J. theor. Biol. (1977) **66**, 437–456

Automaton Models of Reproduction by Self-inspection

RICHARD LAING

Logic of Computers Group, Department of Computer and Communication Sciences, The University of Michigan, Ann Arbor, Michigan 48104, *U.S.A.*

(*Received* 23 *February* 1976, *and in revised form* 13 *July* 1976)

There is a strategy by which an automaton can examine itself to obtain a complete description of its own constituent structure. This strategy is here employed to show that machine self-reproduction by means of self-inspection is possible. Since the parent automaton structure serves as the sole model for the structure of offspring, the result shows that there is a logically consistent strategy of self-reproduction in which any acquired structural characteristics of parent could perforce be transmitted to the offspring.

1. Introduction

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 could, for example, mean that the active protein machinery might somehow analyze itself and use this information to implement reproduction, only incidentally if every being in possession of a separate 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-production. Indeed, Burks (1961) had already conjectured that it is possible for a machine to inspect all of its structure and obtain a complete description of its structure which it can store for its use within a proper part of itself. Laing (1976) has 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 structural description.

In the present paper, 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, there exists a logically consistent strategy of reproduction in which acquired characteristics of parent could be transmitted to the offspring.

In the rest of this introductory section, we describe briefly and informally our kinematic system, and list its essential features; we then explain how reproduction by self-inspection can therein be carried out. (We call our automaton system a kinematic system although it might better be termed a "quasi-kinematic" system since it employs not only the notion of spatial physically interacting componentry but also state-changing notions drawn from the cellular automaton concept.)

The System. The basic components of our system consist of strands or *strings* of primitive constituent finite state automata, these component strings being in sliding contact. [See Laing (1975) for a general discussion of several kinds of such systems.] A primitive constituent of a string can be in an activated or a passive state. An active primitive constituent in contact with a passive constituent of another string will interact with the passive constituent in precisely defined ways only. These ways include *changing* the state of the contacted passive primitive, *reacting* to the state of the contacted passive primitive, *sliding* to the next neighbor of the contacted passive primitive. Since one string (an active string) can be designed to play the part of any Turing machine finite-state read-head, and another string (a passive string) can be designed to play the part of a Turing machine tape, we can carry out any Turing computation in this kinematic machine system. (See Fig. 1 for an illustration of what a Turing machine would look like in our system.)

Some important features

The following are some behavioral and organizational properties of our system important in securing our results.

(1) The major components of our system being connected *strings* of simpler cell-automaton primitives, one string can, by sliding, obtain direct contact access to all primitives of a second string. [This contrasts with von Neumann's two-dimensional "checkerboard" cellular automaton system (von Neumann, 1966) where access to interior cells of a machine can be attained only by penetrating the intervening exterior cells of the machine.]

(2) At any one time, only a *single* primitive constituent need be activated in order to carry out the actions necessary for our inspection and reproduction results. (This again contrasts with the von Neumann cellular system where although the general course of system action as he described it is sequential, in implementing certain processes, activation is sometimes required to be simultaneously present at many locations.)

(3) *Both* strings of the system can contain potentially active primitives, and the active and passive roles of the strings can be exchanged. (This contrasts with the usual Turing machine notion where the roles of active automaton and passive tape remain fixed.)

(4) The assumption is made that available to the system is a population of passive, "null" primitive constitutents. These primitives can be recruited and attached at the ends of passive strings. (This is analogous to the "automatic" addition of blank squares at the end of a Turing machine tape, and related to the von Neumann cellular system notion of a machine having available to it an environment of quiescent cells.)

(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 conversion of quiescent cells of the von Neumann cellular system to any of the other cell-types.)

(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 is the central concept employed in this paper in establishing the self-inspection result which obviates the need for any permanent and separate passive description in self-reproduction. The central idea of the analysis process is as follows. As part of the system, there is a primitive which when active and in contact with a null primitive "detects" this fact by altering its own behaviour (it communicates a different activation). In paragraph (5) above we have said that the system can, by a series of distinct actions convert any known primitive back to a null. Alternatively we can subject an *unknown* primitive to a series of conversion actions and test (with our null-detector) for the arrival of the null status. The record of the conversion actions which reduced the unknown primitive to the null status reveals what the primitive type must have been before the conversion to null took place. By this mean the system can identify

unknown primitives. It should also be pointed out that not knowing what the primitive in question was before the conversion process began, and knowing that it is presently null, we can by a series of constructive conversion actions, restore the primitive to its original type.

Outline of principal results

The result, that a machine can achieve complete self-inspection and by this means obtain the information needed for self-reproduction, can take many forms. We first briefly outline the version of the result as prepared for in section 4 and presented in section 5, and then also the informal version of section 6.

In the first version, the original machine is a single string of constituents. This initially active string constructs a second string consisting principally of an analyzer. The first string relinquishes activation to this second string, which then analyzes the original string and produces a description of it. Activation is then relinquished back to the original machine which employs the newly available description of itself to produce a new copy of itself. The original machine may now (optionally) destroy (reduce to null primitives) the second string (along with the temporary description) so that we are left with the original initial machine and a copy of it. (See Fig. 4 for an illustration of this process.)

In another version (that of section 6), the original machine consists of *two* strings which need *not* be identical, but each containing, in some form, an analyzer and constructor. The initially active string analyzes its passive partner and constructs a copy of it. It then relinquishes activation to the passive partner which analyzes the initially active string and constructs a copy of *it*. The system has thus reproduced itself without recourse to a distinct description of itself. (See Fig. 6 for an illustration of this process.)

We conclude this introduction with some remarks on our use of language. When in our machine system an active primitive has contacted a passive primitive and activation has afterward been routed to precisely one of two different possible adjacent primitives according to whether the passive primitive was a type we call *zero* or a type we call *one*, we shall simply say that the machine *read* the primitive; when a certain primitive is now active and because of this activation all further action the machine may take will be different than if some other primitive was now active, we may simply say that the machine now "knows" something it did not know before. That is, where used in respect to machine behavior, such words as "read", "know", "recall", "discover", "discern", etc. have a precise technical interpretation in terms of specific, completely determined actions and states; such words are employed solely as an expositional shorthand, and are not intended to suggest the presence or necessity in the machine system of any unexplained, or unanalyzed psychological capabilities.

2. A Kinematic Machine System

The machines for which our results are shown are *kinematic machines* which compute and construct by means of a repertoire of simple mechanical actions: viz. sliding local shifts of contact, local changes of state, and local detections of such state changes.

A machine will consist of basic simple finite state automaton *primitives*, organized into *strings*. The primitives (and consequently the strings) possess a *forward* and *backward* direction. For most of our discussion the specific machines employed will consist of at most two separate strings of interacting primitives. 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. In operation, the two strings are always in sliding contact at a single point. One characteristic 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 immediate neighbor primitive in the forward direction of the active string.

We now describe the basic primitives of our system.

Three primitives N (*null*), 0 (*zero*), 1 (*one*) will principally be employed in the passive recording of information. 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. A string consisting entirely of N, 0, and 1 primitives thus takes no action toward other strings.

The two primitives P0 (*print zero*) and P1 (*print one*) will principally be employed in acting upon N, 0, and 1 primitives, to change their states. A P0 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 P0 or P1 will automatically relinquish activation and contact to its next neighbor primitive in the forward direction in its string. (Thus, in contrast to strings consisting entirely of N, 0, and 1 primitives, these primitives and others to be described below, can be activated to produce actions on primitives of other-strings.)

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 assumed always to be available and to be automatically attached to the end of the passive string. 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.

The primitive H (*halt*), if activated, terminates all activity.

There are three conditional transfer primitives, of the form TX, y where X is N, 0, or 1 (the *condition* to be satisfied) and y is the *address* of a primitive in the program. If for example a TN, y primitive is active and in contact with an N primitive in the passive string, then activation and contact are routed to the primitive with the address y in the active string; when an activated TN, y is in contact with a passive primitive other than N, then TN, y simply relinquishes activation and contact to the next neighbor in the forward direction.

The primitive Ty is the *unconditional* transfer which when activated will shift activation and contact to address y.

[The above description of conditional and unconditional transfer action does not explain how the active string actually carried out shifts of activation to an arbitrary address y in the active string. There are many ways in which transfers may be implemented. Here, for expositional simplicity, we merely assume that such transfers can be effected. Various transfer implementation techniques for kinematic machines are discussed in Laing (1975, 1976).]

The A (*activation*) primitive when activated will produce an activation of the *passive* primitive contacted and an immediately subsequent *loss* of activation in the active string. The effect of this primitive is to produce an exchange of the active and passive roles of the strings.

The AD (*activate and detach*) primitive when activated will produce an activation of the passive primitive contacted and an *immediate detachment* of the two strings, *without* a loss of activation in the active string. The role of this primitive is to create an independent activated string.

The C (*conversion*) primitive is such that if active it will systematically convert any passive primitive (including a passive C primitive) with which it is in contact into another primitive type. The C conversions are set forth in Table I. Notice that the conversion sequence forms a closed loop: some specific number of C actions will convert any primitive type to any other primitive type, including itself. The function of the C primitive is to enable us to obtain any of our primitive types, beginning with a freely available N primitive, and to provide a means by which the type of an arbitrary unknown primitive can be ascertained.

(This latter capacity of the system is critical to our principal result, for as will be seen in the next section, it will enable the system to determine its own

TABLE 1

C conversions

C(N) → 0	C(T0) → T1
C(0) → 1	C(T1) → TN
C(1) → P0	C(TN) → T
C(P0) → P1	C(T) → AD
C(P1) → P	C(AD) → A
C(F) → B	C(A) → C
C(B) → H	C(C) → N
C(H) → T0	

structure. It should be emphasized though, that the analysis method described here and upon which the complete self-inspection result depends, is not the only tactic, employing a "biological vocabulary", by which the determination of primitive constituents could be carried out. It is the logical equivalent of any of several means by which a system can ascertain the type of its own constituents. The C-conversion procedure was chosen since it is a straightforward adaptation of the biological-like process of synthesis of primitives required for construction in our system. We might, alternatively, have made the analysis procedure distinct from the construction procedure by introducing the notion that every primitive has permanently attached to it a RNA-like complex in which the anti-codon characters can be read, one-by-one, by a set of character detecting primitives.)

3. Some Basic Kinematic Machines

In this section we describe some basic kinematic machines and establish some preliminary results; in section 4 we will combine these machines and results to show that there is a machine possessing the sought after capacity to reproduce by employing self-inspection.

RESULT 1

Arbitrary Turing computations can be carried out in the kinematic system. This follows immediately from the fact that the properties of our kinematic system with active strings composed only of F, B, P0, P1, TN, T0, T1, and H primitives and passive strings of N, 0 and 1 primitives only can carry out the actions of any Wang "programmed" Turing machine, a class of machines which (Wang, 1957) has been shown capable of carrying out any Turing computation. (Fig. 1 illustrates what a Turing machine would look like in our kinematic machine format.)

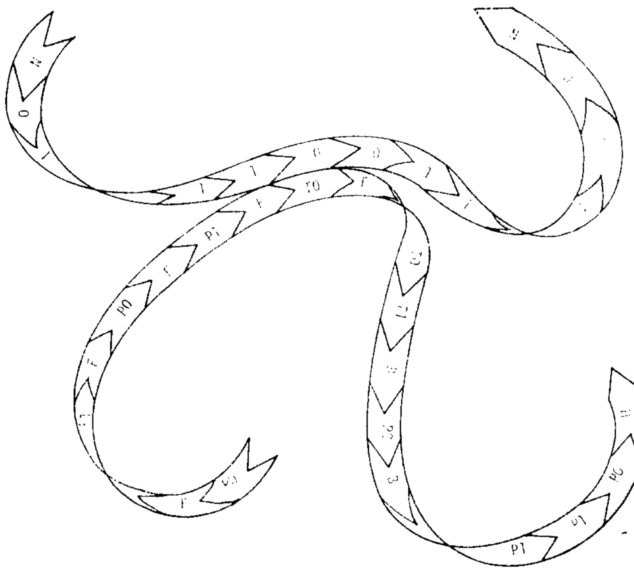

FIG. 1. A Turing machine in kinematic machine format. The lower string consists of program primitives which if activated will act on the passive "tape" primitives of which the upper string is composed.

Fixed sequence emitter

A fixed sequence emitter is a string consisting entirely of P0, P1 and F primitives, each P0 or P1 followed by an F. When activated, such a string of primitives (acting on a passive string consisting of N, 0 and 1 primitives) will produce a particular sequence of zero and one primitives. Clearly for any particular fixed finite sequence of zeroes and ones desired, there is a unique fixed, finite emitter which can be designed to create the desired sequence.

Example. The emitter P1–F–P0–F–P1 would (starting with a string consisting entirely of N, 0, or 1 primitives) produce the string 1–0–1.

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 concreteness we might assign N = 0000, 0 = 0001, 1 = 0010, P0 = 0011, etc.) The description of a kinematic machine will be the code words of the primitives of each of its strings in order (beginning at the top of the originally activated string and proceeding in the forward direction) and (if necessary) employing a special code word marking the end of the first string and the beginning of the second. For any machine, a fixed sequence emitter can be designed which will print out the description of the machine.

Emitter inferrer

An emitter inferrer is an active string which can examine a passive zero–one description and infer the composition of the fixed sequence emitter which produced it. The inferrer works as follows. It reads the first primitive of the passive description string (by means of T0, T1 primitives). If the first primitive is a one, (as in the emitter example above) the inferrer "knows" that it had to have been produced by a P1 primitive present as the first primitive of the emitter (similarly, if the first description primitive was a *zero*, it must have been produced by a P0, as the first 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 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 emitter primitives.

Special purpose constructor

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

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

Analyzer

There exists an (active) string which when presented with any (passive) string can analyze and identify the passive string primitives 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 a unique number of C stimulations followed by a TN "test and transfer" primitive which will enable a machine to ascertain the type of any primitive, and then to transfer to a location where this information can be acted upon.

Example. In the analyzer segment TN-C-TN-C-TN-C-TN-C-TN . . . the first TN will detect if the unknown primitive is already an N, the second

TN will detect if the unknown primitive is a C, the third TN will detect if the unknown primitive is an A, etc. (See Table 1; a part of an analyzer is illustrated in Fig. 2.)

Analyze and restorer

The anlyzer can be augmented with distinct construction routines such that after a primitive type has been determined (by reduction to N) an appropriate re-construction routine (of active C primitive applications) can restore the primitive to its original form. This is possible because the system will know what the primitive now is (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.

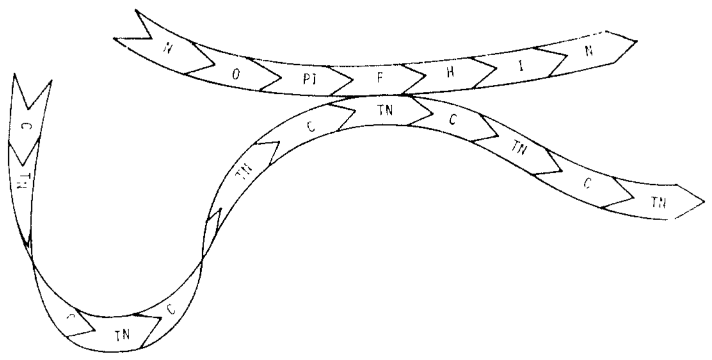

FIG. 2. A segment of analyzer. The lower string represents a segment of analyzer which subjects primitives of the upper string to a series of C conversion stimuli, and then tests (by means of TN primitives) whether the primitive under analysis has yet been converted to N. In the case illustrated, the F primitive in contact will have to undergo ten more C stimuli before it is detectable by a TN.

Example. If the analyzer has just established that the unknown primitive was an AD (i.e. exactly *three* C stimulations have converted the unknown to an N), then the TN which detected this can route activation and contact to a "restore" route C–C–C–C–C–C–C–C–C–C–C–C (12 C stimuli) which will convert the present N back to the AD it was originally.

Special-purpose constructor inferrer

As with the emitter of a *description*, whose structure can be inferred from the description produced, so can the unique *constructor* of any string be inferred from the string produced. This is done as follows. An analyzer determines the primitive type, and then transfers to an inference routine. For each primitive type, a unique number of C stimulations were required to convert an initial N to the discovered type. A description of the required number of C primitives can be set down, followed by the description of an F primitive, necessary in the original construction routine to move on to the next N to be converted. We then transfer back to the analyzer to determine the next primitive, etc.

Example. Given the string P1–F–P0–F–P1 (our example emitter which was constructed by our example constructor) the inferrer analyzes the first primitive and determines that it is a type P1; a P1 is produced by C–C–C–C, and a description (in the zero–one code) for four successive C's can be set down. The emitter then moves on to analyze the F primitive, etc.

Special-purpose fixed destroyer

For any particular known (passive) string, an (active) string consisting of C primitives and F primitives only, can be designed which can reduce each primitive of the string to an N.

Example. We wish to "destroy" the string T–A. This can be done by the string C–C–C–C–F–C–C (see the C conversion table).

General destroyer

Any (initially unknown passive) string can be completely reduced to N primitives, either by analyzing the primitives in turn to determine their type and then applying the requisite number of active C primitives, or by applying C primitives one-by-one and testing for the arrival of the N status.

Substring locator

We shall frequently wish to have our active string move along the passive string and locate a particular region or sub-string of the passive string. The regions of interest in the passive string can be prefaced by a unique zero–one sequence label, and we can have the active string identify a sought-for label by carrying out a scanning and decoding routine. For example, starting at the top end of the passive string, the active locator routine will search for the first symbol of the desired label; if it is found, the locator examines the next primitive to see if it is also correct. If it is, the decoding continues. If it is not, the locator returns to its initial state and moves on to the next label code block and begins again the decoding process.

RESULT 2

There exist self-describing kinematic machines. Self-description of a kinematic machine will take place if beginning with an active string (either alone or together with an all null passive string) the active string finally halts having created a passive string containing a complete correct description of itself in the agreed upon zero–one code. This can be accomplished (Laing, 1975; Lee, 1963; Thatcher, 1963) as follows.

The self-describing machine will be an active program string composed of the following substrings: any desired (optional) finitely long substring, an emitter inferrer, and an emitter of the description of the optional substring (if any) 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 (if included) 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. (See Fig. 3 for an illustration of the self-description process.)

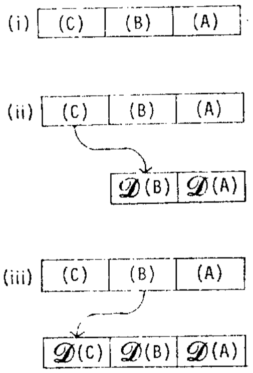

FIG. 3. Self-description process. (Note: no attempt has been made to reflect the fact that the string segments pictured would possess very different relative sizes.) (i). An initial optional segment (A), an inferrer (B), and emitter (C) of the description of (A) and (B). (ii). The emitter (C) produces its stored description of (A) and (B). (iii). The inferrer (B) now reads the description of (A) and (B), and from this infers the description of (C), and produces it, completing the self-description process.

RESULT 3

There exist general-purpose kinematic machine constructors. A general-purpose kinematic machine constructor is an active string such that if it is given the description of any string, the constructor will produce the string desired. Such a general-purpose constructor will work as follows. It will read the first code block of the description and determine the primitive type to be constructed. It will then transfer to the subroutine for the production of such a primitive, viz., the number of continguous C primitives whose application will produce the desired primitive. Activation will then be transferred back to the description reading routine, where the description of the next desired primitive is decoded, etc. At the conclusion of construction, the newly produced machine can be activated by means of an A primitive. Note that the general-purpose constructor can, in reading a description, be equipped to preserve or to destroy the description in the process.

RESULT 4

There exist self-reproducing kinematic machines. Since [by the Lee (1963), Thatcher (1963) results; our result 2] a kinematic machine can produce its *own* complete description, and since given any description of a machine a kinematic machine exists which can construct the machine, a machine consisting of a self-describer and a general-purpose constructor can reproduce itself (Thatcher, 1970).

Result 4 is a variant of the original von Neumann (1966) result that an automaton can reproduce itself if supplied with a description of itself. It is an automaton model of the logic of biological reproduction as it is believed actually to take place in living organisms, viz., by means of a nucleic acid reserved description acted upon by enzymatic protein macro-molecular machinery.

The principal result of the present paper, that in self-reproduction the presence of an explicit prior description can be dispensed with entirely, is shown in the next section.

4. Reproduction by Self-inspection

We are now prepared to show how a machine can reproduce itself, using itself as model.

RESULT 5. (PRINCIPAL RESULT)

Machine reproduction by self-inspection can be exhibited in the kinematic system. We begin with an initial single string "parent" machine consisting of the following substrings.

(A) A special purpose *emitter-constructor* which can construct a new *second* string. This second string which (A) constructs will consist of (1) an *analyzer* *and restorer*, (2) an *inference routine* of the sort which can take a description and infer the description of the emitter which could produce the given description, (3) a general purpose constructor, (4) 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 Fig. 4).)

(i) (A) is activated and constructs the new second string consisting of (1) (analyzer–restorer), (2) (emitter inference routine), (3) (general-purpose constructor), and (4) (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 (E) 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, (5), the description of (A) (B) (C) (D).

The first string now consists of (A) (B) (C) (D) (E) and the second of (1), (2), (3), (4), and (5), the description of (A) (B) (C) (D).

(iv) Activation is now relinquished to (4) the destroyer, of the second string. (4) then reduces (E) back to null primitives.

(v) Activation is now relinquished back to (B) of the original string which destroys all but (5) the description of the second string.

(vi) Activation is transferred to (D), the constructor of the first string, which using the description (5), 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 (5) 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.

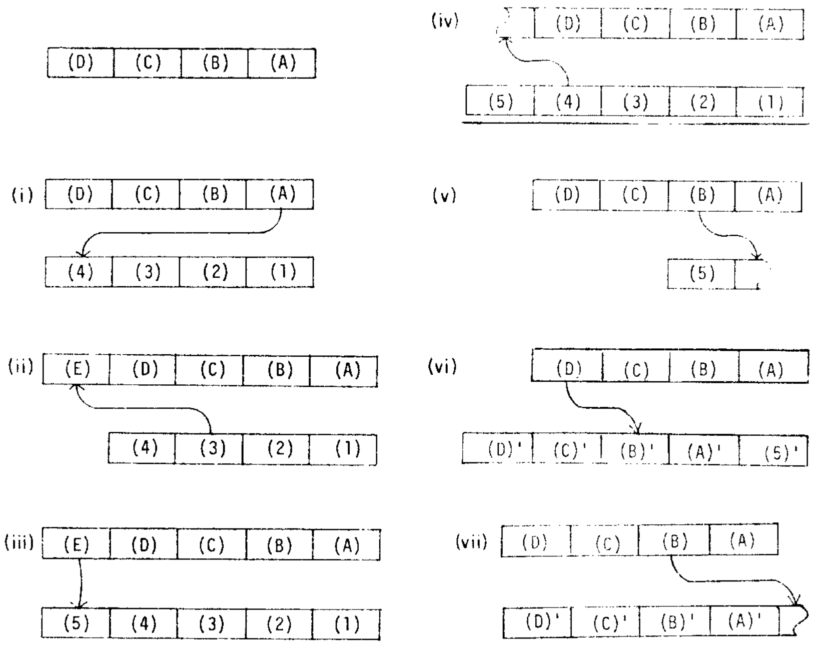

FIG. 4. Reproduction by self-inspection. Initial situation. (A) special-purpose constructor (B) destroyer, (C) optional substring, (D) general-purpose constructor. (i) (A) constructs (1) analyzer, (2) inferrer, (3) general-purpose constructor, (4) destroyer. (ii). analyzer (1) identifies the primitives of (A), (B), (C), (D), and inferrer (2) uses this information to instruct general-purpose constructor (3) to produce (E), the emitter of a description of (A), (B), (C), (D). (iii). The emitter (E) produces the description (5) of (A), (B), (C), (D). (iv). Destroyer (4) removes the emitter (E). (v). Destroyer (B) removes (1), (2), (3), (4). (vi). Constructor (D), using description (5), produces (A)′, (B)′, (C)′, (D)′. (vii). Destroyer (B) removes description (5).

5. 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 new active analyzers and constructing routines, and 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 Fig. 5.) 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".

RESULT 6

Machine reproduction by self-inspection without recourse to temporary self-description can be exhibited in the kinematic system.

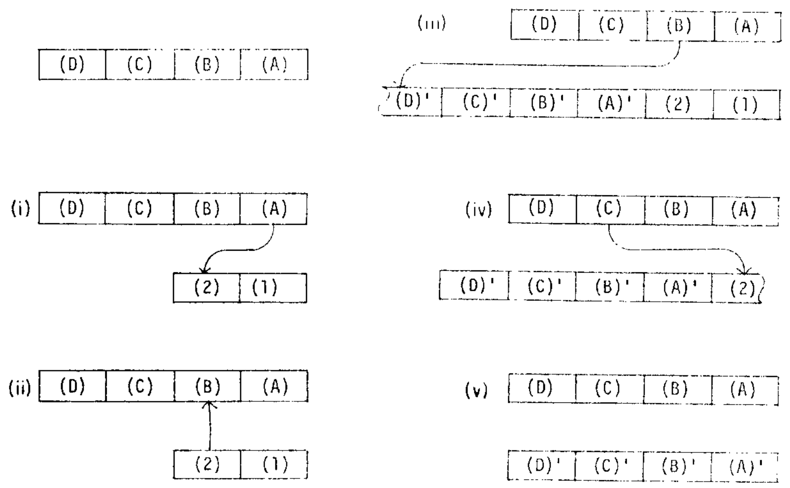

FIG. 5. Temporary description dispensed with. Initial situation. Initial machine consists of (A), a special-purpose constructor of an analyzer and a locator, (B), a constructor, (C), a destroyer, and (D), an arbitrary substring. (i). (A) constructs the second string consisting of (1) analyzer and (2) locator. (ii). Analyzer (1) inspects original machine, primitive-by-primitive, and communicates identities of primitives to locator (2) which activates appropriate constructor region within (B). (iii). Constructor (B) produces copies of the newly identified primitives of itself to form (A)′, (B)′, (C)′, (D)′. (iv). Destroyer (C) removes (1) and (2). (v). Final situation: two copies of the original 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 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.

6. Simplified Reproduction by Means of Self-inspection

Although by the construction of the last section, the self-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 Fig. 6.)

RESULT 7

Machine reproduction by self-inspection without creation and destruction of auxiliary strings can be exhibited in the kinematic system.

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 string. 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 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 reconomical and elegant since we have eliminated the necessity for construction of temporary

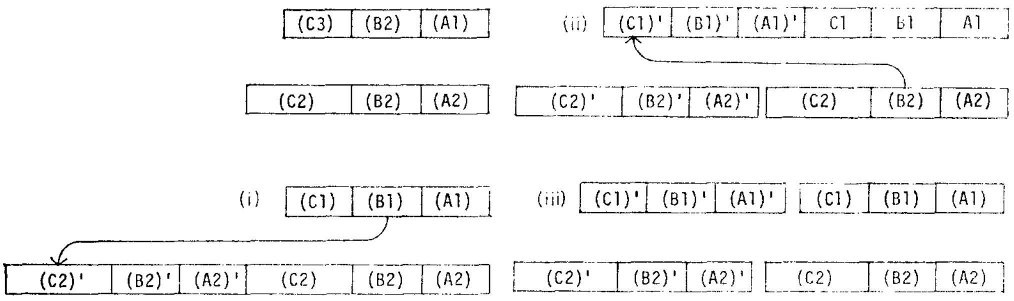

FIG. 6. Simplified reproduction by self-inspection. Initial situation. There are *two* initial strings each possessing an analyzer-restorer (A1), (A2), a constructor (B1), (B2); (A1) and (A2) as well as (B1) and (B2) may differ in their structure but carry out the same functions; (C1) and (C2) are free to be completely different in both structure and function. (i). 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. (ii). (A2) and (B2) act on (A1), (B1), (C1) and produce a copy (A1)′, (B1)′, (C1)′. (iii). Separation of original and new pair is implemented.

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.

7. Discussion

It should be emphasized that what we have presented here is not an abstraction from and explanation of any particular known biological reality, but an abstraction from and explanation of a *possible* biological reality. It can thus be viewed as a contribution to a true theoretical biology which takes as subject matter the explication and explanation of the processes of all *possible* living systems, not merely those which are now known to exist, and which engage our interest and attention.

The machine self-reproduction result of von Neumann showed that machines are capable of carrying out a process once widely thought possible only for biological organisms. Our results in this paper show that machines are capable of carrying out a process which had quite generally been thought impossible for *both* machines and organisms, viz. complete self-inspection of structure, and by this means, self-reproduction. It is thus clear that a physical *artificial* automaton system could be designed and constructed which would exhibit this self-inspection and self-reproduction process. Whether reproduction by self-inspection (by the strategy employed here, or by any other strategy) is presently exhibited (or has ever been exhibited) in any *naturally* occurring organism or system is a question of great interest.

The capacity of a system generally to explore its own structure and produce a complete description of it for its perusal and use (for example, in generation and evaluation of behavioral options open to it) seems a valuable one, and if such a *prima facie* advantageous capacity is *not* exhibited anywhere in naturally occurring systems, this in itself seems of interest. It is possible of course that *complete* self-inspection, possibly necessitating special properties and organization, is costly relative to its benefits, and that a simpler *partial* self-inspection capacity is, all things considered, superior.

In either case, the techniques devised here by which a system can extract and employ partial or complete models of itself may contribute to the explication of the logical bases of several complex biological processes closely allied to reproduction. For example, development can in part be seen as a process of reconciling by the appropriate constructive or destructive action a phenomic description of what the organism *is* at the present instant, to what a reserved genetic description says it *should* be. Since our system possesses the means by which *both* present phenomic and "ideal" genomic descriptions can be obtained (as well as the constructive and destructive capacity for implementing system modification) we thereby have the basis for a logical explication of the processes of development. Similarly with homeostasis, and repair and re-generation, since these processes too take the same logical form of comparing the *actual* state of affairs with a stored *standard*, and then acting to reconcile the differences between the two.

This research was supported in part by the National Science Foundation through Grant No. DCR71-01997.

REFERENCES

ARBIB, M. (1966). *Towards a Theoretical Biology* 2. *Sketches* (C. H. Waddington, ed.), p. 204. Chicago: Aldine.
BURKS, A. W. (1961). *Behavioral Sci.* **6**, 5.
LAING, R. (1975). *J. theor. Biol.* **54**, 63.
LAING, R. (1976). *J. Comp. System Sci.* **13**, 172.
LEE, C. (1963). *Mathematical Theory of Automata*, p. 155. Brooklyn, New York: Polytechnic Press.
THATCHER, J. (1963). *Mathematical Theory of Automata*, p. 165. Brooklyn, New York: Polytechnic Press.
THATCHER, J. (1970). *Essays on Cellular Automata* (A. W. Burks, ed.), p. 101. Urbana: University of Illinois Press.
VON NEUMANN, J. (1966). *Theory of Self-reproducing Automata* (A. W. Burks, ed.). Urbana: University of Illinois Press.
WANG, H. (1957). *J. Assoc. Comp. Mach.* **4**, 63.