the active string locate the first, second, third, etc. transfer point, as necessary, from which the relinquished activation and contact would be carried to the desired location within the newly activated string. A second way in which the activation can be communicated to a specific location is by means of an "address". Each routine which must be specifically activated will be prefaced by a unique 0, 1 address. The address in the passive string will be located by the active string beginning at the head of the passive string and searching for the desired address (by means of T0, T1, tests and transfers). When the desired address is reached, the discovered location is activated.

An important consideration which must be kept in mind in implementing such exchanges of activation, is that the locating routines of the active machines depend on some prior knowledge of the structure of the passive string. In most cases this knowledge can be made available because the active string itself will have been the creator of the passive string (*and* since activation may have to be *returned* to the originally active string, the original string must design the offspring string to implement this also).

Chapter Three. Some Basic Kinematic Machines

3.1 Introduction

In this chapter we describe some basic kinematic mechanisms and routines. In Chapter Four we will combine these mechanisms and routines in various ways to exhibit our principal results for self-description and self-reproduction.

3.2 Fixed Sequence Emitter

A fixed sequence emitter is a string consisting entirely of P0, P1, and F primitives. When activated, such a string of primitives (acting on an existing string consisting entirely of N, 0, or 1 primitives or recruiting N primitives one-by-one with each F application) will produce a particular sequence of 0 and 1 primitives (possibly with some N primitives interspersed). In this paper we shall usually consider only *standard* emitters in which *single* P0s or P1s are prefaced by single Fs. Clearly for any particular fixed, finite passive sequence of 0 and 1 primitives desired, there is a standard emitter which can be designed to create the desired sequence.

Example: The standard emitter F-P1-F-P0-F-P1 would (beginning its action upon an empty string) yield the sequence of primitives 1-0-1.

3.3 Descriptions

Since our kinematic machines are composed of a fixed finite number of primitive types, we can assign a unique fixed finite length code word of zeroes and ones to each primitive type. For example, if we had 16 different primitive types, we might assign N = 0000, 0 = 0001, 1 = 0010, P0 = 0011, P1 = 0100, F = 0101, etc. The *explicit description* of a kinematic machine will be a string of 0 and 1 primitives standing for the successive code words of each primitive of the machine string in order (starting with the primitive which has a successor but no predecessor); the explicit description of a *machine* composed of more than a single string, will consist in the descriptions of each of the separate strings. The description of a multiple string machine can be a *single* 0,1 string, providing we introduce a code word signalling the end of one string description and the beginning of the next. For any (fixed finite) machine, a fixed sequence emitter can be designed which will print out the explicit, 0,1 description of the machine.

Comment. Although by *description* we shall usually mean the 0,1 string of agreed upon code word equivalents for each of the successive primitives of a machine, our discussion in the Comment of Section 1.2 should be kept in mind: the information sufficient to specify the sequence of primitives of a machine can be carried in many forms, and the description in an agreed upon uniquely decipherable 0,1 code is only a very explicit and obvious form of description.

3.4 Emitter Inferrer

An emitter inferrer is an active string which can examine a passive zero-one description (a sequence of zero-one primitives) and infer the composition of the standard fixed sequence emitter which (could have) produced it. The inferrer works as follows. It reads the first primitive of the passive description string (by means of

T0 and T1 primitives which can detect the presence of 0 and 1 primitives, respectively). If for example the first primitive is a *one* (as in the string produced in the example above) the inferrer "knows" that it had to have been produced by a P1 primitive present as the first significant primitive of a standard emitter. (Similarly, if the first description primitive was a *zero* it must have been produced by a P0, as the first significant primitive of the emitter.) If the description continues (there is a second zero or one) it must have been a consequence, first, of the application of an F primitive, followed by a P0 or a P1 according as the second description primitive was a zero or a one. Proceeding in this fashion the inferrer can deduce the complete sequence of primitives of the standard emitter which could produce the description sequence of primitives.

3.5 Emitter Describer

As an emitter inferrer deduces each of the emitter primitives in turn (by examining the sequence of zeroes and one primitives produced by the emitter) a transfer can be made to a location where the P0, P1, or F deduced can have *its* description spelled out by a special-purpose emitter. For example, if the inferrer detects a *one* it could only have been produced by a P1; a P1 is described by 0011 and this can be produced by an emitter F-P0-F-P0-F-P1-F-P1.

3.5.1 A Detailed Example of the Inference and Description Process

Since the inference-description routine is important to the Lee-Thatcher result and in turn to the form of presentation of our central description by self-inspection result, we will present a detailed explication of the procedure.

The inference and emitter-description routine will operate under the assumption that the 0,1 string presented to it was produced by a standard emitter operating on an initially empty string; the job of the inference and description routine is to produce, beginning at the end of the presented 0,1 string, a string of additional 0,1 primitives which is the description (in the agreed upon 0,1 code) of the successive primitives of the assumed standard emitter of the presented string.

The inference routine begins by seeking out the first primitive of the submitted 0,1 string. It does this by moving in the backward direction, testing for the arrival of an N primitive. (This N will be produced by the first B application which would otherwise have disengaged the active string from the top of the passive string.) The inference routine now moves forward one primitive, and reads that primitive (the first of the submitted 0,1 string). (The forward move will result in the detachment and loss of the newly acquired final N whose presence marked the front end of the original string.) The routine now makes the newly read primitive of the 0,1 string an N, and the routine moves to the opposite end of the submitted 0,1 string. (The 0 or 1 which was converted to an N will be detached and lost as soon as contact with it is abandoned, but it will be restored upon return to the location.)

The routine, bearing the identification of the first primitive, determines the bottom end of the submitted string, moves back one primitive to ascertain what the *last* primitive of the 0,1 string is, and notes this for later use; it then changes this last primitive to an N, and moves forward (recruiting an additional N). The inference routine now enters into its *description* sub-routine. It transfers to an emitter which produces (in 0,1 code) the code word (in zero, one primitives) for an F and the code word for the P0 (or P1) which (under the standard emitter assumption) must have produced the first primitive.

The routine now searches in the backward direction, for the *second* N (the first N encountered in the backward direction will mark the end of the originally submitted string; the second N encountered will mark the location of the last read primitive of the original string). When the second N is encountered, it is converted back to its original 0 or 1 status, and the next 0 or 1 is read and replaced by an N.

The routine, bearing the identification of the second primitive of the submitted string, now searches for the *second* N in the forward direction (the first N to be encountered marks the last primitive of the original string, a primitive whose actual 0 or 1 identification we continue to preserve). Encountering the second N (at the open bottom end of the passive string) the routine emits the code word of primitives for an F and a code word for the P0 (or P1) which produced the second primitive of the submitted string.

This process continues until in the reading of a "next" primitive, it is identified as an N (rather than as a 0 or 1). This signifies that the *last* primitive of the original string has been reached. At this point the N is restored to its original 0 or 1 status (we have been retaining this piece of information) and in a final excursion to the bottom end of the passive string the code word for an F and the required P0 or P1 is produced. This completes the process of inference and emitter description. Beginning with a 0,1 passive string we end with the original string having been augmented with a 0,1 string which describes the successive Fs and P0s or P1s which, under the standard emitter assumption, could have produced the originally submitted string.

3.6 Special-Purpose Constructor

Beginning with "null" primitives recruited from the environment, or assuming a passive string of all null primitives, a "standard" active program string consisting of proper sized (of length 0 to 16) blocks of C primitives prefaced by single F primitives can be designed to construct any fixed finite string.

Example. The constructor F-C-C-C-C-C-F-C-C-C-C-C-C-F-C-C-C-C-F C-C-C-C-C-C-F-C-C-C-C-C will, starting with a string of N primitives produce the sequence P1-F-P0-F-P1 (ie. the emitter of the earlier example).

Comment. Notice that both fixed sequence emitters and special-prupose constructors make the assumption that their action is directed only toward certain forms of strings. The emitter action is taken toward an empty string or one consisting of N, 0, and 1 primitives only, and the constructor action is assumed to be directed toward an empty string.

If an emitter begins to work on a string which is *not* all 0 or 1 primitives, then the desired sequence may not actually be produced in the passive string. However every emitter action could be prefaced by a *destroyer* action (see 3.11 and 3.12) reducing all primitives of the passive string to N, and thus dispersing them, so that emitter action would begin by recruiting new N primitives. Similarly for a constructor; a constructor assumes that it starts with N primitives. If the passive primitives in contact are *not* N primitives, then an "incorrect" (though systematically related) string will be produced. This difficulty also can be obviated by prefacing the constructor with a *destroyer* which reduces all primitives to N and (with the exception of the single contacted N) disperses them.

3.7 Analyzer

An analyzer is an active string which presented with any (passive) string can analyze and identify the primitives of the passive string one-by-one, and produce a description of the passive string. An analyzer will consist of a series of C primitives, each C followed by a TN primitive. For each primitive type there is (according to the C-conversion table) a unique number of C stimulations which will reduce the type to an N. Converting the unknown primitive one step at a time and each time testing (with a TN) for the arrival of the N status, the machine can determine what the type was before the test began. Once knowing the type, the machine can transfer to a routine for emitting the description of that type. (This analysis process "destroys" the analyzed string, by reducing it to all N status; but see 3.9 below.)

<u>Example</u>. In the analyzer segment TN-C-TN-C-TN-C-TN-C-TN... (with associated PP or PM primitives omitted) the first TN will detect if the unknown primitive was an N to start with, the second TN will

detect if the unknown primitive was a C, the third TN will detect if the unknown was an A, etc. (See the conversion table.)

3.8 Analysis and Universality

The notion of analysis is an extremely powerful one. Turing showed that a special computing machine (a "universal" machine) could read and interpret a description of any computing machine of the whole class. J. von Neumann showed that in a system in which *construction* of machines was possible there is a *universal constructor* which, given a description of any (passive) machine in the class under consideration, can construct that machine. Our analysis result says that there is an analyzing machine such that given any (passive single string) machine of the class under consideration, the analyzer can "read" the machine and produce its description (and can, as we shall see, if properly equipped, go on to interpret the description or construct a copy of the machine, etc.). It should also be noted that as a machine is analyzed the structural information obtained can be acted upon immediately; an explicit intermediate description composed of zero-one primitives is not always required, since the information, embodied in terms of an activation at a location can be converted directly into a course of action.

3.9 Analyze and Restore

After having been analyzed to determine its type the originally unknown primitive will have been reduced to the N type. It may be useful or important to restore the analyzed primitive to its pre-analysis type. This can be carried out by stimulating the primitive with the appropriate number of C conversions actions to convert an N back to the original type. This process can be carried out because the system will know what the primitive now is (an N), what it was and should be restored to (it has just discovered this), and the number of C stimulations necessary to convert an N to any desired primitive type. (This information is displayed in the C-conversion table and can be embodied in the machine structure.)

Example. If the analyzer has just established that the unknown primitive was an A (ie. exactly *two* C stimulations have converted the unknown to an N), then the TN which detected this N status can route activation and contact to a *restore* routine C-C-C-C-C-C-C-C-C-C-C-C-C-C-C (fifteen C stimuli) which will convert the present N back to the A it was before the analysis of it began.

3.10 Inferrer of Special-Purpose Constructor

As with the standard emitter of a 0,1 description, whose structure can be inferred from the resulting description, so can the structure of a standard special-purpose constructor be inferred from the machine constructed. The procedure is as follows. Recall that a standard special-purpose constructor consists of an initial F primitive (which will recruit an N) followed by a sequence of C primitives of the length required to convert the N to the desired primitive, followed by a second F (to recruit the next N to be acted upon) followed by the next block of C primitives, etc. In the inference process, such a machine constructed by a standard special-purpose constructor is analyzed, primitive-by-primitive.

The analyzer knows that every special-purpose constructor begins with an F. When the analyzer determines what the first primitive of the constructed machine is, it knows how many C stimulations were necessary to convert an N to the discovered primitive type, thus it knows how many C primitives there must have been in the first special-purpose constructor block of C primitives. Similarly for the next primitive: there is a unique number of C stimulations which produced it. By this means, an inference routine can by examining a machine, determine the structure of the standard special-purpose constructor which produced the machine.

3.10.1 Inference and Description

Once the inference routine has analyzed a primitive and established the number of C primitives required to produce that primitive from an initial N, the inference routine can make use of this information immediately or can produce a 0,1 *description* of the C primitives which were inferred. By alternating the analyzing and inferring with describing, a machine can be examined, and a description of the standard special-purpose constructor which produced the machine can be set down.

Example. Suppose an inference machine is given the string P1-N-P0 and is to infer the (standard) special-purpose constructor which (could have) produced it. The analyzer determines that the first primitive is a P1 and (from the conversion table) this means that *five* C stimuli are required to produce a P1 from an N. The description (in the agreed upon zero-one code) of *five* successive C primitives is then set down. Attention then shifts to the second primitive, N. The analyzer would immediately (upon the first TN test) determine that this is an N, and, since the construction process begins with the assumption of an N, the action of *no* C primitives is required and so only the description of the F primitive necessary to bring about the constructor shift of attention to the next primitive is required. Following analysis of the P0 primitive, the F (required in the constructor to move to the next N primitive to be converted) is described, then the number of C primitives necessary to convert an N to a P0 is described, etc. The constructor of P1-N-P0 is thus F-CCCCC-F-F-CCCC, and can be converted into a zero-one description.

Comment. As noted the inferrer of a special-purpose constructor assumes that the constructor is of the "standard" form of blocks of C primitives (including blocks zero in length) separated by F primitives. There are of course many other ways in which strings could be constructed, and for some of these other methods inferences could be designed.

3.11 Special-Purpose Destroyer

For any particular passive string, an active machine string can be designed which will reduce each primitive of the passive string to an N primitive.

Example. Given the passive string T-A, it can be reduced to N-N by the active string CCCCC-F-CC.

3.12 General Destroyer

Any string can be reduced to all N primitives by subjecting each primitive to a succession of stimulations and testing for the arrival of the N status.

3.13 Dispersion

Recall that by convention any N primitive which is unmarked by any transfer symbol, is not in contact with the active primitive of the opposing string, and is at the end of a string, will be detached. Thus, destroyer action which reduces primitives to the N status, also automatically can produce the dispersal of the resulting N primitives.

3.14 Loop Transfer Constructor

An active string can be designed which will take as input a string which is any program, and will append to the bottom of it, a transfer to the top. It does this by first constructing a transfer primitive at the end of the string, then examining the string over its whole length (distinguishing PP and PM primitives from ordinary primitives) and by this ascertaining the number of PM primitives necessary to transfer control to the primitive at the top of the program; the transfer constructor then appends to the transfer this number of PM shift mark primitives.

3.15 Review of Some General Properties of the Kinematic Machine System

1) The tape-like primitives and the program-like primitives need not be permanently segregated in separate strings. The active and passive roles of the strings can be exchanged. (This contrasts with the usual Turing machine system with its fixed roles for automaton and tape.)

2) The major components of our system being connected *strings* of

simpler primitive automata, one string can, by its primitive automata shifting contact to successive next neighbors, obtain direct access to any primitive of another string. (This contrasts with von Neumann's two-dimensional "checkerboard" cellular automaton system, where access to interior cells could be gained only by penetrating intervening exterior cells of the machine. It should be mentioned that by employing a *third* dimension, a two-dimensional regular cell-space could be explored without disrupting intervening machine structure. In effect the KAS system employs a two-dimensional space to explore one-dimensional machines.)

3) At any one time only a *single* primitive automaton need be activated in order to carry out the actions necessary for obtaining the results to be presented. (This restriction to a single activation site (in the Turing machine manner) again contrasts with the von Neumann cellular system where although the general course of system action (in, for example, the process of self-reproduction) is sequential, in implementing many sub-processes, activation is sometimes simultaneously present at many separated locations.)

4) The assumption is made that available to the system is a population of passive *null* primitive automata. These null or N primitives are recruited at the open ends of passive strings. (This property is analogous to the "automatic" addition of blank squares at the ends of a Turing machine tape, and related to the notion in the von Neumann cellular system of a machine being surrounded always by an environment of quiescent cells available

for exploitation.)

5) By a series of actions directed at null primitives, these primitives can be converted to *any* of the other primitive constituent automaton types. (This is analogous to the capacity for systematic conversion of quiescent cells of the von Neumann cellular system to *any* cell type whatsoever. This contrasts with the usual Turing machine notion in which capacity to alter the status of recruited blank tape squares is restricted to a special tape square symbol set.)

6) The process by which a null primitive can be converted to any other primitive type, including returning to the null type itself, can be employed to ascertain the type of any *unknown* primitive of the system. (This contrasts with the von Neumann 29-state cellular automaton system where although any quiescent cell can be converted to any other cell type and thence back to a quiescent type, the particular transformation process adopted by von Neumann does not in all cases permit deduction of unknown cell-types.)

Chapter Four. Kinematic Machines and Their Behavior

4.1 Universal Computation

A kinematic machine consisting of two strings of primitives (a "program" string composed of F, B, PN, P0, P1, TN, T0, T1, PP, PM and H primitives and a "tape" string composed of N, 0, 1 primitives) can be designed to carry out any Turing machine computation. This follows from the fact that we can design a kinematic machine to carry out the computations of any Wang (1957) programmed Turing machine, a class of machines which Wang showed capable of carrying out any Turing computation.

Comment. The computational primitives of our system have been defined to exhibit the same computational properties as the Wang system basic program instructions. In the Wang system, the basic constituents are *instructions* which are to be implemented on a computer. In the kinematic system, each primitive element is itself a small machine, not merely an instruction, and must not only specify an operation, but must also *implement* the operation specified. There is some slight difficulty with the transfer operation. In the Wang system an unseen control and supervisory system of the assumed computer notes whether the transfer condition has been satisfied and if so, transfers control attention to the line of the program given by the transfer instruction. This instruction thus has a *non-local* effect; a change of state at the site of the transfer may in a single act, bring a change at a site far removed from the transfer. In the kinematic system, this complex global process has been implemented by a *series* of local actions of the simple primitive machines (see 2.3.5).

4.2 Machine Descriptions and Machines

Turing (1936) showed in his "universal simulation" result, that a machine could be supplied with a *description* of a second machine, and the first machine could then proceed to read the description and to simulate the action of the second machine. Indeed, the first machine could be supplied with a description of any of a large (universal) class of machines, a class including the simulator machine itself, so that, supplied with a description of *itself* the first machine could still, without paradox, carry out useful simulations of the action of the second machine. On this matter Minsky (1963) remarks that there are machines which can answer any question about themselves that any larger separate machine could answer. In a later paper (Minsky 1968) he again discusses mechanisms employing models of themselves and remarks (p. 430) "while it is impossible for a machine or mind to analyze from moment to moment what it is doing at each step (for it would never get past the first step) there seems to be no logical limitation to the possibility of a machine understanding its own basic principles of operation, or given enough memory, examining all the details of its operation in some previously recorded state". Thus, though there may be limitations to the practical use of machines which simulate themselves, there is no *prima facie* reason why machines cannot successfully make extensive use of their own descriptions.

4.3 Descriptions Self-Supplied

In the last section it was assumed that the descriptions upon which the machine worked, were supplied to the machine from the outside *ab initio*. Any fixed finite description which could be supplied to a machine from the outside, can be supplied to a machine by itself. In the kinematic system this can be done by augmenting the machine with a substring consisting of P1, P0, and F primitives which, when activated, will print out on the second (passive) string the desired description.

Example. Suppose we have an active string which has been initially supplied with a passive string 110 to work upon. The original active string itself could be augmented with the emitter sequence F-P1-F-P1-F-P0 which would, upon a "blank" string create the passive 110 string.

4.4 Self-Description Self-Supplied

Using the technique of 4.3 it is clear that given any active string, it can be augmented with an emitter string consisting of P0, P1 and F primitives only, which would create a 0,1 string of primitives describing the given active string. The active string could, then, proceed to read the string describing itself and to simulate what it would do under various alternative circumstances. Of course the description in the passive string contains no information on the description of the *augmented emitter* and to this extent machine deductions about itself could be in error. For example, the machine might attempt to establish how large it is, how many primitives it itself is composed of. By reading off the code blocks of its description, it would arrive at a certain number, and could print out this number. The machine however, if it is "blind" to the fact that it also contains an emitter of the original description, could be very much in error. This would be especially serious if the machine was attempting to reproduce itself on the basis of the printed description alone. If however, the machine incorporated in its structure the information that an initial description of itself was produced by a part of itself, *not* included in the explicit description, it might be able to deduce accurately the actual number of its primitives. For example, if its structure embodies the knowledge that the initial description it itself supplied was produced by a standard emitter sequence within itself, and that *two* emitter primitives (a P0 or P1 prefaced by an F) were necessary for every 0 or 1 primitive in the description, it could determine the number of code blocks in the original description, and add to this the number of 0 or 1 primitives multiplied by two to get the correct total number of primitives of which it itself consisted.

4.5 Complete Self-Description

Employing notions such as those suggested in 4.4, C. Y. Lee addressed himself to the question whether a machine could supply itself with a *complete* description of itself, and (Lee, 1963) proved that this was possible. Thatcher (1963) followed up on this by exhibiting a specific machine design which would accomplish this. We now show (largely following Thatcher's strategy) how a kinematic machine can obtain a complete description of itself. We will present several versions of this result, differing slightly in detail, in strategy, and in generality.

4.6 Complete Self-Description: Essential Process

Self-description of a kinematic machine will take place if beginning with an active string (either alone or with an all null passive string) the active string finally halts having created a separate passive string containing a complete correct description of itself in an agreed upon (uniquely decipherable) zero-one code of primitives. The self-describing machine will be an active program string composed of the following substrings: an emitter inferrer, and a (standard form) emitter of the description of the emitter inferrer. We begin by activating the emitter. At the conclusion of its operation, the newly created passive string will contain a description of the emitter inferrer. Activation is then shifted to the emitter inferrer. The inferrer examines the passive string and infers the description of the (standard) emitter which produced it, and prints out the description of the emitter. This completes the self-description.

4.7 Complete Self-Description with Arbitrary Substring

In 4.6 we presented the bare essentials of the complete self-description process. The only function of the self-describing machine of 4.6 is self-description. In general however we shall be interested in machines which in addition to the capacity to obtain their own complete descriptions are capable of carrying out some other processes of interest (such as for example reading their own complete descriptions, and from this establishing structural and computational properties). Such a self-describing machine might consist of the following substrings: any desired finitely long arbitrary sub-string (embodying functions of interest), an emitter inferrer, and an emitter of the description of the arbitrary optional sub-string and of the emitter inferrer. We begin by activating the emitter. At the conclusion of its operation, the passive string will contain a description of the optional substring and a description of the emitter inferrer. Activation is then shifted to the emitter inferrer. The inferrer will examine the passive string and infer the description of the emitter which produced it, and print out the description of the emitter. This completes the self-description. Although this self-description procedure is deficient in some details the essential features of the process are apparent. (See Figure 5 for an illustration of this self-description process.)

Note. In Figs. 5 - 11 we attempt to illustrate how the routines and submachines we have designed (in Chapter Three) can be conjoined to produce the desired self-description and self-reproduction processes. In the figures, the following conventions are employed. The conceptually distinct basic routines and sub-machines are enclosed in parentheses and connected by plus signs. The present active site in the machine is marked by a star, and the passive site being acted upon is marked by an arrow. In the absence of an explicit transfer, activation proceeds automatically from left to right; transfers of activation *within* basic routines or sub-machines are not explicitly pictured; transfers of activation *between* basic routines and submachines or between strings are, in some of the figures, made explicit. The vertical bar, |, separates the initial string from the (sole) other string which may be created. For typographical convenience, the submachines and routines are in some

Figure 5. Self-Description Process. (i). The initial machine consists of an emitter, an inferrer, and an arbitrary sub-machine; initial activation site is at the head of the emitter. (ii). The emitter has stored in it a description of the inferrer and the arbitrary sub-machine; at the conclusion of emitter action the second string consists of a 0-1 description of the inferrer and the arbitrary string. (iii). Activation proceeds to the inferrer, which examines the second string and infers the composition of the standard emitter routine which produced the string, and itself produces a description of the emitter routine. Activation now proceeds to the arbitrary sub-machine which is free to make use of the description information.

(i).

$$*\underset{A}{\text{(Emitter)}} + \underset{B}{\text{(Inferrer)}} + \underset{C}{\text{(Arbitrary)}} \quad |^{\downarrow}$$

(ii).

$$\underset{A}{\text{(Emitter)*}} + \underset{B}{\text{(Inferrer)}} + \underset{C}{\text{(Arbitrary)}} \quad | \quad \text{(Description of (B) + (C))}^{\downarrow}$$

(iii).

$$\underset{A}{\text{(Emitter)}} + \underset{B}{\text{(Inferrer)*}} + \underset{C}{\text{(Arbitrary)}} \quad | \quad \text{(Description of (B) + (C))} + \text{(Desc of (A))}^{\downarrow}$$

Figure 5

cases abbreviated by capital letters.

4.8 Complete Self-Description with Explicit Transfer Deduction

In 4.6 and 4.7 we have described the simple essential process of complete self-description. In the process, the machine has employed first its emitter, and then its emitter inferrer. To go from one sub-string region to another in an active machine requires the use of a transfer. Thus a transfer operation *and* its description must be incorporated into the self-description process. We now show how this can be accomplished (see also Fig. 6).

The complete self-describing machine with explicit transfer deduction will be an active program string composed of four sub-strings. At the top of the active string will be an *emitter inferrer,* followed by a *transfer inferrer* then by an *emitter* of the description of the first two substrings (the *emitter inferrer* and the *transfer inferrer*), followed by an (unconditional) *transfer primitive* (and its associated PM string, assuming our machine employs the method of positive or negative shifts for transfer implementation) which will implement a transfer to the top of the program (the first primitive of the *emitter inferrer*). We begin by activating the third substring, the *emitter.* At the conclusion of its operation, the passive string will contain a description of the first two substrings (the *emitter inferrer* and the *transfer inferrer*) of the active string. Activation is then (by means of the fourth substring, the *transfer primitive*) shifted to the *emitter inferrer* (at the top of the program string). The *inferrer* will examine the passive string and infer the description of the *emitter* which produced it, and print out the description of the *emitter.*

Figure 6. Complete Self-Description with Explicit Transfer Deduction. (i). Initial situation, with activation at emitter. (ii). Emitter action produces description of Inferrer and Transfer inferrer. (iii). Transfer operates to transmit activation to inferrer. (iv). Inferrer examines the existing description and produces the description of the standard emitter which produced it; activation proceeds to the transfer inferrer. (v). Transfer inferrer produces a description of the transfer primitive and its associated shift primitives, completing self-description. Note: An arbitrary sub-machine could have been inserted between the transfer inferrer and the emitter (and the emitter equipped with the capacity to print out the description of the arbitrary machine); activation would then pass to the arbitrary machine, which could make use of the complete description now available.

```
(i).

(Inferrer) + (Transfer Inferrer) + *(Emitter) + (Transfer)  |↓
    A                B                  C            D


(ii).

(Inferrer) + (Transfer Inferrer) + (Emitter) + *(Transfer)  |
    A                B                C             D

                                               (Desc (A) + (B))↓
                                                      E


(iii).

*(Inferrer) + (Transfer Inferrer) + (Emitter) + (Transfer)  |
     A                B                C             D

                                              ↓(Desc (A) + (B))
                                                      E


(iv).

(Inferrer) + *(Transfer Inferrer) + (Emitter) + (Transfer)  |
    A                 B                C             D

                                    (Desc (A) + (B)) + (Desc (C))
                                                           F


(v).

(Inferrer) + (Transfer Inferrer)* + (Emitter) + (Transfer)  |
    A                B                 C             D

                           Desc((A) + (B)) + (Desc (C)) + (Desc (D))↓
                                                 F            G
```

Figure 6

The *transfer inferrer* (next in line) now takes over and infers and describes the *unconditional transfer* and the requisite number of PM primitives necessary to shift activation to the top of the program (see 3.14). This completes the complete self-description with explicit transfer deduction.

4.9 Complete Self-Description: Active Machine in Loop Form

In 4.8 we pointed out that detailed explication of the self-description process obliges us to consider the transfers between sub-routines and how these are to be described. The particular issue raised there (the presence and consequent inference and description of the unconditional transfer from the end of the emitter routine to the beginning of the inference routine) can be shown to be somewhat of an artefact of the particular kinematic machine system employed (in particular, the mode of transfer implementation employed). Consider a self-describing machine consisting of an *inferrer* and an *emitter* of the description of the inferrer, in which the last primitiv of the *emitter* is connected directly to the first primitive of the inferrer (thus, by connecting the active machine in a *loop* we obviate the requirement for an explicit separate unconditional transfer from the end to the beginning of the active string). We begin by activating the *emitter*, which prints out a description of the rest of the string (the inferrer). The activation now (owing to the loop structure of the active string) proceeds "automatically" to the inference routine, which examines the passive description and deduces and prints out the description of the *emitter*. This completes self-description.

4.10 Construction of Machines from Descriptions

Any of the primitives of which our kinematic machines are composed can be obtained by means of a constructor acting upon an initial recruited N primitive. The action of the constructor can be compounded so that any *sequence* of primitives can be constructed. Thus, a machine equipped with an active constructor string can, given a (passive) description string for any machine (in zero-one primitive coded form) read the description code-word by code-word, and then, moving to the end of the description string and recruiting an N primitive, construct a copy of each described primitive in turn. After having constructed a copy of the machine described the constructing machine can activate the new machine (after possibly detaching the new machine from its description, or destroying the description entirely). It should be explicitly pointed out that the description given the constructing machine could be a description of the constructor itself; thus we see that a machine can construct a copy of itself. (Note that this is not precisely reproduction, since, beginning with two strings, one an active machine and the other a passive description, we ended with two machines (and possibly the original description): for complete reproduction we should, beginning with a machine and a description, end with *two* machine-description pairs.)

4.11 Self-Reproduction by Means of Description

We now show how complete self-reproduction in machines can take place. Since by the Lee (1963) and Thatcher (1963) self-describing results there exist kinematic machines which can produce their *own* complete descriptions, and since given any description a kinematic machine exists which can construct the described machine, a machine consisting of a self-describer and a general constructor provides the basis for self-reproduction (Thatcher 1970). More precisely (see Figure 7) our self-reproducing machine will initially consist of (A) an inferrer, (B) a constructor, (C) a destroyer, (D) emitter of description of (A)(B)(C). We begin by activating (D) the emitter of the description of (A)(B)(C). At the conclusion of (D)'s action, we will have an active string consisting of the original machine (A)(B)(C)(D) and a new passive string consisting of the description of the (A)(B)(C) parts of the original string. Activation now passes to (A) the emitter inferrer, which reads the description in the second string and infers (and describes) the structure of (D). At this point we have two strings, one the original active machine and the other the complete (passive) description of the original machine. Activation now passes to (B) the constructor. (B) reads the passive description and constructs a copy of the machine there described. Finally, activation passes to (C) the destroyer, and (C) destroys the description so that we are left with the original machine, and its copy, completing self-reproduction.

Comment. Essentially, the result exhibited in 4.11 is a variant of the original von Neumann (1966) result that an automaton can reproduce itself if it is supplied with a *description* of itself (alternatively, if it is supplied with explicit *instructions* embodying a description for the construction of itself). It is thus an automaton model of the logic of biological reproduction as it is believed actually to take place in living organisms, viz., by means of cell enzymatic protein machinery acting upon a prior supplied nucleic acid

Figure 7. Self-reproduction Utilizing Prior Description. (i). Initial machine consisting of an inferrer of the standard emitter of a description, a general-purpose constructor (ie. a constructor which given the description of a machine will construct the machine), a destroyer (which will reduce a description to N primitives), and an emitter of the description of the inferrer, the constructor, and the destroyer. (ii). The emitter produces the description D of the inferrer, constructor and destroyer. (iii). Activation is passed to the inferrer. (iv). Inferrer produces the description of the emitter, completing the description. (v). Constructor employs the description to produce a duplicate of the original machine. (vi). Destroyer reduces the description; final situation consists of two structurally identical machines.

(i).

$\underset{A}{(\text{Inferrer})} + \underset{B}{(\text{Constructor})} + \underset{C}{(\text{Destroyer})} + *\underset{D}{(\text{Emitter})} \quad |^{\downarrow}$

(ii).

$(A) + (B) + (C) + (D)* \quad | \quad \mathcal{D}((A) + (B) + (C))^{\downarrow}$

(iii).

$*(A) + (B) + (C) + (D) \quad | \quad {}^{\downarrow}\mathcal{D}((A) + (B) + (C))$

(iv).

$(A)* + (B) + (C) + (D) \quad | \quad \mathcal{D}((A) + (B) + (C)) + \mathcal{D}(D)^{\downarrow}$

(v).

$(A) + (B)* + (C) + (D) \quad | \quad \mathcal{D}((A) + (B) + (C)) + \mathcal{D}(D) + (A') + (B') +$
$(C') + (D')^{\downarrow}$

(vi).

$(A) + (B) + (C)* + (D) \quad | \quad {}^{\downarrow}(A') + (B') + (C') + (D')$

Figure 7. Self-reproduction Utilizing Prior Description

description of the cell.

4.12 Another Example of Reproduction by Means of a Description

In close analogy to the logical form of the natural reproduction process, we might begin with a machine consisting of an active string and a separate description string. The active string contains a *constructor* (that is, a routine which given a description produces the string described) and a *copier* (which when given a description makes a copy of it). The active string, acting upon its supplied description, produces a duplicate of itself and a copy of its description, thus completing self-reproduction (beginning with a machine-description pair, we end with *two* machine-description pairs). (Note that this form of reproduction requires the active machine to direct its attention systematically to a total of *three* other strings: its description, its duplicate, and a copy of its description. Although the capacity to switch the attention of an active string to *several* passive strings could be made an explicit feature of our kinematic system, we have, at this point, not done so. Note the resemblance of such a multi-string system to multi-tape Turing machines.)

4.13 Another Variant: Reproduction without Use of Temporary Description

In the reproduction process described in 4.11 we employed the notion of a machine first producing for its perusal its complete description, employing the description to construct a duplicate of itself, then destroying the superfluous description. The reproduction process can be modified to obviate the need for first producing and then later destroying a description.

Our initial machine is a string consisting of (A) an Analyzer, (B) Constructor Inferrer, (C) General-purpose Constructor, (D) Special-purpose Constructor of (A), (B) and (C). The reproduction process begins with the activation of the Special-purpose Constructor of (A), (B) and (C). Activation is then transferred to (A) the Analyzer. The Analyzer determines each of the primitives of the newly constructed string in turn. As each is discovered, activation is transferred to (B) the Constructor Inferrer, which determines the primitives of the Special-purpose Constructor which could have produced the new string. For each primitive determined, activation is transferred to (C), the General-purpose Constructor, which constructs the determined primitive. Continuing thus, the sequence of primitives of (D) is constructed, and we have reproduced the initial machine without use of a temporary description.

4.14 Complete Self-Inspection: Background

In all the cases so far considered, a machine has made use of a description which has explicitly or implicitly, been made available to it. The question has been raised whether a machnne can by self-inspection acquire its own complete description. It is often remarked that no system can comprehend itself completely (since for example, its organ of comprehension is a part of itself, and the part cannot comprehend the whole), or examine itself completely (since, the organ which examines cannot examine itself). Burks (1961) however has conjectured that it *is* possible for an automaton to sense its own complete constituent structure and construct and store within a proper part of itself information which can make the complete self-description available to the machine for its own perusal. We now show how this can indeed be done.

4.15 Self-Description by Self-Inspection (Principal Result)

We now exhibit a machine which can obtain its own complete constituent structure description by means of self-inspection. The machine proper will begin by consisting of a single string (it will later temporarily consist of two strings). This single string will consist of the following machine substrings (listed in the order of their employment).

(A) A special purpose emitter-constructor which can construct a new temporary separate string, a string consisting of 1) an *analyzer* and 2) a *constructor* which will take as input a description of any string and will produce an emitter of the description of the string and 3) a *transfer constructor*, and 4) a *locator* and *activator routine* which will relinquish activation to (B).

(B) A general or a special-purpose *destroyer* which can convert the separate string constructed by (A) back to null primitives.

(C) A two-part *inference routine* which applied to a description first infers the string which emitted the description and second, by means of a transfer inferrer, appends at the end of the description a description of a transfer to the top of the program.

We may also optionally include:

(D) A general processor substring capable of carrying out some useful computation. This substring takes no active part in the self-analyzing and describing processes.

The creation of a complete self-description proceeds as follows. (See Figure 8.) The machine begins by constructing the new second string (consisting of an analyzer, a constructor of an emitter, a transfer constructor, and a locator). At the conclusion of this construction, the original machine, by means of an A primitive, relinquishes activation to the new string (at the *head* of the created string, the analyzer). The new string now proceeds to analyze the original machine and to construct at the end of the original machine, an emitter of the description of the original machine. At the end of this emitter, the transfer constructor present in the new string constructs a transfer to the inference routine at the top of the original string.

When the new string has completed this task, it employs an A primitive to relinquish activation back to the original machine at the head of the destroyer routine (which can be at the head of the original string or separately addressed). The original machine now employs its destruction sub-program to destroy the second string.

The existing machine now has a complete description (residing in the newly appended emitter) of what it was initially. Also note we have in effect re-created the organization of the Lee-Thatcher self-describing Turing machine (see 4.8 and Fig. 6). That is, the emitter can be activated to provide the description of all of the present machine save the emitter and the final transfer (that is, the *complete description* of the *original* machine). In addition (employing the inference routine with its transfer inferrer), the present machine can provide itself with a description of the appended emitter, and

Figure 8. (i). Initial Situation. The initial machine consists of a single string possessing the following sub-routines: (A), a special-purpose constructor (of a single second string which will have the following sub-routines (1) an analyzer, (2) a general-purpose constructor of an emitter, (3) a transfer constructor, (4) a locator and activator of the address of (B), the destroyer); (B), a destroyer (which includes a transfer to the end of (D)); (C), an inference routine; (D), an optional arbitrary routine of any fixed finite length. (ii). Special Purpose Constructor Action. (A), the special-purpose constructor is initially activated. (A) produces a single second string possessing the following four sub-routines: (E) an analyzer, (F) a general-purpose constructor of an emitter, (G) a transfer constructor, (H) a locator and activator of address of (B). At the conclusion of construction (A) will relinquish activation to (E) the analyzer in the new second string. (iii). The Action of the Second String. The newly activated second string begins with (E) the analyzer. The analyzer moves to the top of the first string, and determines the type of the first primitive. Activation is then transferred to (F) the constructor of an emitter. (F), bearing the information as to the type of the first primitive of the initial string, constructs at the end the routine which when activated will emit the description of the identified first primitive. Activation is then transferred back to (F), and the analysis process continues. When all of the original string has been inspected and the emitter of the description of (A)(B)(C)(D) produced, activation is passed on to (G) the transfer constructor. (iv). The Transfer is Constructed. The transfer constructor (G) constructs an unconditional transfer primitive and an associated number of PM primitives required to transfer to the top of the original program. It does this by examining all of the initial string, determining the shift length required, and constructing the required number of PM primitives. Activation is then transferred to (H) the locator and activator of (B). (v). Location and Activation of Destroyer. The locator and activator routine (H) relinquishes activation to (B), the destroyer routine of the original string. (vi). Destruction of Second String. (B) reduces all the primitives of the second string to N (and thus disperses them). Afterward (B) will transfer activation to the end of (D), (which is the beginning of I). (vii). Self-describing Machine. Resulting single string machine now has (in its essentials) the form of the self-describing machine exhibited in Fig. 6. (That is, (I) will produce a description of all but itself and (J). (C) the inference routine will deduce the description of I and add the description of J, completing self-description.)

(i).

$$*\underset{A}{\text{(Special-purpose Constructor)}} + \underset{B}{\text{(Destroyer)}} + \underset{C}{\text{(Inferrer)}} +$$

$$\underset{D}{\text{(Arbitrary)}} \;|^{\downarrow}$$

(ii).

$$\underset{A}{\text{(Special-purpose Constructor)}}* + \text{(B)} + \text{(C)} + \text{(D)} \;|\; \underset{E}{\text{(Analyzer)}} +$$

$$\underset{F}{\text{(Constructor of Emitter)}} + \underset{G}{\text{(Transfer Constructor)}} +$$

$$\underset{H}{\text{(Locator and Activator of (B))}}^{\downarrow}$$

(iii).

$$\text{(A)} + \text{(B)} + \text{(C)} + \text{(D)} + \underset{I}{\text{(Emitter of }\mathcal{D}\text{((A) + (B) + (C) + (D))}}^{\downarrow} \;|$$

$$\underset{E}{\text{(Analyzer)}} + \underset{F}{\text{(Constructor of Emitter)*}} + \underset{G}{\text{(Transfer Constructor)}} +$$

$$\underset{H}{\text{(Locator and Activator of (B))}}$$

(iv).

$$\text{(A)} + \text{(B)} + \text{(C)} + \text{(D)} + \text{(I)} + \underset{J}{\text{(Transfer to (A))}}^{\downarrow} \;|\; \text{(E)} + \text{(F)} +$$

$$\underset{G}{\text{(Transfer Constructor)*}} + \underset{H}{\text{(Locator and Activator of (B))}}$$

Figure 8

(v).

$$(A) + {}^{\downarrow}\underset{B}{\text{(Destroyer)}} + (C) + (D) + I + J \;\big|\; (E) + (F) + (G) +$$

$$\underset{H}{\text{(Locator and Activator of B)*}}$$

(vi).

$$(A) + \underset{B}{\text{(Destroyer)*}} + (C) + (D) + (I) + (J) \;\big|^{\downarrow}$$

(vii).

$$\underset{A}{\text{(Special-purpose Constructor)}} + \underset{B}{\text{(Destroyer)}} + \underset{C}{\text{(Inferrer)}} + \underset{D}{\text{(Arbitrary)}} +$$

$$*\underset{I}{\text{(Emitter)}} + \underset{J}{\text{(Transfer)}} \;\big|^{\downarrow}$$

final transfer, thus making available to the present machine its *present complete* description.

Thus we have exhibited a machine which has discovered its own structure completely, contains this description in a proper part of its present self, and can make this description available to itself for its own perusal and calculation, confirming the Burks conjecture mentioned in 4.14. (For a critique of this self-inspection process see 5.1.)

Comment. Some remarks on the implementation of transfers of activation which are required may be useful. We are most concerned with relinquishing activation from one string to another, and with transfer of activation from one sub-routine to another non-contiguous routine (not to transfers within sub-routines or to next neighbor routines).

The special-purpose constructor (A) must be equipped with a sub-routine which implements a relinquishing of activation to the new analysis routine it has constructed. When the second string has completed its construction actions upon the original string, it must relinquish activation to (B) the destroyer of the original string. For this, the second string must be equipped (by (A) the special-purpose constructor which produced it) with a routine which locates the address of (B) and activates (B). When (B) completes its destruction and dispersal of the second string, activation is transferred to the end of the original program (ie. to the head of (I) the newly created emitter). Upon the conclusion of (I)'s action, a transfer is made to the top of the program. At the top (just before the beginning of (A)'s construction routine) there is a transfer to (C) the inference routine. At the conclusion of the inference (with its completion of the description of the machine), activation is allowed to pass to (D) the arbitrary routine, which is free to make use of the now available complete structural description.

4.16 A Minor Variant

In the self-description by self-inspection result just presented we had the analyzer of the newly constructed second string inspect the very constructor which produced the analyzer. This is not strictly necessary, providing that the original string is properly augmented with a special sort of inferrer which can analyze the second string and infer the special-purpose constructor which could have produced it. Thus the final self-describing machine would have to read the machine part of the second string to infer the special-purpose constructor of the first string, then emit the description of the arbitrary parts, and infer the emitter of the arbitrary part, thus making a complete description available.

4.17 Self-Inspection: Improved Version

Although the machine of 4.15 satisfies the conditions of the Burks conjecture on self-description by inspection, the description (the new emitter) though a proper part of the new total machine, nevertheless constitutes a large part of the machine: in standard form it is several times the size of the non-description part of the machine. We therefore now describe a kinematic system in which a machine obtains a description of itself, and in which the description need be no larger than the non-description part of the machine.

We begin with an initial string which first produces and then activates a separate analyzer string equipped with a locator subroutine. The new analyzer string is to be used to discover the primitives of its "parent" the original string, one by one, and to disclose them to the original string. It does this by the following process. After analyzing and ascertaining the type of a primitive of the first string, the second string transfers to its locator routine and moves along the first string to a region which, when activated, constructs a duplicate of the newly identified primitive type at the end of the second string. Activation is then transferred back to the second string, and the second primitive of the first string is read. Continuing thus, a complete duplicate of the original string can be constructed and appended to the second string. At the conclusion of the creation of the duplicate of the original string, the first string can destroy the analyzer and locator portion of the second string, leaving only the duplicate of itself. Thus we now have two copies of the original string, one active, and one passively available to be read (by an analyzer contained in the original string).

If the original string requires a description of itself as it was when the self-description process began, it can analyze the passive string to discover this. If it wishes a description of itself as it now is (consisting of two copies of itself) it can obtain this by examining its passive copy *twice* (since it now exists in the form of two identical strings) or, active and passive roles can be exchanged between the two copies, so that each can examine the other in turn.

The "description" we have now produced is precisely the length of the non-description part of the machine. Can it be reduced further? Clearly if large sections of the original machine possess a very regular structure, it may be possible to describe these sections using fewer primitives than the corresponding part of the original machine. Two questions remain however. 1) Can the machine itself examine a copy of itself, detect regularities, and produce a more compact description? 2) If the description can be so reduced cannot the machine itself be correspondingly reduced; is there a *shortest* machine, and can its description be any shorter than itself?

It will be seen that at this point our considerations begin to join with those of Chaitin (1974). "Small" or smallest machines needs must possess a "random" structure, since there can be nothing redundant (patterned, regular, expected) about such machines (else they could be made even more compact). Since most machines of a given length are random, for most machines their minimum description is a duplicate copy of themselves, a copy obtainable by our strategy.

4.18 A Minimal Self-Inspecting System

The minimal self-inspecting system would perhaps consist initially of two strings, each an analyzer. The first analyzer analyzes the second, the second the first. We will of course have to augment the analyzers with the capacity to relinquish activation to the other. An even more serious deficiency is the fact that such a system does *nothing but* inspect itself (it does not even produce a distinct description of itself; it merely ascertains its own structure, primitive by primitive, and moves on). We can of course add substrings embodying "interesting" functions to either or both of the original strings.

4.19 An "Almost Successful" Strategy of Complete Self-Inspection

Let us consider the extent to which the general Lee-Thatcher strategy of self-description might be successful in producing a self-description by self-inspection. The system will consist initially of two strings. The first string will consist of an arbitrary machine and an emitter of a description of an analyzer. The second string consists of an analyzer alone. The original activation is in the separate analyzer, and we imagine (for concreteness) that the description is to appear on a third string, accessible to both of the original two. The analyzer examines the first string, and produces its description. It cannot produce its own description though. This deficiency is supplied by activating the emitter of the first string which prints out its stored description of the analyzer. Thus an arbitrarily large part of a machine can be self-inspected, leaving only an analyzer-sized component inaccessible to inspection, the description of which can however be "pre-packaged" to complete the self-description.

Comment. It may be useful to set down some of the ways in which systems can inspect themselves (or other machines). First, a system may be equipped with sensory organs making it possible for the system to detect or inspect directly certain sectors or components of the system. The detectable parts of the system will in general *not* include all of the system: parts of the system may be blind to direct, "at a glance" detection. This will generally be the case with the inspecting system itself, for it will require *its* inspector, and this inspector, yet another inspector, so that a regress may be produced. We see that first of all, the system, to inspect all of itself, should have the means to detect *all* of its constituent parts. The methods for this which have been suggested are analysis, and "signature" reading.

Once the issue of being able to read *all* the constituents is resolved, the problem of a machine having *access* to all its constituents must be considered. One aspect of this is spatial. If a system has interior or inaccessible regions then the system itself (or another inspecting system) may not be able to bring its detector units to bear on every constituent. This is especially critical if the machine or system under inspection is *active*, since the "radical surgery" necessary to attain access to interior constituents may disrupt or destroy important system activities. In the kinematic system, this is overcome by making all the components *strings* in which the constituents are always directly exposed. Use of the von Neumann two-dimensional cellular space system may require intrusion into a block of cells and consequent alteration of peripheral cells in the inspection of the interior. If however the inspected system is *passive* then an inspection and repair routine in an appropriately designed two-dimensional cell-space is possible. (Also we have pointed out that a two-dimensional machine may employ a third dimension in its analysis process obviating the violation of machine structure.) Finally, the *behavior* of the inspector machine must be *complete and thorough*. That is, the inspecting system, whether its actions are directed toward itself or another system, must behave so as to read *all* and *only* the subject system. Correct such behavior may require an intricate schedule of transfers and actions, so that *all* parts are inspected, and no inspection duplication is inadvertently carried out.

We thus see that among our possibilities are the following. Systems can behaviorally inspect *all* regions of a system, or only some; systems can detect all constituent types of a system or only a sub-set of them; systems can have access to all regions of systems or only some of them; systems can inspect only passive systems (or regions) or may inspect active systems (or regions).

This last point, the capacity of a system to inspect another *active* system, remains unresolved.

4.20 Reproduction Without Description: Background

John von Neumann's successful strategy of machine self-reproduction (von Neumann, 1966) makes use of an active machine reading, and then employing information obtained from a separate passive "blueprint" *description.* (Very roughly speaking we can equate the active components of von Neumann's machine with enzymatic proteins of the cell, and the separate passive description with nucleic acids.) At the same time, von Neumann (1966, p. 122) was aware of alternative possible reproductive strategies, including the idea that a machine might "read" itself directly and act upon the information thereby disclosed, to reproduce itself. (In the cell this would mean that the active protein machinery might somehow analyze itself and use this information to reproduce, only incidentally if ever being in possession of a nucleic acid description of the protein machinery.)

He rejected this strategy however since there were evident practical difficulties of automaton implementation, and he also feared there might even be an essential logical paradox inherent in the notion of a system actively inspecting its active self.

Arbib (1966, p. 217) however was somewhat more sanguine, stating that he was not convinced that there is any logical paradox in a machine examining itself and thereby obtaining a description it can use for carrying out complete self-reproduction. Indeed, Burks (1961) had already conjectured that it is possible for a machine to inspect all of itself and obtain a complete description of itself which it can store for its use within a proper part of itself. We have shown the Burks conjecture to be true; there *is* a strategy by which a machine can examine itself to obtain for its use a complete description of itself.

In the next section, this result is employed to show that self-reproduction by means of self-inspection is possible. In effect, there is no *logical* necessity for the presence of an explicit blueprint in the reproduction of completely general information transactional systems. Also, since the complete "mature" original parent machine serves as the model for the offspring machine, we perforce have a logically consistent strategy of reproduction in which acquired characteristics of parent can be transmitted to the offspring.

4.21 Reproduction by Self-Inspection

We are now prepared to show how a machine can reproduce itself, using itself as model. We begin with an initial single string "parent" machine consisting of the following substrings.

(A) A special-purpose *constructor* which can construct a new *second* string. This second string which (A) constructs will consist of (E) an *analyzer* and *restorer*, (F) an *inference routine* of the sort which can take a description and infer the description of the emitter which could produce the given description, (G) a general-purpose constructor, (H) a general-purpose destroyer.

(B) A *destroyer* which can convert strings back to null primitives.

(C) A fixed finitely long *optional substring*, capable of carrying out some desired general behavior; this substring plays no active role in the self-analyzing and self-reproducing process.

(D) A *general-purpose constructor* which takes the description of any string and constructs the string.

The process of reproduction can be carried out as follows. (See Figure 9.)

(i) (A) is activated and constructs the new second string consisting of (E) (analyzer-restorer), (F) (emitter inference routine), G) (general-purpose constructor), and (H) (destroyer).

(ii) Activation is relinquished to the new string and the new string analyzes the original string, and constructs at the end of the original machine a new substring (I) which is an emitter of the description of (A),(B),(C), and (D).

(iii) The new string now relinquishes activation to (E) of the first string. (E) then constructs, as part of the second string, (J), the description of (A)(B)(C)(D).

The first string now consists of (A)(B)(C)(D)(I) and the second of (E),(F),(G),(H), and (J), the description of (A)(B)(C)(D).

(iv) Activation is now relinquished to (H) the destroyer, of the second string. (H) then reduces (I) back to null primitives.

(v) Activation is now relinquished back to (B) of the original string which destroys all but (J) the description of the second string.

(vi) Activation is transferred to (D), the constructor of the first string, which using the description (J), of (A),(B),(C),(D) constructs a copy (A)',(B)',(C)',(D)'.

(vii) Activation in the first string is now passed to the destroyer (B), which reduces (J) to null primitives, leaving the second string consisting of the copy of (A)(B)(C)(D) only. This copy can now be activated and released by application of an AD primitive.

We thus have reproduced our original machine, by an examination of its structure. Whatever properties the original possessed at the time of reproduction are now recreated in the offspring. Roughly speaking, we have here a model of reproduction in which the use of a distinct description is not central to the process and in which any acquired characteristics of the original parent string would be reproduced in the offspring string.

Many slight variations of this reproduction by self-inspection can be devised; the sequence of the "clean-up" activities can, for example, be altered. In the next sections we exhibit some more radical variants of the process.

4.22 Comment on Concealed Descriptions in Self-Inspection

In both the self-description by self-inspection and self-

Figure 9. Reproduction by Self-Inspection. (i). Initial situation. (A) Special-Purpose Constructor, (B) Destroyer, (C) Optional Substring, (D) General-Purpose Constructor. (ii). (A) constructs (E) Analyzer, (F) Inferrer, (G) General-Purpose Constructor, (H) Destroyer. (iii). Analyzer (E) identifies the primitives of (A),(B),(C),(D), and Inferrer (F) uses this information to instruct General-Purpose Constructor (G) to produce (I), the Emitter of a description of (A), (B),(C),(D). (iv). The Emitter (I) produces the description (J) of (A),(B),(C),(D). (v). Destroyer (H) removes the Emitter (I). (vi). Destroyer (B) removes (E),(F),(G),(H). (vii). Constructor (D), using description (J), produces (A)',(B)',(C)',(D)'. (viii). Destroyer (B) removes description (J); two structurally identical copies of the original machine remain.

Figure 9. Reproduction by Self-Inspection

(i).

$$*\underset{A}{(\text{Special-purpose Constructor})} + \underset{B}{(\text{Destroyer})} + \underset{C}{(\text{Arbitrary})} + \underset{D}{(\text{General-purpose Constructor})} \;|^{\downarrow}$$

(ii).

$$(\text{Special-purpose Constructor})^{*} + (B) + (C) + (D) \;|\; \underset{E}{(\text{Analyzer})} + \underset{F}{(\text{Inferrer})} + \underset{G}{(\text{General-purpose Constructor})} + \underset{H}{(\text{Destroyer})}^{\downarrow}$$

(iii).

$$(A) + (B) + (C) + (D) + \underset{I}{(\text{Emitter of } \mathcal{D}((A) + (B) + (C) + (D))}^{\downarrow} \;|\; (E) + (F) + (G)^{*} + (H)$$

(iv).

$$(A) + (B) + (C) + (D) + \underset{I}{(\text{Emitter of } \mathcal{D}((A) + (B) + (C) + (D))}^{*} \;|\; (E) + (F) + (G) + (H) + \underset{J}{\mathcal{D}((A) + (B) + (C) + (D))}^{\downarrow}$$

(v).

$$(A) + (B) + (C) + (D)^{\downarrow} \;|\; (E) + (F) + (G) + \underset{H}{(\text{Destroyer})}^{*} + \underset{J}{\mathcal{D}((A) + (B) + (C) + (D))}$$

(vi).

$$(A) + \underset{B}{(\text{Destroyer})^*} + (D) \;\;|^{\downarrow}\;\; \underset{J}{\mathcal{D}((A) + (B) + (C) + (D))}$$

(vii).

$$(A) + (B) + (C) + \underset{D}{(\text{General-purpose Constructor})^*} \;\;|\;\; (J) + (A)' +$$

$$(B)' + (C)' + (D)'^{\downarrow}$$

(viii).

$$(A) + \underset{B}{(\text{Destroyer})^*} + (C) + (D) \;\;|\;\; {}^{\downarrow}(A)' + (B)' + (C)' + (D)'$$

reproduction by self-inspection results, we can achieve very "small" systems exhibiting the desired properties. In the self-description case the system consists of a pair of analyzer strings, and in the self-reproduction case, a pair of analyzer-constructor strings. It had long been believed that for self-description and self-reproduction some part of the system must essentially be a *description* of the remaining part of the system. In this self-inspection case, it might seem that indeed this is true: that each analyzer (analyzer-constructor) string serves as the description of the other string. This is not in fact the case. For example, each string of the pair can differ structurally, as long as the proper *functions* are carried out; when one analyzer reads the other analyzer, it will *not* be obtaining a description of itself, for the two analyzers need not be identical. The true nature of the relationship is even more striking when we consider that either or both of the strings could be augmented with arbitrary additional sub-strings, the sub-strings not necessarily having any structural or functional relationships with each other.

In Laing (1975) it had been conjectured that in any system capable of complete, general self-reproduction there would always be found "paired organs", such as machine and its description (though perhaps in very cryptic form). That the relation between the machine and its description might be quite obscure was evident. For example, the machine could be in the form of a special-purpose constructor of a machine, and the description in the form of a series of instructions, or of an emitter of the description (or instructions), or the special-purpose constructor of an emitter of the description (or instructions), etc. Systems consisting of two such components could reproduce if the machine or active component could act upon the description (or instructions) in a *dual* fashion: first it used the description (or instructions) to make a copy of the description (or instructions) and second it used the description (or instructions) to construct what was there described (or obeyed the instructions to the same end). The self-inspection results show that such "paired organs" are *not* required in the strong sense originally intended: there need not be a machine and its complete description present initially. Our result says that only an *analyzer-constructor* need be in "paired" form, the rest can be arbitrary.

4.23 An Improved Result

Although we have just shown that a machine can achieve a complete reproduction of itself by employing itself as model, the strategy is not entirely satisfactory, requiring as it does the creation and later destruction of whole substrings (in particular the creation and destruction of both a description of the initial machine and an emitter of this description).

We now show how the self-reproduction by self-inspection can be simplified. (See Figure 10.) In particular, we re-design our system so that the information acquired by the new analysis string need not be temporarily stored in the description and emitter of a description, but is acted upon as it is acquired. In this strategy, only the analyzer of the second string will prove eventually to be redundant and thus condemned to dissolution in the final "clean-up".

We begin with an initial string which first produces and then activates a separate analyzer string equipped with a locator subroutine. The new analyzer string is to be used to discover the primitives of its "parent" the original string, one by one, and to disclose them to the original string. It does this by the following process. After analyzing and ascertaining the type of a primitive of the first string, the second string switches to its locator and moves along the first string to a region which, if activated, will construct a copy of the newly identified primitive type. In effect, the first string will contain regions for constructing each of the primitive types, and the second string will locate the proper constructor region and transfer activation to it. The now active first string can construct and append to a reserved part of the second string a copy of the primitive named. Activation is then transferred back to the second string, and the second primitive of the first string is read. Continuing thus, a complete copy of the original string can be constructed and appended to the second string. At the conclusion of the creation of the second copy of the original string, the first string can destroy the originally created locator portion of the second string, leaving only the copy of itself. This copy can be activated and dispersed, completing the reproduction by means of self-inspection.

Here our model effectively dispenses entirely with even the temporary use of a separate description.

4.24 Simplified Reproduction by Means of Self-Inspection

Although by the construction of the last section, the self-

Figure 10. Temporary Description Dispensed With. (i). Initial situation. Initial machine consists of (A), a special-purpose constructor of an analyzer and a locator, (B), a constructor, (C), a destroyer, and (D), an arbitrary substring. (ii). (A) constructs the second string consisting of (E) analyzer and (F) locator. (iii). Analyzer (E) inspects original machine, primitive-by-primitive, and communicates identities of primitives to locator (F) which activates appropriate constructor region within (B). (iv). Constructor (B) produces copies of the newly identified primitives of itself to form (A)',(B)',(C)',(D)'. (v). Destroyer (C) removes (E) and (F). (vi). Final situation: two copies of the original machine.

(i).

$$*\underset{A}{\text{(Special-purpose Const)}} + \underset{B}{\text{(General-purpose Const)}} + \underset{C}{\text{(Destroyer)}} + \underset{D}{\text{(Arbitrary)}} \;|^{\downarrow}$$

(ii).

$$\text{(Special-purpose Const)}^{*} + (B) + (C) + (D) \;\Big|\; \underset{E}{\text{(Analyzer)}} + \underset{F}{\text{(Locator)}}^{\downarrow}$$

(iii).

$$(A) + \underset{B}{\text{(General-purpose Const)}}^{\downarrow} + (C) + (D) \;\Big|\; \underset{E}{\text{(Analyzer)}} + \underset{F}{\text{(Locator)}}^{*}$$

(iv).

$$(A) + \underset{B}{\text{(General-purpose Const)}}^{*} + (C) + (D) \;\Big|\; (E) + (F) + (A)' + (B)'$$
$$(C)' + (D)'^{\downarrow}$$

(v).

$$(A) + (B) + \underset{C}{\text{(Destroyer)}}^{*} + (D) \;\Big|\; {}^{\downarrow}(A)' + (B)' + (C)' + (D)'$$

Figure 10

inspection reproduction process can be made considerably simpler, it remains complex, and also still requires that at each reproduction cycle an organ be created which is later destroyed. The source of these characteristics lies principally in the design and ground rules of our underlying automaton system and in particular in the requirement that active strings always have their action directed toward the (sole) possible other string. By re-designing slightly our underlying automaton system we can eliminate this need to create and destroy a subroutine, and can greatly simplify the description of the reproductive process. (See Figure 11.)

In our re-designed automaton system for exhibiting reproduction by means of self-inspection, the initial machine will consist of a *pair* of associated strings (which need not be identical), and reproduction will have taken place when there are *two* pairs of these associated strings. The first string of the initial pair will consist of an analyzer (with restorer) and a constructor (and may optionally include some additional string of primitives not directly taking part in the reproductive process). The action of the analyzer will be directed toward the second string of the initial pair; the action of the constructor will be directed toward producing the second string of the offspring machine pair of strings. The second string of the initial pair will also consist of an analyzer (with restorer) and a constructor. The action of the analyzer will be directed toward the first string of the initial pair, and the action of the constructor will be directed toward producing the first string of the offspring pair of strings.

The reproduction process can now be informally described as

Figure 11. Simplified Reproduction by Self-Inspection. (i). Initial situation. There are *two* initial strings each possessing an analyzer-restorer (A1), (A2), a constructor (B1), (B2) and arbitrary sub-machines (C1), (C2); (A1) and (A2) as well as (B1) and (B2) may differ in their structure but carry out the same functions; (C1) and (C2) are free to be completely different in both structure and function. (ii). The analyzer (A1) and constructor (B1) of the first string of the initial pair read (A2), (B2), (C2) of the second string of the initial pair and produce a copy (A2)', (B2)', (C2)' of the second string. (iii). (A2) and (B2) act on (A1), (B1), (C1) and produce a copy (A1)', (B1)', (C1)'. (iv). Separation of original and new pair is implemented.

(i).

$$*\underset{A1}{\text{(Analyzer-restorer 1)}} + \underset{B1}{\text{(Constructor 1)}} + \underset{C1}{\text{(Arbitrary 1)}} \;\Big|$$

$$^{\downarrow}\underset{A2}{\text{(Analyzer-restorer 2)}} + \underset{B2}{\text{(Constructor 2)}} + \underset{C2}{\text{(Arbitrary 2)}}$$

(ii).

$$(A1) + \underset{B1}{(\text{Constructor})^*} + (C1) \;\Big|\; (A2) + (B2) + (C2) - (A2)' + (B2)' + (C2)'^{\downarrow}$$

(iii).

$$(A1) + (B1) + (C1) + (A1)' + (B1)' + (C1)'^{\downarrow} \;\Big|\; (A2) + \underset{B2}{(\text{Constructor})^*} + (C2) - (A2)' + (B2)' + (C2)'$$

(iv).

$$(A1) + (B1) + (C1) \quad (A1)' + (B1)' + (C1)' \;\Big|\; (A2) + (B2) + (C2) \quad (A2)' + (B2)' + (C2)'$$

Figure 11

follows. The analyzer of the first string of the initial machine examines the second string of the initial machine and constructs a separate copy of the second string. Activation is now transferred to the second string of the initial machine. This second string now examines the first string of the initial machine and constructs a separate copy of it. This new first string of the (offspring) second machine is now activated, and the offspring pair of strings is detached and dispersed (our automaton system provides at present no explicit implementation of this separation process). This completes our description of a simplified form of reproduction by means of self-inspection (and consequent transmission of any acquired characteristics).

This model of reproduction by self-inspection is more economical and elegant since we have eliminated the necessity for construction of temporary substrings and their later destruction. On the other hand, this model of self-reproduction is more complex in that actions must systematically be directed toward several *different* strings, and joining and separating mechanisms for associated *pairs* of strings must be employed, properties we have as yet *not* made an explicit part of our kinematic system.

4.25 An Alternative Strategy of Reproduction by Self-Inspection

We begin with an original string, which produces an analyzer-constructor. This newly created analyzer-constructor reads and analyzes the original string, then constructs a copy of the original string at the end of the original string. The analyzer-constructor then activates and detaches the new copy (employing a *detach* primitive). The original string can now (if desired) employ a *destroy* routine to dissolve the analyzer-constructor. We thus end with two independent activated copies of the original string. (Note: if we allow our analyzer-constructor to persist in a free-floating form, it will act as a reproducer of whatever strings it encounters in the system, including copies of itself. Once multiple free copies of analyzer-constructors exist, there will be no need for the original strings to possess separate constructors for them; a population of analyzer-constructors is self-catalyzing or self-reproducing by itself, but more "interesting" systems would possess either other strings to be copied or would augment the analyzer-constructors with additional routines.)

4.26 Reproduction through Construction of Immature Offspring

In the process of reproduction described in 4.11 (as well as in the process as described by von Neumann (1966)) the offspring is constructed, in *mature form*, by reference to a reserved prior description of the initial machine. In section 4.21 we showed how a machine could reproduce a *mature form* of itself, *without* reference to a reserved prior description of the initial machine. This production under the control of the parent machine, of offspring machines in mature form (and even *identical* to the mature parent) contrasts with the empirical biological situation where an offspring machine is usually released to assume independent existence *before* it has attained completely mature status. We now show how, in a machine system, offspring machines can be produced which are in immature form (in the sense that they are capable of carrying out at least some independent developmental or other action before they in turn reproduce). We show this both for systems in which reproduction depends on a reserved description, and for systems which it does not. Thus, the first method subscribes to the presently accepted theory of reproduction of organisms through use and transmission of reserved hereditary material, while the second method is a logically consistent model of something like the now generally discredited notion of reproduction by reference to the parents' acquired characteristics.

4.27 Rationale of von Neumann's Machine Self-Reproduction

In his theory of self-reproducing automata, von Neumann shows how a parent machine can construct an offspring machine which is a copy of itself. Since von Neumann was trying to show that many if not all complex behaviors of organisms could also be exhibited by machines, it should be noted that this form of reproduction, the offspring being a copy of the entire parent, may itself seem to be unnatural. A more "natural" form of reproduction would first have the parent machine produce an "immature" machine, an incomplete machine, something less than the parent, which, after release from the parent would independently carry itself to completion with the same form and capabilities as the parent. This he could have done in his cellular space format.

At least these points ought to be made. The machine which von Neumann describes has been held down in complexity quite close to what is required for continued self-reproduction. Thus, starting from what is essential, we transmit the essential to the next generation, and each such "essential properties only" generation will be the same.

(He did however suggest that the tape memory of the machine could store instructions for carrying out some tasks not directly connected to the reproductive process.) In any case dealing only with the *essential* you transmit only the *essential* and it must be the same (at least functionally). (It should be pointed out that *essential* here is not precise, and it is possible that we may be able to whittle away at what *seems* to be the essential reproductive system.)

Probably at least as important, von Neumann was explicitly aware of the informal argument against machine self-reproduction, that the process is always "degenerate", that a machine can produce only a machine *less* complex than itself. By having his self-reproducing machine produce a machine which is identical to its parent, he confronted this argument directly, in its baldest form. (He could have had a machine produce a "less" complex offspring, which then autonomously grows to be its parent's equal, but then a separate argument and discussion would have been required to show that the argument from "degeneracy" is refuted.)

Comment. It should also be pointed out that a machine can be designed which will produce offspring possessing *all* the powers of the parent and some *additional* powers not possessed by the parent. Indeed, the process of producing more capable offspring machines can be continued indefinitely. This result was shown by Myhill (1970) who remarks (*op. cit.* p. 218) that it "suggests the possibility of encoding a potentially infinite number of directions to posterity on a finitely long chromosomal tape, a possibility which seems hitherto to have escaped the notice of biologists." The key to Myhill's result lies in the fact that successively "better" formalized systems for arithmetic can be *effectively* discovered and implemented ("better" means that more theorems can be proved and also that their already established theorems may be provable in fewer deductive steps). Each successive machine acts upon a description of itself, produces an improved description of itself, produces an improved design, and constructs an offspring on the basis of the improved design. (Note that in contrast to our system, Myhill's machines act upon *descriptions* of their present selves, supplied by their parent machines and do not inspect or improve themselves directly.)

4.28 Reproduction of Immature Offspring by Reserved Description

We now show how an automaton can reproduce itself by the construction, activation, and release of an "immature" offspring machine, a machine which will go on independently to complete its own development into a copy of its parent.

Our parent machine will consist initially of *two* submachine strings. The first string S_1, of our initial parent machine will be composed of two substrings: a *duplicator* and a *constructor*. The second string of our initial parent machine will be composed of four substrings: a *description of a duplicator*, a *description of a constructor*, a *description of an arbitrary submachine*, and the *arbitrary submachine* itself. (This arbitrary submachine is that part of the mature machine which will *not* initially be part of the independent immature offspring, and must thus be constructed by the offspring itself, in order to complete its development into a copy of its mature parent.)

The process of reproduction is as follows. Activation resides initially in the duplicator of the first string. The duplicator acts upon the second string to produce as part of the offspring machine a string consisting of a copy of all the *description* substrings of the second string (that is, the duplicator takes no action toward the arbitrary submachine itself, although it does produce a copy of the description of this substring). Activation is now transferred to the constructor of the first string. The constructor reads the first two descriptions of the second string (the description of the duplicator and of the constructor) and constructs as part of the offspring machine, a second string consisting of a duplicator and a constructor. This offspring string is then activated and released in conjunction with the first offspring string produced, to create the desired "immature offspring" machine.

At this point the offspring consists of an activated string containing a duplicator and a constructor, and a second string containing descriptions of a duplicator, a constructor, and an arbitrary machine substring. The activated constructor of the juvenile machine now proceeds to read the description of the arbitrary machine substring, and to construct the machine described. At the conclusion of the construction, the offspring machine will have completed its development into a copy of its parent machine. (This procedure is spelled out in Table Two.)

The above procedure is not the only strategy by which a machine can produce an immature offspring capable of independently completing its development. We now describe an alternative strategy in which the offspring machine is given developmental autonomy even earlier in the reproductive process.

TABLE TWO

Parent Machine S_1	Parent Machine S_2	Offspring Machine S_1	Offspring Machine S_2
1. Duplicator 2. Constructor	1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine 4. Arbitrary Machine		

Initial Situation.

Step One: Duplicator of Parent S_1 acts on 1., 2., 3., of S_2 to produce offspring S_2.

Parent Machine S_1	Parent Machine S_2	Offspring Machine S_1	Offspring Machine S_2
1. Duplicator 2. Constructor	1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine 4. Arbitrary Machine		1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine

Situation after Step One.

Step Two: Constructor of Parent S_1 employs Description of Duplicator and Description of Constructor (from either parent or offspring) to produce offspring S_1, consisting of Duplicator and Constructor. Offspring Constructor is activated and the whole offspring machine is released.

1. Duplicator 2. Constructor	1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine 4. Arbitrary Machine	1. Duplicator 2. Constructor	1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine

Situation at immediate conclusion of Step Two.

Step Three: Newly activated offspring Constructor employs Description of Arbitrary Machine to create Arbitrary Machine.

		1. Duplicator 2. Constructor	1. Description of Duplicator 2. Description of Constructor 3. Description of Arbitrary Machine 4. Arbitrary Machine

Situation after Step Three. Parent Machine initial situation has now been achieved in the offspring machine.

We begin with the same initial parent machine. The first string of the parent machine employs its constructor to read the descriptions of the duplicator and constructor, and to construct an offspring duplicator and constructor. This offspring machine is activated to carry out its further development independently. The duplicate routine of the offspring machine reads the second string of the parent machine, and copies into a separate string, for its own use, all the description portions of the parent second string. The offspring now consists of a duplicator and a constructor string and a second string consisting of a description of a duplicator, a description of a constructor, and a description of an arbitrary string. The offspring constructor can now read the description of the arbitrary string and produce the submachine there described, completing independent development into a copy of the parent machine.

4.29 Reproduction of Immature Offspring by Means of Self-Inspection

We now show how an automaton can reproduce itself by creating an "immature" offspring whose specifications are obtained by parent machine *self-inspection* (*not* by employing a prior-existing reserved description).

We begin with a parent machine composed of two strings. The first string will contain an *analyzer* which is equipped with a *constructor* and a *describer* , and a *general-purpose constructor*. The second string of the parent machine consists of an *analyzer* which is equipped with a *constructor*, and the finite but arbitrary "mature" portion of the parent machine. This arbitrary mature portion of the parent is that part of the system which will be independently constructed by the immature offspring.

The reproduction process begins by the analyzer of the first string of the parent machine examining its second string and constructing a *copy* of the analyzer and construction portions, and a *description* of the arbitrary mature portion of the string.

Activation is now transferred to this newly created string (the parent machine remaining quiescent). The analyzer and constructor portions of the new string now examine the parent first string and construct a copy of it. Upon completion of this action, the new string re-activates the parent and detaches itself, to complete its development independently.

The independent offspring now consists of a first string containing an analyzer (along with a constructor and describer) and a general-purpose constructor (with a destructive read out), and a second string containing an analyzer (equipped with a constructor) and a *description* of the "mature" portion of the parent machine. This independent machine can now, autonomously, complete its development into a copy of its mature parent. It does this by having the general-purpose constructor of its first string read the description of the mature substring and construct a copy of that substring (and, since the general-purpose constructor employs a "destructive" reading system, destroying the description in the process). When the construction is complete we end with an offspring machine which is an exact copy of the parent at the time reproduction was initiated.

TABLE THREE

Parent Machine		Offspring Machine	
S_1	S_2	S_1	S_2
1. Analyzer	1. Analyzer		
2. Constructor	2. Constructor		
3. Describer	3. Arbitrary		
4. General-purpose Constructor			

This is the initial situation.

Step One: The *Analyzer* of Parent S_1 examines Parent S_2 and *constructs* a *copy* of the Analyzer and Constructor of Parent S_2 and a *description* of the Arbitrary portion of S_2. This becomes offspring S_2.

1. Analyzer	1. Analyzer		1. Analyzer
2. Constructor	2. Constructor		2. Constructor
3. Describer	3. Arbitrary		3. Description of Arbitrary
4. General-purpose Constructor			

This is the system situation after Step One.

Step Two: Offspring S_2 is now activated. This string now analyzes all of Parent S_1 and constructs a copy of it; this string becomes offspring S_1. Relationship with parent is now severed.

System situation after Step Two.	1. Analyzer 2. Constructor 3. Describer 4. General-purpose Constructor	1. Analyzer 2. Constructor 3. Description of Arbitrary

Step Three: The General-purpose Constructor of S_1 reads the Description of the Arbitrary substring and constructs the machine there described, (destroying the description in the process).

System situation after Step Three.	1. Analyzer 2. Constructor 3. Describer 4. General-purpose Constructor	1. Analyzer 2. Constructor 3. Arbitrary

Chapter Five. Further Directions

In this chapter we present a critique of our principal results, indicate some additional results and suggest some directions further research might take.

5.1 Self-Reference and Self-Exploration: A Critique of our Principal Results

It has been said that no system can unaided obtain its own complete description; that some part of a system will always be inaccessible to inspection from within. We shall evaluate these assertions in the light of the both new and old results described in this paper.

The results of von Neumann of self-reproduction and of Lee on self-description would seem to refute the assertion that a system cannot obtain its own complete description. For in the von Neumann machine-reproduction process, the machine constructs, structural element by structural element, a duplicate of itself, and in the Lee self-description result, the machine produces a description of all of the computationally relevant elements of its program.

Now there are senses of which it might be said that von Neumann and Lee have failed to overcome the claim that no system can unaided obtain its own complete description. For von Neumann and Lee, by *description* is meant a *structural* description, the information for specifying the type and location of all of the basic componentry of the system, and it *may* be that in the above enunciated putative constraints on self-description, something different or in addition, is meant by *description*. For example, *complete* description may be intended to mean not only a description of *structure* but of the *activity* of that structure, or the possible *behaviors* or *computations* of that structure. *Complete* description may even be taken to include the *meaning* or *significance* of the structural or behavioral elements, or a description of the engineering of some implementation of the system or even a detailed description of all the physical or chemical componentry of the system. The von Neumann machine exists in a supporting carefully contrived artificial regular array medium and the Lee self-describing Turing machine exists in a supporting world of assumptions of tape moves and reading, reading, printing, erasing, state changes, etc. None of these supporting assumptions are explicitly expressed or acknowledged in the process of self-description.

In the assertions which head this section, it is also required that the self-describing process be *unaided*, and it might be claimed that despite the putative autonomy of both the von Neumann self-reproducing machine, and the Lee self-describing machine, the "environment of assumption" of the system in which the machines are embedded, somehow spuriously, provides crucial information to the machine. There is also another sense of "unaided" which may be relevant to the discussion. In both the von Neumann and Lee cases, a description of at least some of the system is *ab initio* already a part of the system. In the von Neumann machine a "tape" region of the machine contains (in the form of instructions for construction) a description of all of the non-tape part of the machine (both parts which have a role in the reproduction process, and those which carry out an arbitrary computation); in the Lee result, the machine contains a special sub-routine whose output when analyzed in the proper manner, is not only a description of both the essential self-describing and computationally arbitrary parts of the machine but a concealed form of the special sub-routine itself (from which an explicit description of the sub-routine can be retrieved). Thus, it might be claimed that in both the von Neumann and Lee cases, important self-description information was made available to the machine *ab initio* and that thus the self-description process cannot rightfully be spoken of as "unaided".

The principal goal of the present paper was to explore the extent to which this presence of *ab initio* descriptions could be eliminated. In the von Neumann and Lee cases each arbitrary part of the original system (as well as the inferential and other machinery of the system, including in the von Neumann case, constructional machinery) was obliged to have a description counterpart present initially (indeed, von Neumann argued that it was necessary to his process of self-reproduction). In the principal results of this paper we showed that there need be no counterpart (and thus no concealed description) for either the arbitrary part of the system or the inferential parts of a self-describing system. A part of the initial system produces a separate analyzer which then examines and provides a description of all of the original machine. This procedure would seem to eliminate completely the presence of *ab initio* concealed descriptions, and thus also the criticism that the system is (or has been) aided in its self-description endeavor. Several objections can however yet be raised. The first is that by constructing a new machine which treats the original as object to be analyzed, the integrity of the original machine has been violated (*two* distinct machines being employed, and the original machine thus not being of itself self-inspecting and self-describing). A second objection might be that even if the newly created distinct analyzer is acceptable as merely an auxiliary part of the original machine, in a sense the description of the analyzer itself, is, in barely concealed fashion, present in the original machine. That is, it may be claimed that the special-purpose constructor which produces the analyzer (which can in turn analyze the entire original machine) is merely a *potential* form of both an initially existing machine and a description of that machine. Thus (it may be objected) the procedure has not yet satisfied the criterion that it, "unaided" and "from within" obtain its own complete description. Be that as it may, we have reduced the essential core of the problem to a *fixed finite* part (the constructor of an analyzer) of any otherwise arbitrary and indefinitely large machine.

The (possibly irreducible) "hard core" of machine self-description or reproduction can thus be expressed in terms of an initial machine which contains a fixed finite special-purpose constructor of an auxiliary machine which reads or analyzes the original machine and then acts to produce a duplicate or other transformation of the original machine. (Once a duplicate of the original machine is produced, the process can repeat itself, more duplicator machines being then produced to duplicate yet more "parent" machines. When this happens, the nature of the process can be transformed, since, in a *population* of such machines, a duplicator can act directly on other duplicators to reproduce them, directly without the requirement for any intervening additional original "parent" machines. In a biological interpretation, at this point, variations and combinations of parent machines or duplicator machines can be pictured as proliferating subject to evolutionary selective pressures.)

5.2 Machine Processes of Regulated Growth and Repair

Most of our results so far have been presented as applications of a theory of machine construction and self-duplication, that is, as an analogy to the biological processes of physical growth and reproduction of individual organisms. In this respect, it is clear that one direction in which the research might be extended is in the modelling of regulated growth, development, and repair or regeneration in individual organisms.

The techniques we described in the last chapter provide a means by which a machine can acquire a complete description of its *present* self. The Lee-Thatcher technique provides a means by which a machine can acquire a complete "master" original description of itself. Since a single machine can thus produce for its use a copy of both its present and original self and compare them, the basis for a self-monitoring and self-diagnosing and consequent self-repairing or replacing system is provided.

We outline the organization and behavior of such a self-diagnosing machine system. We begin with an initial string consisting of

(1) an emitter-constructor of an analyzer

(2) a destroyer (of an analyzer)

(3) an analyzer

(4) an inference routine

(5) (optionally) a general processor

(6) an emitter of a description of (1), (2), (3), (4), and (if present), (5).

The machine begins by activating the emitter-constructor and producing an analyzer capable of examining the original string. This analyzer is activated and examines the original and prints out its description in a separate string. Thus the string will contain a description of substrings (1), (2), (3), (4), (possibly) (5), and (6); that is, the present composition of the original string. Activation is now relinquished to the destroyer routine of the original string. This routine destroys the second string (analyzer). The emitter is now activated and produces on the external string a description of the non-emitter part of the original string. Activation is then shifted to the inference routine, which examines the emitter output, infers the structure of the emitter, and prints out its description. The external string will now contain a description of the machine as presently constituted, and a description of the machine obtained by means of the emitter, and these two descriptions can be compared and any discrepancies noted and acted upon.

Of course this self-diagnosis scheme may fail if corruptions of constituents occur in the diagnostic system itself. If however the sum of parts (1), (2), (3), (4),and (6) is small and infrequently employed relative to (5), the scheme might still have considerable utility.

5.3 Regulated Growth

Since in our KAS system we can exhibit a machine which possesses (or which can be designed to possess) a complete description of what its structure should be *and* the capacity to inspect itself and produce a description of what its structure presently is, our system possesses the logical basis for exhibiting machines possessing regulated development (since the machine can constantly adjudicate between what it is (as a consequence of any vagaries in the environment) and what it "ought" to be on the basis of its reserved complete description). The KAS system, as presently formulated, is however completely *deterministic*, and thus provides no scope for exhibiting the uncertainties which would make the modeling of regulation, error control, self-repair, reliability, etc. meaningful.

We will therefore content ourself with suggesting a few very simple modifications of our KAS system and explore briefly some of the ways in which aspects of regulation of development might thereby be exhibited.

1. In our present formulation, a machine in the system always assumes that a primitive newly recruited into the construction process is always of the N type. We might weaken this assumption of determinate arrival of N types, making the environment of the machine less certain, types of primitive other than N presenting themselves at the construction site. The machine would then have to "regulate" its construction action by first subjecting any recruited primitive to a test, and then either modifying or rejecting non-N types.
2. We might severely limit the construction capabilities of our

machines, so that they cannot *directly* construct some arbitrary part of themselves called for in their internalized description. The arbitrary part might be constructed by chance concatenations of primitive component types from the environment. This chance construction could be inspected by the machine and its structure compared with the record of what it *ought* to be, and either permitted to stand, or reduced and dispersed, according as the test is passed or not.

3. Another strategy would lie between these two. In this, the environment would contain various types of primitives (not solely N type primitives) and the direct constructing capability of the machine would be limited to concatenation of the primitive types as they are encountered. This (in almost all cases) "incorrect" structure could then be inspected, compared with the "proper" description, and set right.

5.4 Some Formalizable Problems

We have so far largely confined ourselves to considering single machines interacting with their descriptions, their constructed sub-machines, or their offspring during the process of reproduction. We now briefly consider some problems which arise when we permit several possibly different, simultaneously active KAS strings to interact. A machine can for instance be conceived of as encountering strings of primitive constituents of the same sort of which it is composed. These strings may be passive strings of 0,1 primitives, or can themselves be active or passive machines, and the possible outcomes of their encounters and interactions should be explored.

1. Can in general a machine presented with a machine and a complete structural description of a machine tell in fact whether it *is* a description of the machine? (We have shown that in the KAS a machine can analyze a *passive* machine and can produce a description of it. By slight modification of system properties we could undoubtedly arrange for switching of machine attention between two separate other strings, one a passive machine, one a description in a known uniquely decipherable code. By this means it could be established whether or not the given description was of the given machine. The unresolved issues revolve about cases where the given machine may be *active* and thus may resist or corrupt the analysis process, or even radically transform itself during analysis, and where the primitives may possess different relationships than those assumed by the analyzing machine, and where the coding system may not be completely known, or where unique decipherability may not hold.)
2. Can a machine presented with two strings tell in general if the strings are machines, whether they are identical machines; if they are both descriptions, they describe the same machine; if they are a machine and a description, whether the description is of the machine? (If two strings have the same sequence of primitives, then we might be able to assume that, in the same frame of reference, they have the same significance. If however two strings differ, they may still bear some very close relationship, a relationship which may be difficult to discern. Two machines, structurally different, may carry out the same functions or computations, although their particular courses of action,

their behaviors, may differ. We have (in Section 1) remarked on the metamorphoses of descriptions and machines. Agreement between a machine and its description in understood 0,1 code words can readily be confirmed; what however if a machine M is in the form of a constructor of M, or if M's description is in the form not of its 0,1 code words, but in the form of a standard *emitter* of its description, or even of a *non-standard* emitter of the description and the explicit form is not known by the examining machine?)

3. Can a machine, given any other machine produce a description of it? (The central issue here is whether a machine can analyze all *active* as well as passive machines, and whether the active aspect of a machine, if identifiable, can be appropriately captured in a description.)

4. Can a machine given a string tell if it is a description of itself? (We have shown that a machine can, by several strategies, produce its own description. We have also noted that a description can have many forms, and this fact will complicate the comparison. In addition, a machine, in the process of obtaining its self-description may alter or augment itself, so that the question of *which* self, at *which* time would have to be decided upon.)

5. Can a machine given a machine tell if it is a duplicate of itself? (We have already pointed out that two machines might be computationally equivalent, but structurally different. In addition, if *duplicate* implies an identical pattern of *activity*, then attempts at analysis may produce immediately contradictory

behavior as each machine tries to move to the top of the opposing machine program to begin a mutual analysis procedure.)

6. Can a machine, given a string tell whether it could have constructed the string? (A machine can produce its own description and then simulate the actions of the described machine. If the given string appears as one of the products of the simulation, well and good; if not, the machine may not be able to decide. This "paternity" question is clearly related to the word-problem and other derivability considerations. A simpler, perhaps tractable form of the question would assume that the string could be produced only by a standard emitter or constructor within the machine.)
7. Can a machine, given a machine, tell whether the machine could have constructed *it*? (This is a more convoluted form of question 6., and it is conjectured that the general problem of "knowing one's own father" is unsolvable, while special cases where the offspring is produced in "standard" fashion may be decidable.)
8. Can a machine given any other (possibly *active*) machine destroy it? (The answer here will probably depend very heavily on the design of the particular machine system.)
9. Can a machine destroy itself? (A machine actively destroying its active self seems to make it impossible to elude the issue of active machines clashing. If in a given system it can be proved that a machine *cannot* completely destroy itself, then the issue shifts to producing the smallest or least significant "residue" machine.)
10. Can all machines (including active machines) be constructed?

(For some systems of constructing automata, eg. von Neumann's cellular automaton system, it is known that Garden-of-Eden configurations exist, configurations which can be initially placed in the system but cannot be produced from within the system. The question of creating KAS machines with all possible activated configurations has not been explored.)

In general conclusion, there are probably "reasonable" conditions under which the answers to most of the above questions will be in the affirmative; that is, if the strings to be examined are composed of passively analyzable primitives, if the machines are passive, if the coding employed in the descriptions is known to the inspecting machine and is uniquely decipherable, if the question posed relates to the *structure* of the string, if two machines are the "same" only if they have identical constituent structure.

If on the other hand, not all the primitives are analyzable as passive entities but must be identified by their active behavior, if the machines may be active (and thus possibly alter themselves or their examiners in the course of inspection), if the coding scheme for descriptions is not known to the examining machine, or contains ambiguities, if two machines are to be considered the same if, though their structure differs, their computation is the same, then many of the questions posed above may be in the negative.

5.5 Some Problems of Sophisticated Systems

The "formal" questions posed in the last section have some considerable bearing on specific problems of complex systems. For example, the essential problem of the immune system of vertebrate organisms is to be able to create a mechanism which will selectively recognize, disable and expel entities not like the self and at the same time to tolerate (which is itself a form of recognizing) the componentry of the self. This basic dichotomy - self vs. not-self - (and consequent tolerance or elimination) can be further ramified so that among the self entities those which are to be repaired are distinguished from those which are to be replaced, etc. and among those which are not-self whose which are to be clumped and expelled are distinguished from those which are to be sought out, broken down, and consumed.

The notion of identifying and behaving differently toward constituents of self and not-self seems to imply that such a system must possess *internalized descriptions* not only of self but also of the non-self environment of the system. The system must model within it, not only what it itself should or should not be, but what its environment should or should not be.

So equipped, it will be seen that such a system will possess the logical basis for regulating the conditions of its local environment, viz., the system may possess an internalized standard model of the desired condition of the system proper and also its environment, its "external self", the means of generating descriptions of the actual status of the system proper and its environment, and the means to alter itself or the environment in ways to move the actual situation closer to the desired.

It should also be noted that while macro-molecular biochemical systems provide a very natural interpretation of the systems and processes we have described here, other interpretations (and thus applications) are possible and should be considered. The mechanisms we have described can readily be interpreted as cognitive or even social processes. For a system (an individual organism, a family, a society or nation) to discern, to employ foresight, to plan ahead, indeed to exhibit many other sophisticated psychological or social behaviors, it must to some degree have available to itself a model both of itself and its pertinent relationships to the world about it, as well as some capacity for comparing models of what *is* with what *should* be, and means to affect what is so as to alter it in the direction of what should be. (For an examination of this notion of the use of internalized models of self and environment in psychological and social behavior see Craik (1943), Boulding (1956) and Miller, *et al* (1960).)

REFERENCES

1) Arbib, M. (1966), "Self-reproducing Automata: Some Implications for Theoretical Biology", *Towards a Theoretical Biology 2. Sketches*, edited by C. H. Waddington, 204-226, Aldine Pub., Chicago.

2) Arbib, M. (1966), "Simple Self-Reproducing Universal Automata", *Information and Control*, 9, 177-189.

3) Boulding, Kenneth (1956), *The Image*, University of Michigan Press.

4) Burks, A. W. (1961), "Computation, Behavior, and Structure in Fixed and Growing Automata", *Behavioral Science* 6, 5-22.

5) Chaitin, G. (1974), "Information-theoretic Limitations of Formal Systems", *Journal of the ACM* 21, 403-424.

6) Craik, Kenneth (1943), *The Nature of Explanation*, Cambridge University Press.

7) Laing, R. (1975), "Some Alternative Reproductive Strategies in Artificial Molecular Machines", *J. Theoretical Biology* 54, 63-84.

8) Laing, R. (1976), "Automaton Introspection", *J. Comp. and Systems Sci.* 13, 2, 172-183.

9) Lee, C. Y. (1963), "A Turing Machine Which Prints its Own Code Script", *Mathematical Theory of Automata*, Polytechnic Press, Brooklyn, New York, 165-171.

10) Miller, G., E. Galanter and K. Pribram (1960), *Plans and the Structure of Behavior*, Holt. New York.

11) Minsky, M. (1963), "Steps Toward Artificial Intelligence", in *Computers and Thought*, edited by E. Feigenbaum and J. Feldman. McGraw-Hill, 406-450.

12) Minsky, M. (1968), "Matter, Mind, and Models", in *Semantic Information Processing*, edited by M. Minsky, MIT Press, 425-432.

13) Moore, E. F. (1970), "Machine Models of Self-Reproduction", in *Essays on Cellular Automata*, edited by A. W. Burks, U. of Illinois Press, 187-203.

14) Myhill, J. (1970), "The Abstract Theory of Self-Reproduction", in *Essays on Cellular Automata*, edited by A. W. Burks, U. of Illinois Press, 206-218.

15) Thatcher, J. (1963), "The Construction of a Self-describing Turing Machine", *Mathematical Theory of Automata*, Polytechnic Press, Brooklyn, New York, 165-171.

16) Thatcher, J. (1970), "Self-describing Turing Machines and Self-reproducing Cellular Automata", *Essays on Cellular Automata*, edited by A. W. Burks, U. of Illinoss Press, 101-131.

17) Turing, A. M. (1936), "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proc. of the London Mathematical Soc.*, series 2, 42, 230-265.

18) von Neumann, J. (1951), "The General and Logical Theory of Automata", in *Cerebral Mechanisms in Behaviors - The Hixon Symposium*, edited by L. H. Jeffress, John Wiley, 1-41.

19) von Neumann, J. (1966), *Theory of Self-reproducing Automata*, edited and completed by A. W. Burks, U. of Illinois Press, Urbana.

20) Wagner, E. (1967), "On the Structure of Programming Languages", *Conference Record: Eighth Annual Symposium on Switching and Automata Theory*, IEEE, New York, 45-54.

21) Wang, E. (1957), "A Variant to Turing's Theory of Computing Machines", *Journal of the ACM* 4, 63-93.

22) Watson, J. D. (1976), *The Molecular Biology of the Gene*, Third Edition, W. A. Benjamin.