

TOWARD A THEORY OF INTRINSIC COMPUTABILITY

MARCO GIUNTI

Università di Cagliari, Italy

Abstract: This paper investigates the foundations of the view that (i) computation theory is a special branch of dynamical systems theory, and (ii) a computational system is a special kind of discrete dynamical system whose specific property, *being computational*, can be thought to be *intrinsic* to the dynamics of the system itself. I give a formal explication of the concept of an intrinsic computational system, and I then consider its possible consequences for computability theory and Turing-Church thesis.

Key words: dynamical systems theory, discrete system, computational system, computation, computability theory, recursive function, effective procedure, Turing-Church thesis, machine, mechanism, representability.

1. INTRODUCTION

The main goal of this paper is to investigate the foundations of a new approach to computation theory. The central tenet of this view is that computation theory is a special branch of dynamical systems theory, and its objects (the *computational systems*) are a special kind of discrete dynamical systems; the specific difference of these objects, i.e. the property of *being computational*, can be thought as an *intrinsic* property of their *dynamics*.

The intuitive concept of a computational system admits both an extensional and an intensional characterization. Extensionally, the term *computational system* refers to any device of the kind studied by standard (or elementary) computation theory. Thus, for example, Turing machines, register machines, cellular automata, and finite state automata are four different types of computational systems. Discreteness and determinism are two properties shared by all such devices; therefore, so called analog

computers are not computational systems in this sense. Intensionally, computational systems can in fact be thought as dynamical systems of a special kind; from an intuitive point of view, the computational systems can be identified with those discrete, deterministic, dynamical systems that can be described or represented effectively.

Giunti (1992, 1997) gave a formal explication of this intuitive concept, and showed that Turing machines and all other systems that have been actually studied by standard computation theory (register machines, cellular automata, monogenic production systems, etc.) satisfy the formal definition.

According to this definition,¹ a discrete dynamical system DS is computational just in case there is a recursive representation of DS , where a *recursive representation*² of DS is a pair $(u, DS_{\#})$ such that: (i) $DS_{\#}$ is a dynamical system whose state space is the set of the natural numbers Z^+ or a finite initial segment of Z^+ ; (ii) u is an isomorphism of $DS_{\#}$ in DS ; (iii) all the state transitions of $DS_{\#}$ are recursive.

Giunti (2004) asks whether computational systems whose dynamics is not *intrinsically* recursive exist. Such a non-intrinsic computational system would admit two isomorphic numeric representations such that one is recursive and the other is not – i.e., one representation would satisfy all three conditions above, while the other one would only satisfy (i) and (ii). The quite surprising result is that even the discrete system generated by the successor function is of this kind (Giunti 2004, th. 3).

Giunti then argues that this counter-intuitive result depends on the fact that one of the two representations involved (namely, the non-recursive one) is not effective *with respect to the dynamics* of the system. By a *dynamically effective representation*³ of a discrete dynamical system DS he means a pair $(u, DS_{\#})$ such that: (i) $DS_{\#}$ is a dynamical system whose state space is the set of the natural numbers Z^+ or a finite initial segment of Z^+ ; (ii) u is an isomorphism of $DS_{\#}$ in DS (and thus, u is a bijective enumeration from the state space N of $DS_{\#}$ to the state space M of DS); (iii.a) the enumeration $u: N \rightarrow M$ can be constructed effectively by means of a mechanical procedure that takes as given the *whole structure* of the state space M , and *nothing more*.

Giunti also makes clear that, as it stands, the definition of a dynamically effective representation is not formally adequate, for a precise analysis of condition (iii.a) is needed. He then suggests that such a formal analysis can be given on the basis of (a) some general concepts of dynamical systems theory and graph theory and (b) the notion of an *enumerating machine*, i.e., a machine that effectively produces an enumeration of the state space by moving from state to state in accordance with the state transitions determined by the dynamics of the system, and by counting $0, 1, 2, \dots, n, \dots$ whenever it reaches a new state.

In sec. 2 of this paper, I will first provide the necessary formal framework, and I will then precisely define the intuitive idea of a dynamically effective representation of a discrete system *DS*.

In sec. 3, I will state and discuss the conjecture that any two dynamically effective representations of a discrete dynamical system are either both recursive or both non-recursive. Thus, if we only consider dynamically effective representations, the dynamics of any discrete system should turn out to be either intrinsically recursive or intrinsically non-recursive. The complete proof of the conjecture goes beyond the limits of this paper. However, I will give an informal argument that only presupposes the truth of Turing-Church thesis.⁴

In the last section (sec. 4), I will propose a revision in the explication of the concept of a computational system, which is suggested by the previous results. According to this revision, computational systems should no longer be identified with those dynamical systems that can be represented *recursively* but, rather, with those dynamical systems that can be represented in a *dynamically effective* way, that is to say, effectively with respect to the structure of their state space. Computational systems would then include discrete systems whose dynamically effective representations are all recursive (*intrinsically recursive* computational systems); in addition, the existence of discrete systems whose dynamically effective representations are all non-recursive (*intrinsically non-recursive* computational systems) is also possible.

A theory of computability that were based on this modified notion of a computational system might thus be called a theory of *intrinsic computability* and, just in case intrinsically non-recursive computational systems exist, it allows the computation of both recursive functions and non-recursive ones. The recursive functions are the functions computed by the intrinsically recursive computational systems; should the intrinsically non-recursive computational systems exist, they would compute non-recursive functions. Thus, according to this scenario, a particular version (MT) of Turing-Church thesis⁵ might either be proved or disproved.

In the final part of sec. 4, I will make clear how a proof/disproof of MT is possible, provided that the proposed explication of the intuitive concept of a computational system is a good one. In addition, I will contrast MT with other versions of Turing-Church thesis (Copeland 2002; Gandy 1980), and I will make explicit the possible relationships between my proposal and Gandy's axiomatization (1980) of the intuitive notion of a mechanism.

2. COMPUTATIONAL SYSTEMS AS EFFECTIVELY REPRESENTABLE DYNAMICAL SYSTEMS

A dynamical system is a mathematical model that expresses the idea of an arbitrary deterministic system, either reversible or irreversible, with discrete or continuous time or state space (Arnold 1977; Szlenk 1984). Let Z be the integers, Z^+ the non-negative integers, R the reals and R^+ the non-negative reals; below is the exact definition of a dynamical system.

- [1] DS is a dynamical system iff there is $M, T, (g^t)_{t \in T}$ such that $DS = (M, (g^t)_{t \in T})$ and
1. M is a non-empty set; M represents all the possible states of the system, and it is called the *state space*;
 2. T is either $Z, Z^+, R,$ or R^+ ; T represents the time of the system, and it is called the *time set*;⁶
 3. $(g^t)_{t \in T}$ is a family of functions from M to M ; each function g^t is called a *state transition* or a *t -advance* of the system;
 4. for any $t, v \in T$, for any $x \in M$,
 - a. $g^0(x) = x$;
 - b. $g^{t+v}(x) = g^v(g^t(x))$.

[2] A *discrete*⁷ dynamical system is a dynamical system whose state space is finite or denumerable,⁸ and whose time set is either Z or Z^+ ; [3] a *continuous dynamical system* is a dynamical system that is not discrete; [4] a *cascade* is a dynamical system with discrete time, i.e., whose time set is either Z or Z^+ . Thus, all discrete dynamical systems are cascades, but the reverse is not true.

[5] A dynamical system is *reversible* iff its time set is either Z or R ; [6] it is *irreversible* iff its time set is either Z^+ or R^+ . If a dynamical system DS is reversible, then any state transition is bijective, and the set of all state transitions $\{g^t\}_{t \in T}$ is a commutative group with respect to the composition of functions; the unit is g^0 and, for any $t \in T$, the algebraic inverse of g^t is $g^{-t} =$ the inverse function $(g^t)^{-1}$. If DS is irreversible, $\{g^t\}_{t \in T}$ is a commutative monoid with respect to the composition operation, with unit g^0 .

Any t -advance ($t > 0$) of an irreversible cascade can always be thought as being generated by iterating t times a given function $g: M \rightarrow M$ (thus, $g^1 = g$). As for a reversible cascade, the generating function $g: M \rightarrow M$ must be bijective; the positive t -advances are then obtained as before, while the negative ones ($t < 0$) are generated by iterating $|t|$ times the inverse function g^{-1} .

[7] $DS = (M, (g^t)_{t \in T})$ is a *possible dynamical system* iff DS satisfies the first three conditions of def. 1. We can now define the concept of an isomorphism between two possible dynamical systems as follows. [8] u is an *isomorphism of DS_1 in DS_2* iff $DS_1 = (M, (g^t)_{t \in T})$ and $DS_2 = (N, (h^v)_{v \in V})$ are possible dynamical systems, $T = V$, $u: M \rightarrow N$ is a bijection and, for any $t \in T$, for any $x \in M$, $u(g^t(x)) = h^t(u(x))$.

[9] DS_1 is *isomorphic to DS_2* iff there is u such that u is an isomorphism of DS_1 in DS_2 . It is easy to verify that the isomorphism relation is an equivalence relation on any given set of possible dynamical systems. (The concept of *set of all possible dynamical systems* is inconsistent, and we must then take as the basis of the theory of dynamical systems a specific, sufficiently large, set of possible dynamical systems.)

It is also not difficult to prove that the relation of isomorphism is a congruence with respect to the property of being a dynamical system, that is to say: if DS_1 is isomorphic to DS_2 and DS_1 is a dynamical system, then DS_2 is a dynamical system. This allows us to speak of abstract dynamical systems in exactly the same sense we talk of abstract groups, fields, lattices, order structures, etc. We can thus define: [10] an *abstract dynamical system* is any equivalence class of isomorphic dynamical systems.

[11] A *recursive representation* of a discrete dynamical system DS is a pair $(u, DS_\#)$ such that: (i) $DS_\#$ is a dynamical system whose state space is the set of the natural numbers Z^+ or a finite initial segment of Z^+ ; (ii) u is an isomorphism of $DS_\#$ in DS ; (iii) all the state transitions of $DS_\#$ are recursive.

Note that, since any discrete system is a cascade, condition (iii) of def. 11 reduces to the following condition. Let $g_\#$ be the generating function of the positive state transitions of $DS_\#$. Then, if DS is irreversible, (iii) is equivalent to requiring that $g_\#$ be recursive; if DS is reversible, (iii) is equivalent to requiring that both $g_\#$ and its inverse $g_\#^{-1}$ be recursive. However, by condition (i), the domain of $g_\#$ is a recursively enumerable subset of Z^+ ; this, together with the recursivity of $g_\#$, entails that $g_\#^{-1}$ is recursive too. Therefore, condition (iii) is equivalent to requiring that $g_\#$ be recursive.

Giunti's definition (1992, 1997) of a computational system is equivalent to the following:⁹ [12] DS is a *computational₁ system* iff DS is a discrete dynamical system, and there is a recursive representation of DS .

Giunti's pre-analytic definition (2004) of a dynamically effective representation¹⁰ is as follows: [13] a *dynamically effective representation* of a discrete dynamical system DS is a pair $(u, DS_\#)$ such that: (i) $DS_\#$ is a dynamical system whose state space is the set of the natural numbers Z^+ or a finite initial segment of Z^+ ; (ii) u is an isomorphism of $DS_\#$ in DS (and thus, u is a bijective enumeration from the state space N of $DS_\#$ to the state space M of DS); (iii.a) the enumeration $u: N \rightarrow M$ can be constructed effectively

by means of a mechanical procedure that takes as given the *whole structure* of the state space M , and *nothing more*.

As it stands, def. 13 is not formally adequate, for a precise analysis of condition (iii.a) is needed. I will carry out this analysis in sec. 2.3. However, it is convenient to introduce before-hand a few more general concepts¹¹ of dynamical systems theory.

2.1 Decomposable vs. undecomposable systems

If $DS = (M, (g^t)_{t \in T})$ is a dynamical system, [14] X is an *invariant set* of M (or a *subspace* of M) iff $X \subseteq M$, $X \neq \emptyset$ and, for any $x \in X$, for any $t \in T$, $g^t(x) \in X$. For any $X \subseteq M$ and $t \in T$, let g^t/X be the restriction of g^t to X . It is then clear that $(X, (g^t/X)_{t \in T})$ is a dynamical system iff X is an invariant set of M ; [15] if $(X, (g^t/X)_{t \in T})$ satisfies such a condition, it is called a *subsystem* of DS . Obviously, any dynamical system is a subsystem of itself. [16] A dynamical system is *simple* iff it doesn't have any subsystem but itself. For example, any dynamical system of the form $(\{x\}, (i^t)_{t \in T})$, where i is the identity function, is simple.

For any $X \subseteq M$, [17] the *future* of X is $F(X) = \{y: \text{for some } x \in X, \text{ for some } t > 0, g^t(x) = y\}$, and [18] the *past* of X is $P(X) = \{y: \text{for some } x \in X, \text{ for some } t > 0, g^t(y) = x\}$. If DS is reversible, by def. 14, any invariant set X contains both its future and its past. However, if DS is irreversible, def. 14 only ensures that X contains its future, but not its past. We then say that [19] X is a (temporally) *complete invariant set* of M (or a *complete subspace* of M) iff X is an invariant set of M and $P(X) \subseteq X$. Thus, by def. 19 and 14, any subspace of a reversible dynamical system is complete.

[20] X is a (temporally) *connected set* of M iff $X \subseteq M$ and, for any $a, b \in X$, there is $t, v \in T$ such that $g^t(a) = g^v(b)$. Thus, for any two different states a and b of a connected set X , four cases are possible: (i) $g^t(a) = b$ (if $v = 0$); (ii) $a = g^v(b)$ (if $t = 0$); (iii) $g^t(a) = g^v(b)$ (if $t = v$); (iv) $g^t(a) = g^v(b)$ (if $t \neq v$). That is to say, any two different states of a connected set are such that either one is in the future of the other, or there is a third state that is in the future of both. (Of course, we could rephrase this observation in terms of the past, instead of the future.)

Let X be a connected and complete invariant set of M . It is then easy to verify that, with respect to this property, X is both minimal and maximal.¹² Also note that, by def. 20 and 19, any two connected and complete invariant sets of M are either identical or disjoint; furthermore, the union of all such sets is identical to M . It thus follows that the set of all connected and complete invariant sets of M is a partition of M . [21] Any connected and complete invariant set of M is called a *constituent set* of M , and the set of all constituent sets of M is indicated by C_M ; C_M is also called the *decomposition* of M into its constituent sets.

Let $DS = (M, (g^t)_{t \in T})$ be an arbitrary dynamical system. For any constituent set $X \in C_M$, consider the subsystem $(X, (g^t/X)_{t \in T})$; [22] any such subsystem of DS is called a *constituent* (system) of DS , and the family of all constituents of DS is indicated by $(C_X)_{X \in C_M}$; $(C_X)_{X \in C_M}$ is also called the *decomposition* of DS into its constituent subsystems.

Note that a constituent of DS not necessarily is simple. However, by the minimality of its state space, it follows that it does not have any constituent but itself. [23] A dynamical system is *undecomposable* iff it is its only constituent. Thus, in particular, any constituent of any dynamical system is undecomposable. [24] A dynamical system is *decomposable* iff it is not undecomposable.

Example 1 [undecomposable and decomposable systems]

1. Let s be the successor function; $(Z^+, (s^n)_{n \in Z^+})$ is undecomposable.
2. Let i be the identity function; $(Z^+, (i^n)_{n \in Z^+})$ is decomposable.

[25] Two dynamical systems are *disjoint* iff the intersection between their state spaces is empty, and they have the same time set. Thus, any two constituents of any dynamical system are disjoint. Two disjoint dynamical systems, $DS_1 = (M_1, (g_1^t)_{t \in T})$ and $DS_2 = (M_2, (g_2^t)_{t \in T})$, can always be composed by means of the *composition operation* \oplus , which is defined as follows: [26] $(DS_1 \oplus DS_2) = (M_1 \cup M_2, (g_1^t \oplus g_2^t)_{t \in T})$ where, for any $t \in T$ and $x \in M_1 \cup M_2$, $g_1^t \oplus g_2^t(x) = g_i^t(x)$ such that $x \in M_i$ ($i = 1$ or 2). It is immediate to verify that $(DS_1 \oplus DS_2)$ is a dynamical system, and that \oplus is commutative. The composition operation can be generalized to an arbitrary family¹³ $(DS_i)_{i \in I} = (M_i, (g_i^t)_{t \in T})_{i \in I}$ of mutually disjoint dynamical systems as follows: [27] $\Sigma(DS_i)_{i \in I} = (\cup(M_i)_{i \in I}, (\Sigma(g_i^t)_{i \in I})_{t \in T})$ where, for any $t \in T$ and $x \in \cup(M_i)_{i \in I}$, $\Sigma(g_i^t)_{i \in I}(x) = g_i^t(x)$ such that $x \in M_i$.

Let $(C_X)_{X \in C_M}$ be the family of all constituents of an arbitrary dynamical system $DS = (M, (g^t)_{t \in T})$. Then, by def. 27 and def. 22, $\Sigma(C_X)_{X \in C_M} = DS$. Thus, we have just shown:

Theorem 1 [decomposition theorem]

Any dynamical system is identical to the composition of all its constituents.

Proof

See the text above.

Q.E.D.

Henceforth, I will only consider *undecomposable* dynamical systems. We can do this without any loss of generality because, by the result just stated, any decomposable dynamical system can always be thought as the (infinite) composition of its undecomposable constituents.

2.2 Connected state-space graphs

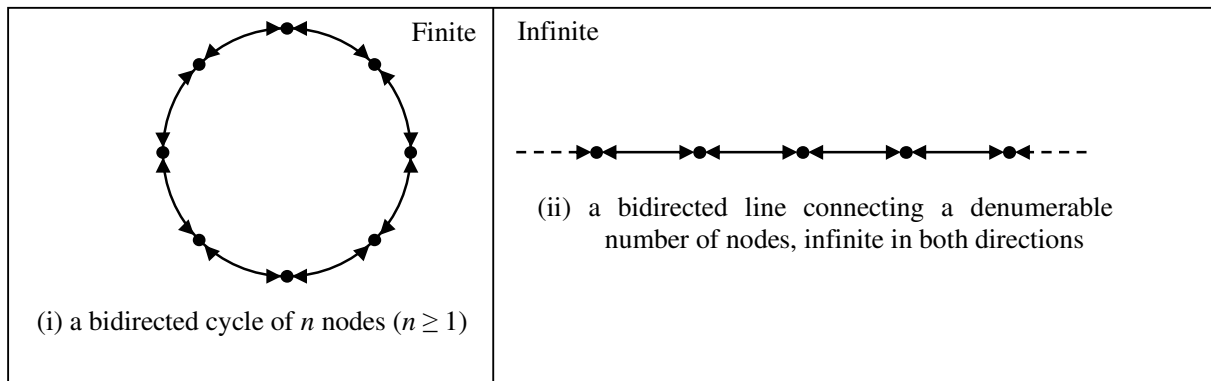
What does the state-space structure of an undecomposable discrete dynamical system look like? In general, since the positive t -advances of any discrete dynamical system $DS = (M, (g^t)_{t \in \mathbb{T}})$ are generated by a function $g: M \rightarrow M$, the structure of M can always be thought as a finite or denumerable *directed graph*, whose nodes are the states of M , and whose arrows (i.e., the directed edges of the graph) are determined by g . Note that, since g is a function, the graph has no *outward fork*, i.e., there is no node with two or more departing arrows; on the other hand, *inward forks* (i.e., nodes with two or more incoming arrows) are present iff g is not injective. If DS is irreversible, this directed graph represents the *whole* structure of M .

If DS is reversible, however, the directed graph determined by g only represents the structure of M fixed by the positive t -advances (the *forward structure* of M). The *backward structure* of M , which is given by the negative t -advances, is generated by the inverse function $g^{-1}: M \rightarrow M$. We then must add to the graph a new set of arrows, corresponding to g^{-1} . However, since g^{-1} is the inverse of g , each of the g^{-1} -arrows is just the inverse of one of the g -arrows. The resulting graph is thus a *bidirected graph*, i.e., a graph whose edges are all bidirectional arrows. Also note that, since g is bijective, the graph has no inward fork either, and thus no fork whatsoever, both with respect to the g -arrows and to the g^{-1} -arrows.

If DS is undecomposable, its state space is connected. Therefore, for a reversible, undecomposable, discrete dynamical system, there are only two possible types of state-space graphs: if DS is finite, (i) the graph is a bidirected cycle of n nodes ($n \geq 1$); if DS is denumerable, (ii) the graph is a bidirected line connecting a denumerable number of nodes, infinite in both directions (see figure 1).

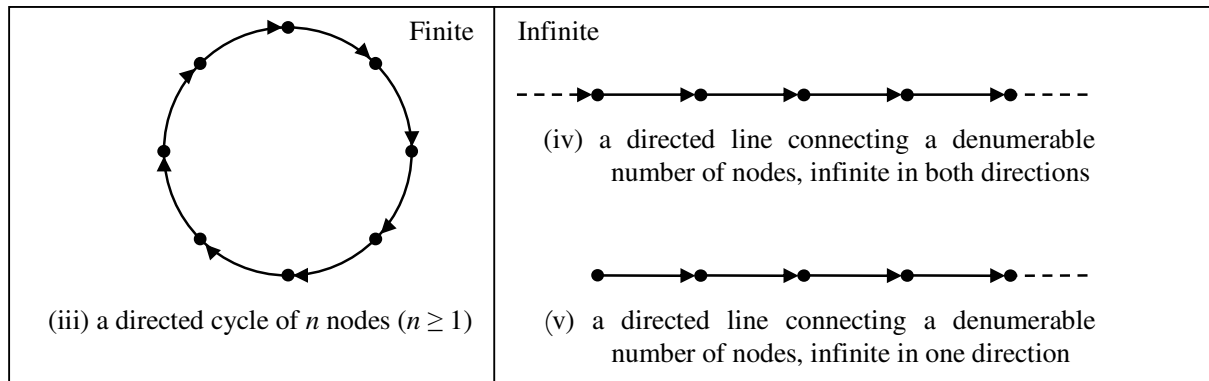
FIGURE 1

Reversible Systems



If DS is irreversible, but g is injective (i.e., if DS is *logically reversible*), the two previous kinds of state-space graphs are still possible. The only difference is that now they are *directed* graphs. If DS is denumerable, however, also a third kind of state-space graph is possible; thus: (iii) if DS is finite, the graph is a directed cycle of n nodes ($n \geq 1$); if DS is denumerable, either (iv) the graph is a directed line connecting a denumerable number of nodes, infinite in both directions, or (v) it is a directed line connecting a denumerable number of nodes, infinite in *one* direction (see figure 2). The latter kind of connected graph corresponds to any denumerable undecomposable discrete system generated by an injective but not surjective function. Note that this is the kind of graph of the system $(Z^+, (s^n)_{n \in Z^+})$ generated by the successor function s , for s is injective but not surjective.

FIGURE 2 Logically Reversible Systems



If DS is irreversible and g is not injective (i.e., if DS is *logically irreversible*) things get much more complex, because the state-space directed graph now has some inward fork. It is convenient to distinguish between two different kinds of inward forks, depending on whether the fork is either (a) a periodic point¹⁴ (*periodic fork*), or (b) it is not (*non-periodic fork*). In addition, let us also distinguish between (c) a *finite* fork (with a finite number of incoming arrows), and (d) an *infinite* fork (with an infinite number of incoming arrows). Then, by combining these two pairs of concepts, we get four mutually exclusive and exhaustive different kinds of inward forks.

Suppose first that DS is finite. Then, all inward forks are finite and, since the state space M is connected, its graph is one of the possible outcomes of the following procedure. Let us call a node with no departing arrow *tail-free*, a node with one or more incoming arrows *fork-free*, and a node with no incoming arrow *head-free*. The procedure starts from a graph containing just one node and, at each successive step, adds one arrow to the

graph, by choosing one of the following four operations: (1) head-operation: connect the head of an arrow to a head-free node; (2) fork-operation: connect the head of an arrow to a fork-free node; (3) tail-operation: connect the tail of an arrow to a tail-free node; (4) cycle-operation: connect the tail of an arrow to a tail-free node, and the head of the same arrow to either a head-free node or a fork-free one. The procedure must also satisfy the three following conditions: (α) it performs the cycle-operation exactly once; (β) it performs at least once either a fork-operation, or a cycle-operation such that the head of the arrow is connected to a fork-free node; (γ) it stops after n ($n \geq 2$) steps.

Condition (γ) ensures that the output is a finite graph, and the requirement that $n \geq 2$, in conjunction with condition (β), entails that it is a graph with at least one fork, i.e., the graph of a logically irreversible dynamical system. However, to understand why the cycle operation must be performed at least once, and not more than once, we need a few more considerations.

Note first that the starting node is tail-free. Thus, at step 0, the graph has 1 tail-free node. Second, the head-operation, the fork-operation and the tail operation all conserve the number of tail-free nodes, while the cycle-operation decreases of 1 the number of such nodes. Therefore, at any step of the procedure, the number of tail-free nodes is either 1 or 0. But, since to apply the cycle-operation we need a tail-free node, and after its application the number of such nodes decreases of 1, the cycle operation cannot be performed more than once. Finally, to understand why the cycle operation must be performed at least once, suppose the procedure stopped after n steps without having performed such operation. The output graph would then have 1 tail-free node, and thus it would not be the directed graph of any function $g: M \rightarrow M$.

Finally, note that the cycle-operation always produces either a simple cycle or a cycle with some periodic fork. Since the cycle-operation is performed exactly once, and the state-space graph cannot contain a simple cycle (for DS is logically irreversible), the general form of the state-space graph of a finite, undecomposable, logically irreversible system consists of (vi) a directed cycle to which the roots of a finite number of finite trees are attached (see figure 3). An interesting subcase is that of a directed cycle with *just one line* attached to it. An undecomposable discrete dynamical system with this special kind of state-space graph is *weakly irreversible*¹⁵ (see figure 4).

FIGURE 3 Finite Logically Irreversible Systems

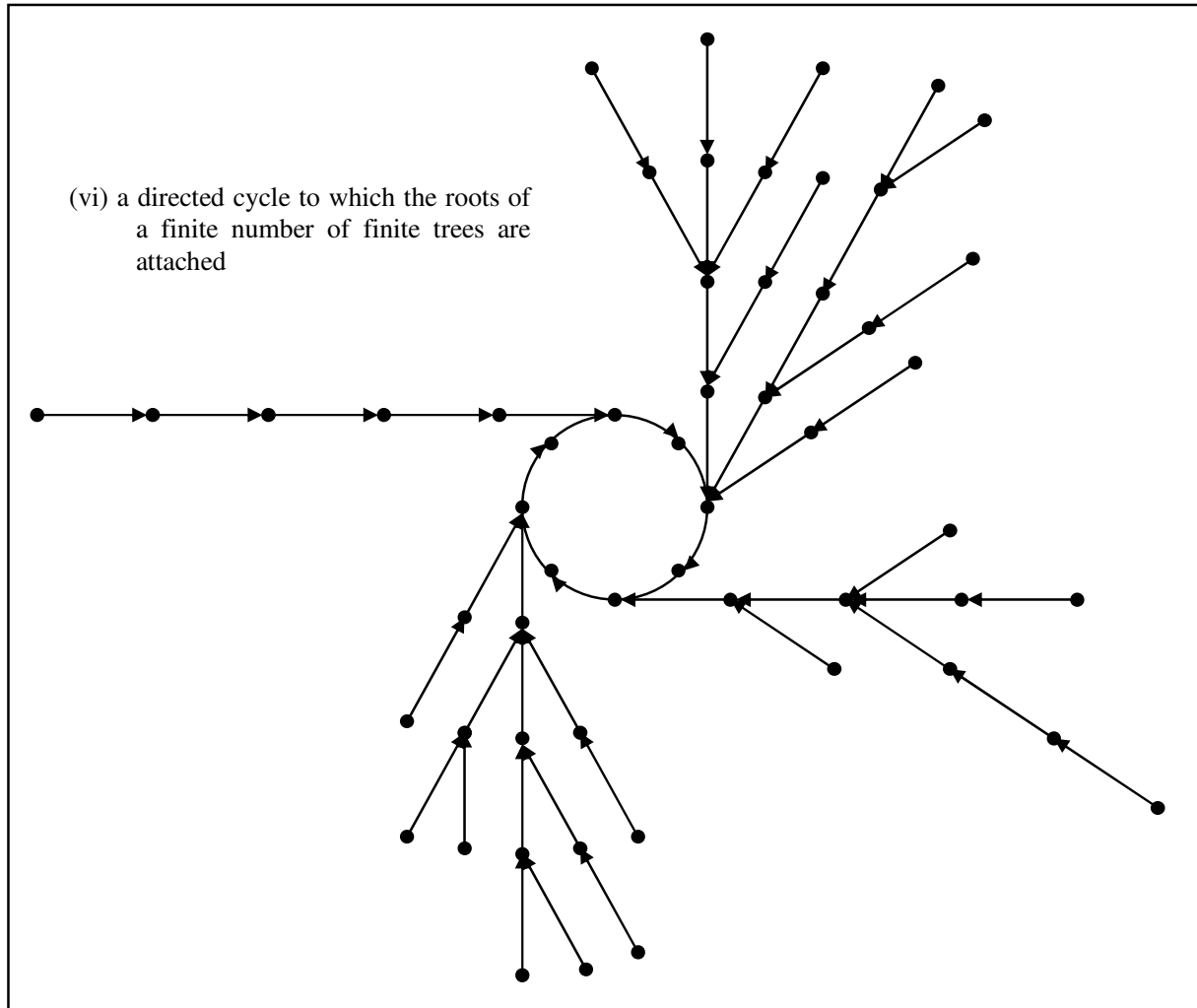
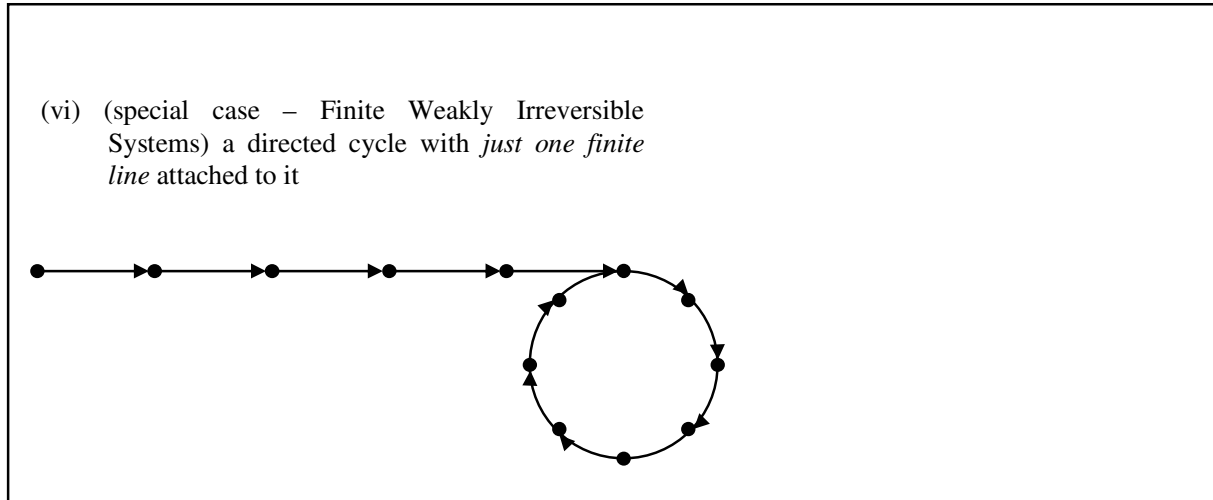


FIGURE 4 Finite Logically Irreversible Systems



Suppose now that DS is logically irreversible and denumerable. Therefore, the state-space graph is infinite; however, since the state space is connected, its graph can still be thought as being generated by an *indefinite* application of the procedure described above. Thus, since the procedure must be applied indefinitely, condition (γ) must be dropped. Also, it is no longer necessary that the cycle-operation be performed at least once, for infinitely many applications of the tail-operation also ensure that no node is tail-free. The generating procedure, then, must satisfy the following modified conditions: (α') it performs either the cycle-operation *at most* once, or a tail-operation infinitely many times; (β) it performs at least once either a fork operation, or a cycle-operation such that the head of the arrow is connected to a fork-free node; (γ') it *never* stops.

If the procedure performs the cycle-operation, the general form of the state-space graph (see figure 5) is very similar to the previous finite case, namely: (vii) a directed cycle to which the roots of a finite number of trees are attached; in this case, however, at least one of the trees is *infinite*, either with respect to the number of levels (*infinite height*), or to the number of nodes in some level (*infinite thickness*). Note that a tree has infinite thickness iff the state-space graph has at least one infinite fork (either periodic or non-periodic). Also, analogously to the previous finite case, the state-space graph of a denumerable, undecomposable, weakly irreversible system has the special form of a directed cycle with *just one infinite line* attached to it (see figure 6).

FIGURE 5 Infinite Logically Irreversible Systems

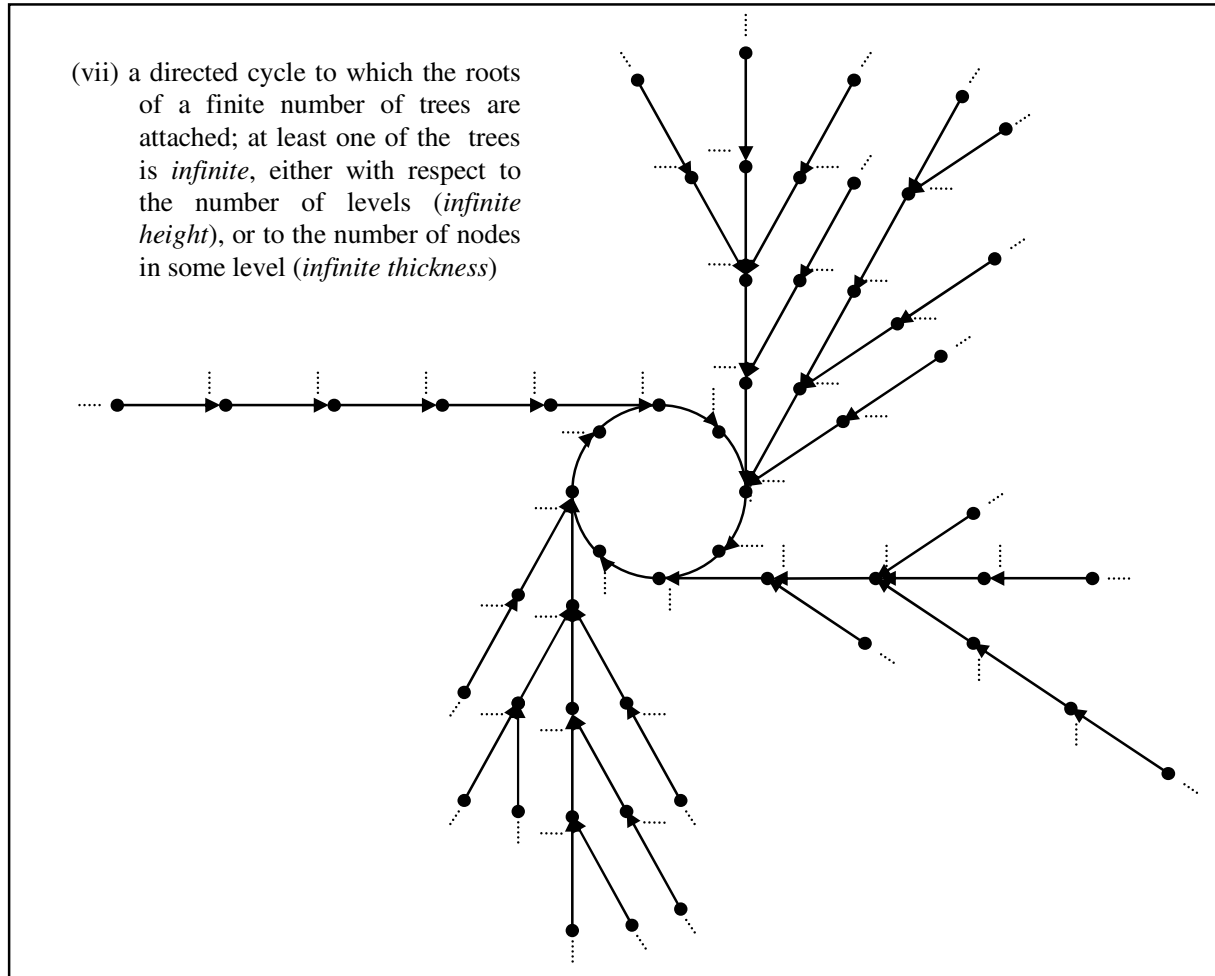
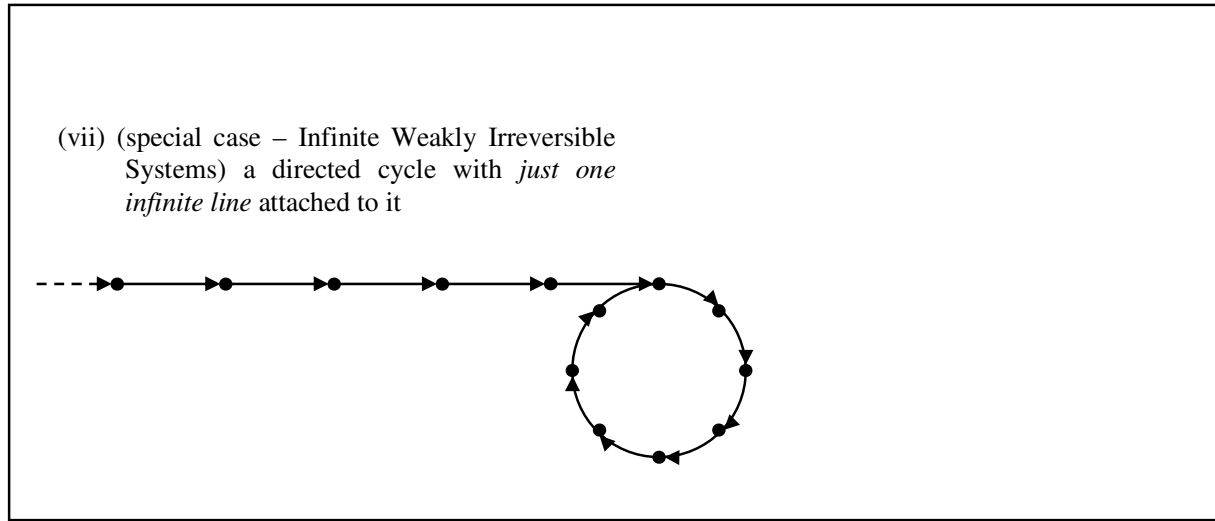


FIGURE 6 Infinite Logically Irreversible Systems



If the procedure never performs the cycle operation, but it performs instead the tail-operation infinitely many times, we can distinguish two further cases, according to whether or not the line produced by the tail-operations has at least one infinite incoming line. If it does not, the general form of the state-space graph is (viii) a directed line infinite in one direction, to which the roots of a possibly infinite number of trees of finite height (but possibly infinite thickness) are attached (see figure 7); in the second case, the form is (ix) a directed line infinite in two directions, to which the roots of a possibly infinite number of possibly infinite trees (in either height or thickness) are attached (see figure 8).

FIGURE 7 Infinite Logically Irreversible Systems

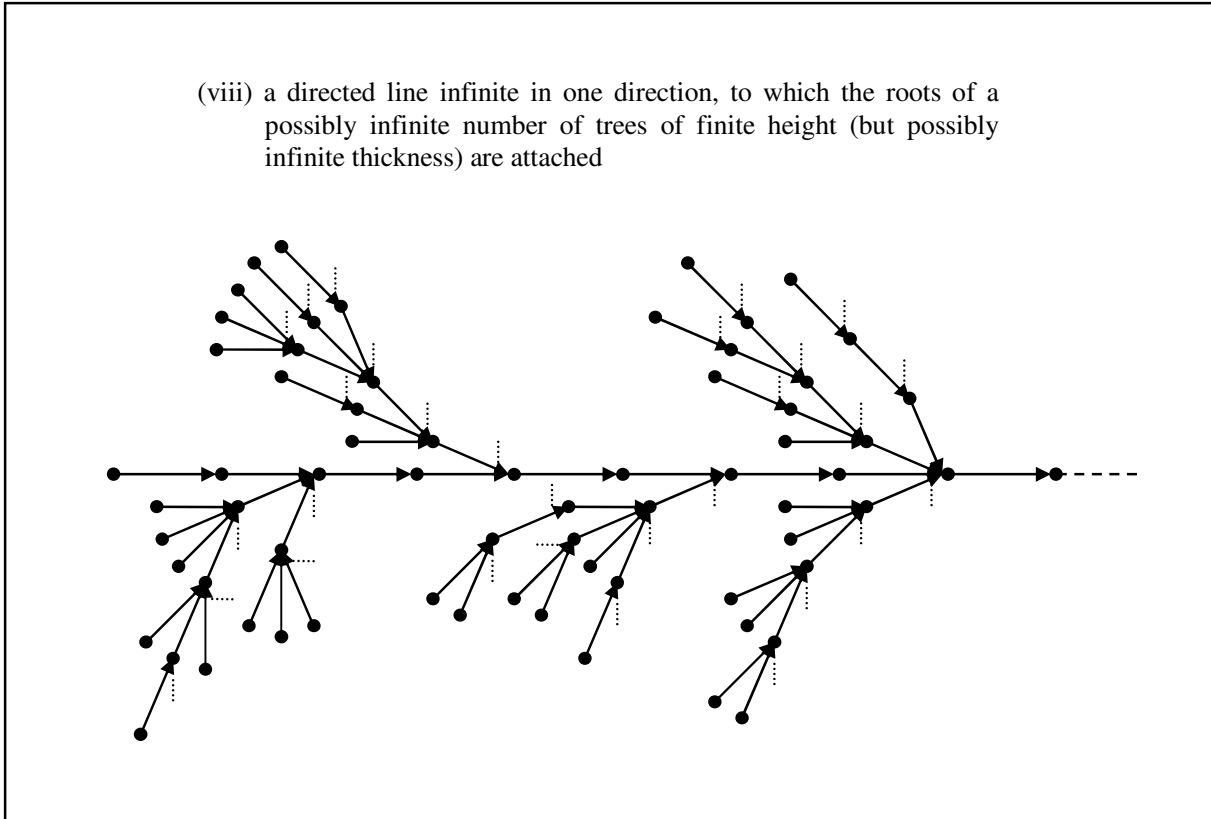
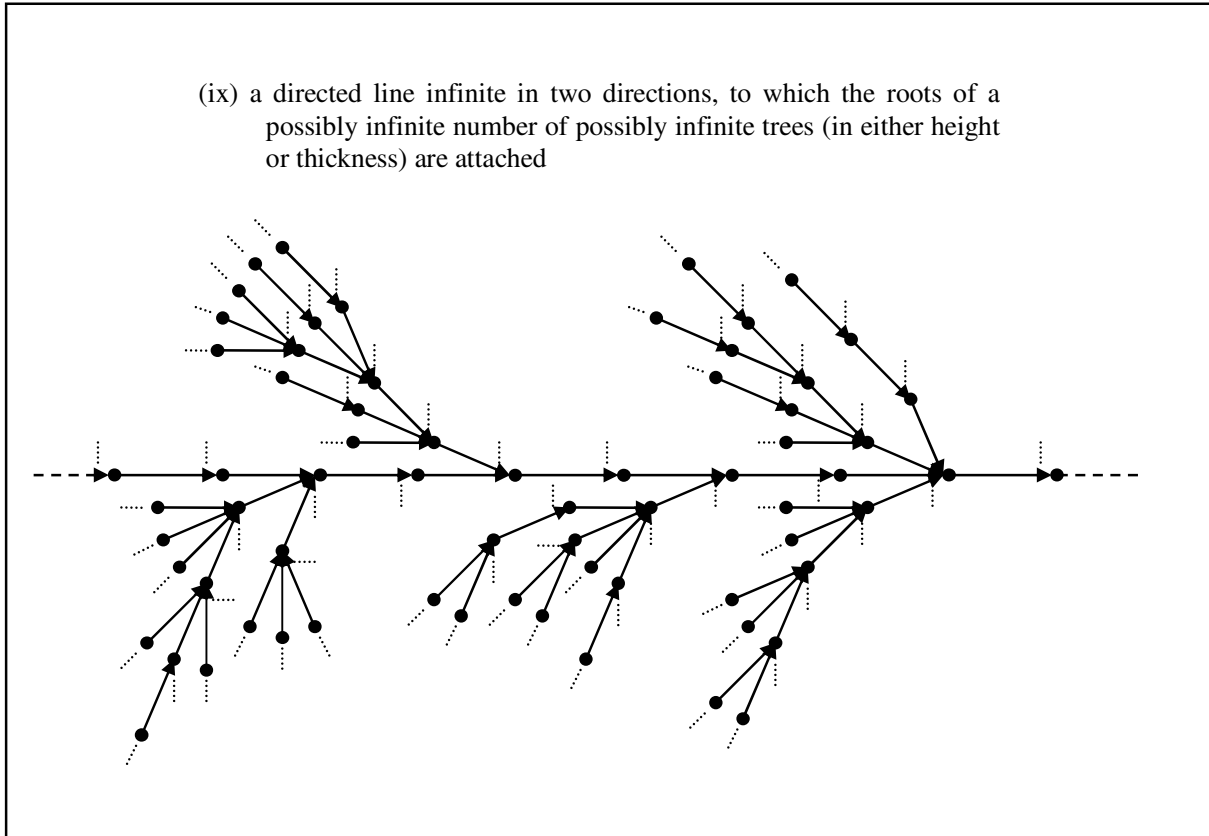


FIGURE 8 Infinite Logically Irreversible Systems



In conclusion, the foregoing analysis can be summarized as follows:

Theorem 2 [classification of connected state-space graphs]

The state-space graph of an arbitrary undecomposable discrete dynamical system is of one of the forms (i) – (ix). In particular (i), (iii) and (vi) are the possible general forms of the graph of a *finite* undecomposable discrete system; (ii), (iv), (v), (vii), (viii) and (ix) are the possible forms of the graph of a *denumerable* undecomposable discrete system.

Proof

See the text above.

Q.E.D.

2.3 Enumerating machines

Let $DS = (M, (g^t)_{t \in T})$ be an arbitrary, *undecomposable*, discrete, dynamical system. We are now sufficiently equipped to attempt a precise definition of the idea of an enumeration $u: Z^+ \rightarrow M$ that is effective with respect to the structure of the state space M . (If M is finite, u is an enumeration from an initial segment of Z^+ to M .)

We have just seen (sec. 2.2) that the state-space structure of any undecomposable discrete dynamical system can be identified with a special kind of *connected graph*, which can assume nine different types of general forms. We have also seen that any such graph is generated by a mechanical procedure that, at each step, applies one of four operations. Of these, only the cycle operation does not add a new node to the graph. Therefore, the generation procedure naturally induces a bijective enumeration of all the nodes of the graph, which is obtained by associating the starting node to 0, and then counting 1, 2, ... n , ... whenever a new node is added. This enumeration can thus be thought to be effective with respect to the state-space structure of the system.

More generally, an enumeration effective with respect to the state-space structure can be thought as an enumeration that is obtained by (1) starting from an arbitrary node and counting 0, then (2) moving back and forth along the edges of the graph and counting 1, 2, ... n , ... whenever we reach a new node, in such a way that (3) in our wandering, we will visit each node at least once.

I will now describe a kind of ideal machine that operates on an arbitrary state-space connected graph, and which produces an enumeration of its nodes just in the way specified by (1), (2) and (3). I call a machine of this kind an *enumerating machine* (on a state-space connected graph).

An enumerating machine can be thought as a device that, at any step of its operation, is located on exactly one node of the state-space graph, and which is capable of (i) writing a number on the node where it is located and (ii) moving from node to node by following the edges of the graph (the machine can go *back and forth* along any edge, even though the edge is directed). In addition, the machine is equipped with a finite map that reproduces exactly the part of the graph that it has already explored; at any step, the machine uses this map to perform both the writing and moving operations, and then (iii) it updates the map in order to keep track of the modified situation.

The initial map consists of just one point \bullet (which represents the initial node), together with the *pointer* \wedge , which indicates the current position of the machine. At any successive step, the map represents the part of the graph already explored by the machine. It is thus itself a graph, isomorphic to a connected finite portion of the state-space graph; in addition, it satisfies

the following conditions: (a) exactly one point of the map is marked by the pointer; this point represents the node where the machine is currently located, and it is called the *current point*; (b) all points, except at most the current point, are consecutively numbered; (c) some points (possibly none) are marked by a +, and some points (possibly none) are marked by a -; these points represent those nodes that have already been explored, but that must be visited again in the future, because they have at least one neighbor (*new neighbor*) that is not already represented in the map; these points are called the *active points*, and the corresponding nodes *active nodes*; the + and the - indicate the priorities of the future visits, in the sense that all the nodes represented by +points will be visited before any of the nodes represented by the -points.

An arbitrary cycle of the machine operation works as follows. In the first place, the machine checks whether the current point is numbered. If (α) the current point is not numbered, the machine writes on the current node the successor of the highest point-number in the map; if no point in the map has a number (i.e., if the map is the initial map) the machine writes 0. If (β) the current point is numbered, the machine does not write anything.

The machine then performs a movement on the state-space graph in the following way. If case (α) applies, the machine goes to a node that, in the map, is represented by a +point; if there is no +point, the machine goes to a node that, in the map, is represented by a -point; if there is no -point either, the machine goes to the first new neighbor¹⁶ of the current node; if there are no new neighbors, the machine does not move. If case (β) applies, the machine goes to the first new neighbor of the current node; if there are no new neighbors, the machine does not move.

The machine now updates the map to keep track of both the writing and moving operation it has performed. If the machine has written a number, the machine writes the same number on the current point; otherwise, it does not write anything on the current point. (Note that the current point has not been updated yet, and it thus represents the position of the machine *before* the movement it has just performed.) This takes care of the writing operation.

As for the updating relative to the movement operation, there are two different cases to be considered, depending on whether the machine has gone to an active node or to a new neighbor. If the machine has gone to an active node, (i) if in the map there are only -points, all the -markers are replaced by +markers; (ii) if the node represented by the current point has new neighbors, the machine marks the current point with a -; otherwise, it does not place any marker next to the current point (and, eventually, it removes any previous marker); (iii) it then moves the pointer to the point in the map that represents the node where the machine has just gone (thus, the current point is now updated and, from now on, it represents again the current position of the machine, i.e., the current node).

If the machine has gone to a new neighbor, (i.a) if the node represented by the current point has new neighbors, the machine marks the current point with a $-$; otherwise, it does not place any marker next to the current point (and, eventually, it removes any previous marker); (ii.a) it then draws a new arrow that links the current point with a new point (this new point represents the new neighbor where the machine has gone); (iii.a) it then moves the pointer to this new point (thus, the current point is now updated and, from now on, it represents again the current position of the machine, i.e., the current node).

Note that, after the machine has completed its updating, it has a new map which is isomorphic to the portion of the state-space graph it has already explored, indicates its current position, and keeps track of all the active nodes, i.e., those nodes that must be visited again because they have new neighbors that have not been counted yet. The rules of the machine make sure that each active node will be visited again in the future, and that, at each subsequent visit, its first new neighbor will be taken into account. Also, whenever a new node is reached, it is counted and, if it is an active node, it is then inserted in the $-$ -list, and thus visited again later on. Since the state-space graph is connected, all this entails that the machine will reach, sooner or later, all the nodes. Furthermore, since each node is counted only once (when it is reached for the first time), the machine will produce a bijective enumeration of the state-space. We can thus sum up this observation in the result below.

Theorem 3 [enumeration theorem]

For any undecomposable discrete dynamical system DS , if an enumerating machine is started on an arbitrary node of the state-space graph of DS , the machine produces a bijective enumeration of the nodes (and, consequently, of the states of DS).

Proof

See the text above.

Q.E.D.

Finally, the preceding theorem allows us to give a formally adequate version of definition [13]. Thus, [13A] a *dynamically effective representation* of an undecomposable discrete dynamical system DS is a pair $(u, DS_{\#})$ such that: (I) $DS_{\#}$ is a dynamical system whose state space is the set of the natural numbers Z^+ or a finite initial segment of Z^+ ; (II) u is an isomorphism of $DS_{\#}$ in DS (and thus, u is a bijective enumeration from the state space N of $DS_{\#}$ to the state space M of DS); (III.A) the enumeration $u: N \rightarrow M$ is produced by an enumerating machine that is started on some node of the state-space graph of DS .

A straightforward consequence of th. 3 and def. 13A is that any undecomposable discrete dynamical system DS has a dynamically effective representation. Let $u: N \rightarrow M$ be the bijective enumeration produced by an enumerating machine that is started on some node of the state-space graph of DS . Let $g_{\#}: N \rightarrow N$ such that, for any $m \in N$, $g_{\#}(m) = u^{-1}(g(u(m)))$, where $g = g^1$ is the generating function of DS . Let $DS_{\#} = (N, (g_{\#}^t)_{t \in T})$ be the discrete dynamical system generated by $g_{\#}$. Then, by construction, $(u, DS_{\#})$ is a dynamically effective representation of DS .

3. DYNAMICALLY EFFECTIVE REPRESENTATIONS PRESERVE RECURSIVITY

Giunti (2004, th.3, thesis 2) showed that the undecomposable discrete system generated by the successor function admits two numeric isomorphic representations such that one is recursive and the other is not. Thus, recursivity is not preserved by numeric representations. What happens with respect to dynamically effective representations? I will give below an informal argument to support the conjecture that any two dynamically effective representations of an arbitrary undecomposable discrete system are either recursive or non-recursive.

This argument is not a formal proof, because it relies on intuitive considerations about the effectivity of some procedures, and it also explicitly assumes the truth of Turing-Church thesis.¹⁷ It is not difficult to see how, by a sort of gödelization, the basic idea of the argument could be transformed into an actual proof. However, such a proof goes beyond the limits of this paper.

Conjecture 1 [dynamically effective representations preserve recursivity]

For any undecomposable discrete dynamical system DS , if some dynamically effective representation of DS is recursive, then any dynamically effective representation of DS is recursive.

Argument

Let $DS = (M, (g^t)_{t \in T})$ be an arbitrary undecomposable discrete dynamical system. Let $(u, DS_{\#})$ be a dynamically effective and recursive representation of DS , and let g_u be the generating function of $DS_{\#}$. Let (v, DS_*) be an arbitrary dynamically effective representation of DS , and let g_v be the generating function of DS_* . Then, $DS_{\#}$ and DS_* have identical state space (either Z^+ , or an initial segment of Z^+), which we indicate by $Z_{\#}$; in addition, $g_u = u \circ g \circ u^{-1}$ and $g_v = v \circ g \circ v^{-1}$. To prove the conjecture, it suffice to show that g_v is recursive. Note (see figure 9) that $g_v = (v \circ u^{-1}) \circ g_u \circ (u \circ v^{-1})$ and, by hypothesis, g_u is recursive. I will give an informal argument to show that both $(v \circ u^{-1})$ and $(u \circ v^{-1})$ are recursive,

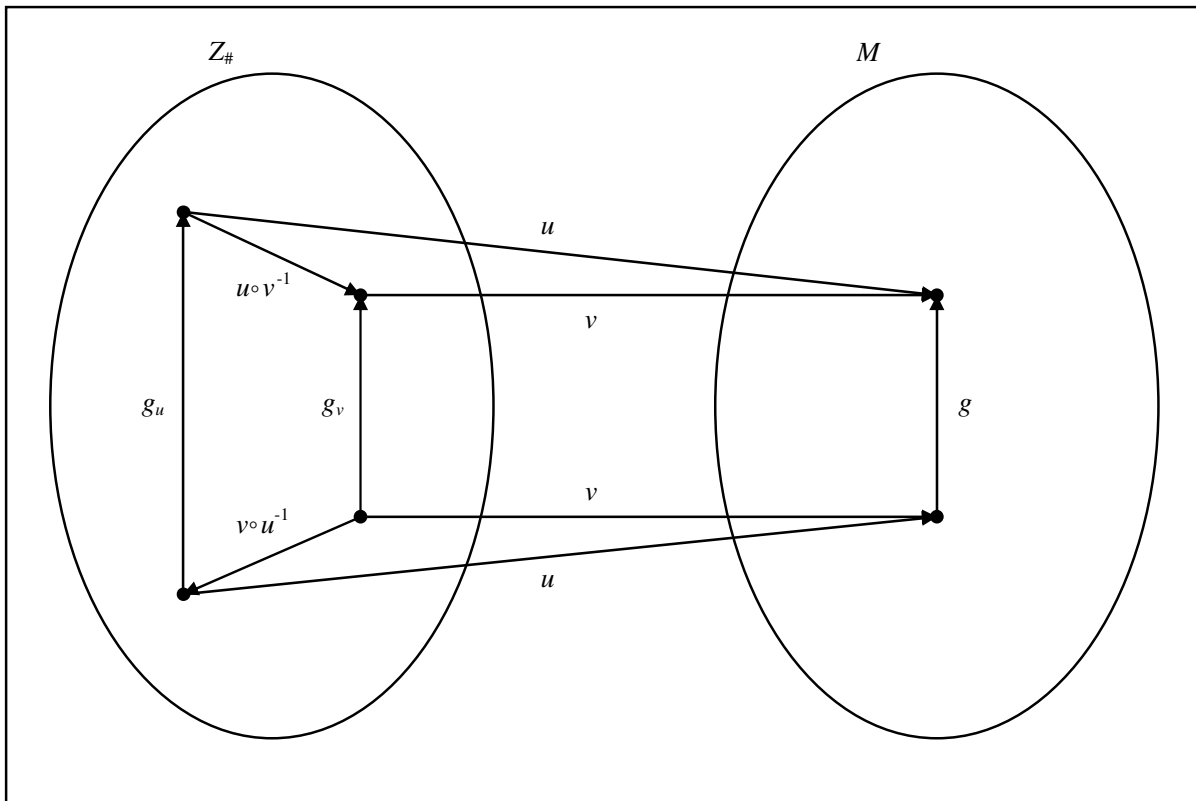
and thus g_v is recursive as well.

Since both $(u, DS_{\#})$ and (v, DS_*) are dynamically effective representations of DS , both u and v are enumerations of M produced by an enumerating machine that operates on the state-space graph of DS . The procedure executed by any such machine is clearly effective, except for the hypothesis that the *whole* state-space graph of DS is *given*. Note, however, that the state-space graph of DS is isomorphic to the state-space graph of $DS_{\#}$, and thus the two graphs can be identified; moreover, since g_u is recursive, there is an effective procedure for generating the state space-graph of $DS_{\#}$, and thus for generating the state-space graph of DS as well. It then follows that both u and v are bijective enumerations of M produced by a clearly effective procedure. Therefore, also their inverses u^{-1} and v^{-1} are effective. Hence, $(v \circ u^{-1})$ and $(u \circ v^{-1})$ are effective as well. Since both $(v \circ u^{-1})$ and $(u \circ v^{-1})$ are numeric functions, by Turing-Church thesis, they are recursive.

Q.E.D.

FIGURE 9

Conjecture 1



Let me finally remark that the previous conjecture and def. 13A vindicate Giunti's intuitive analysis of the representations involved in the proof of th. 3, thesis 2 (2004). Let $DS_2 = (Z^+, (s^n)_{n \in Z^+})$ be the discrete dynamical system generated by the successor function s ; i be the identity function on Z^+ ; $p^*: Z^+ \rightarrow Z^+$ be any bijection such that $s_{p^*}: Z^+ \rightarrow Z^+$ (where, for any $m \in Z^+$, $s_{p^*}(m) = p^*(s(p^{*-1}(m)))$) is not recursive; $DS_{p^*} = (Z^+, (s_{p^*}^n)_{n \in Z^+})$ be the discrete dynamical system generated by s_{p^*} . By def. 13A, (i, DS_2) is a dynamically effective representation of DS_2 , and (p^*, DS_2) is a dynamically effective representation of DS_{p^*} , in accordance with Giunti's 2004 intuitive analysis. Furthermore, since DS_2 is generated by the successor function, both representations are recursive. On the other hand, the generating function of DS_{p^*} is s_{p^*} , which is not recursive. Therefore, by conjecture 1, (p^{*-1}, DS_{p^*}) is not a dynamically effective representation of DS_2 , and (i, DS_{p^*}) is not a dynamically effective representation of DS_{p^*} , as Giunti 2004 claims.

4. INTRINSIC COMPUTATIONAL SYSTEMS AND TURING-CHURCH THESIS

Intuitively, a computational system is a discrete dynamical system that admits an effective representation. According to Giunti's 1997 explication of this concept, a discrete dynamical system DS is *computational*₁ iff there is a recursive representation of DS (def. 12). However, Giunti 2004 makes clear that the dynamics of most *computational*₁ systems is not *intrinsically* recursive, for even the discrete system generated by the successor function admits two isomorphic numeric representations such that one is recursive and the other is not. In this paper, I have proposed a formal analysis of the concept of a dynamically effective representation (def. 13A), and I have then argued that dynamically effective representations preserve recursivity (conjecture 1).

These results suggest a natural revision in the explication of the intuitive concept of a computational system. Computational systems should no longer be identified with those dynamical systems that can be represented *recursively* but, rather, with those dynamical systems that can be represented in a *dynamically effective* way, that is to say, effectively with respect to the structure of their state space. According to this revision, I thus define: [28] an undecomposable discrete dynamical system DS is *computational*₂ iff there is a dynamically effective representation of DS . Note that, by def. 28, def. 13A and th. 3, the set of all *computational*₂ systems turns out to be identical to the set of all undecomposable discrete systems.¹⁸

But then, if conjecture 1 is true, all computational₂ systems would turn out to be intrinsic, for the existence of just one *recursive* dynamically effective representation entails that *any other* dynamically effective representation is recursive as well. On the other hand, it is also possible that there are undecomposable systems that admit *non-recursive* dynamically effective representations; if such systems exist, *all* their dynamically effective representations will be non-recursive as well.

Therefore, according to the revised notion of a computational system, there are two possible kinds of such a system: (a) *intrinsically recursive* computational₂ systems, i.e., those systems whose dynamically effective representations are all recursive, and (b) *intrinsically non-recursive* computational₂ systems, i.e., those systems whose dynamically effective representations are all non-recursive.

A theory of computability based on this new notion of a computational system may be called a theory of *intrinsic computability*. Such a theory will allow the computation of both recursive and non-recursive functions if, and only if, intrinsically non-recursive computational₂ systems exist. It is in fact obvious that intrinsically recursive computational₂ systems can only compute recursive functions; in addition, if intrinsically non-recursive computational₂ systems exist, they will definitely have the means to compute at least some (and perhaps all) non-recursive functions.

It is well known that Turing-Church thesis can be stated in many different ways. The most uncontroversial version is perhaps the following:

[HT] Any numeric function that, in a clearly intuitive sense, is effective for a human being is recursive. An equivalent formulation is:
A human being that works in a routine way just equipped with paper and pencil can only compute recursive functions.

I call HT the *human version* of Turing-Church thesis. HT refers to the computing capacities of a human being, and it is probably the version that better conveys Turing and Church's original intentions. Somewhat misleadingly, Gandy (1980) calls HT "Theorem T", and he maintains that Turing's analysis (1936) of the process of calculation by a human being gives a cogent argument in its support.

Copeland (2002) refers to HT as the "Church-Turing thesis properly so-called", and he stresses that this version is to be clearly distinguished from a different one, which instead concerns the computing capacities of an arbitrary *machine*. Copeland's formulation of the second version is:

Thesis M: Whatever can be calculated by a machine (working on finite data in accordance with a finite program of instructions) is Turing-machine-computable. (Copeland 2002)

He also distinguishes between two different interpretations of thesis M, according to whether by the term “machine” we intend a real world device or an abstract or notional one. According to Copeland 2002, “it is an open empirical question whether or not the narrow this-worldly version of thesis M is true”, while under the second interpretation thesis M is obviously false, for “it is straightforward to describe notional machines, or ‘hypercomputers’ ... that generate functions not Turing-machine-computable”.

The class of the computational systems (as intended in this paper and in Giunti 1992, 1995, 1996, 1997, 1998, 2004) is strictly included in the class of all abstract or notional machines. Thus, if we take the notional interpretation of thesis M and we limit its scope to computational systems, we obtain a third version of Turing-Church thesis, which I propose to call the *mathematical version*:

[MT] Any numeric function that can be computed by a computational system (in the intuitive sense, see the second paragraph of sec. 1, and Giunti 1997, ch. 1, sec. 1 and 3) is recursive.

Unlike the unrestricted notional interpretation of thesis M, MT is not obviously false, for all the systems studied so far by standard computation theory can only compute recursive functions. On the other hand, MT is not known to be true either, for, despite the current evidence in its support, we might nonetheless discover some discrete dynamical system that (i) admits an effective representation in a clear and uncontroversial sense (and thus is computational) but, nonetheless, (ii) is capable of computing non-recursive functions.

In fact, provided that *computational*₂ as defined above (def. 28) is a good explication for the intuitive concept of a computational system, we can even state a necessary and sufficient condition for the falsity of MT: MT is false iff the class of the intrinsically non-recursive *computational*₂ systems is not empty. At the moment, we don’t know whether or not this class has members. But note that this is the kind of question that can be settled by proof, in either the positive or negative sense. Should such a proof be given, the falsity/truth of MT would follow.

MT is very close, if not equivalent, to the intended meaning of Gandy’s formulation of thesis M: “What can be calculated by a machine is computable” (1980, 124). For, on the one hand, Gandy explicitly affirms that by “computable” he means “computable by a Turing machine” (1980, 123); and, on the other hand, Gandy’s remarks (1980, 124-126) on the intended meaning of “machine” make clear that, by this term, he means a discrete, deterministic dynamical system for which some form of effective representation is given.

Gandy develops an axiomatization of the intuitive concept of machine by using hereditarily finite sets, and he then proposes to identify the machines

of his thesis M with the systems that satisfy his axioms I-IV (1980, 126, thesis P). Finally, he proves that “what can be calculated by a device satisfying principles I-IV is computable” (1980, 126, 145).

Thus, if Gandy’s axiomatization and my def. 28 turned out to be equivalent, the class of the intrinsically non-recursive computational₂ systems would be empty; conversely, if intrinsically non-recursive computational₂ systems exist, Gandy’s and mine are not equivalent explications of the intuitive concept of a computational system; and, finally, also a third case is possible: intrinsically non-recursive computational systems do not exist, but Gandy’s axiomatization and my def. 28 are nonetheless non-equivalent.

¹ According to Giunti 1992, 1995, 1996, 1997, 1998 a discrete dynamical system is computational iff there is an isomorphic system (i) whose state space is a recursive subset of the non-negative integers and (ii) whose state transitions are all recursive. This definition is in fact equivalent to the one given in the text (Giunti 2004).

² Giunti 2004 distinguishes between *recursive* representations (def. 13) and *recursive canonic* representations (def. 15). Here, I will not adopt such a distinction, and by the term “recursive representation” I will always intend what Giunti 2004 calls a “recursive canonic representation”.

³ In Giunti 2004, def. 23, a *dynamically effective representation* is called an “intrinsic representation”.

⁴ The version of Turing-Church thesis assumed in the argument is the usual one: [HT] any numeric function that, in a clearly intuitive sense, is effective for a human being is recursive. See sec. 4 for a discussion of HT and other versions of Turing-Church thesis.

⁵ I call this form of Turing-Church thesis the *mathematical* version: [MT] any numeric function that can be computed by a computational system (in the intuitive sense) is recursive.

⁶ It is important to keep in mind that T is not a *bare* set, but rather, a set on which we implicitly assume the whole usual structure of, respectively, the (non-negative) integers or the (non-negative) reals. More precisely, we could say that T is the domain of a model $(T, (\sigma_i)_{i \in I})$ of, respectively, the theory of the (non-negative) integers or the theory of the (non-negative) reals.

⁷ The term “discrete dynamical system” is often used as a synonym for “cascade”, i.e., a dynamical system with discrete time; see for example Kulenovic and Merino 2002, Martelli 1999, and Sandefour 1990. My use of the term “discrete dynamical system” is in accordance with Turing 1950.

⁸ I use the term “denumerable” as a synonym for “countably infinite”; this is in accordance with Boolos and Jeffrey 1980, 1.

⁹ See note 1.

¹⁰ See note 3.

¹¹ The concepts I am going to introduce are most useful when applied to either discrete or logically irreversible dynamical systems. A *logically irreversible* dynamical system is an irreversible dynamical system such that at least one of its t -advances is not injective. Most computational systems are of this kind. Note, however, that there are both discrete and *continuous* logically irreversible systems.

-
- ¹² That is to say, if X is a connected complete invariant set of M , there is no $Y \subseteq M$ such that Y is either a proper subset or superset of X , and Y is also a connected and complete invariant set of M .
- ¹³ Since the composition operation is commutative, the index set of the family need not be linearly ordered.
- ¹⁴ A *periodic point* is a state x such that $g^t(x) = x$, for some $t > 0$. If time is discrete, for any periodic point x , the set $P = \{t: t > 0 \text{ and } g^t(x) = x\}$ has a minimum. Any $t \in P$ is called a *period* of x , and its minimum *the (least) period* of x . If the time set $T = \mathbb{R}$ or \mathbb{R}^+ , P may not have a minimum, and thus there may be periodic points with no definite least period; e.g., in the dynamical system $(\{x\}, (i^t)_{t \in T})$, where i is the identity function, x is periodic with *any* period $t > 0$.
- ¹⁵ A dynamical system is weakly irreversible iff it has eventually periodic orbits, but it does not have any merging orbit (Giunti 1997, ch. 1, th. 6). For an undecomposable discrete system, the condition that eventually periodic orbits exist is equivalent to the condition that the state-space graph has the form of a cycle with lines departing from it; and the condition that no merging orbit exists is equivalent to (i) the non-existence of non-periodic forks; (ii) the non-existence of periodic forks with three or more incoming arrows; (iii) the non-existence of more than one periodic fork (thus, to the fact that the cycle has *just one line* attached to it, and not any tree with different branches). Also note that the state-space graph of a weakly irreversible system can always be obtained without any application of the fork operation. Thus, if the state-space graph of an undecomposable discrete system cannot be obtained without applying the fork operation at least once, such a system is logically irreversible, but not weakly irreversible. A system of this kind is called *strongly irreversible*, and it is characterized by the existence of merging orbits.
- ¹⁶ We have seen that, since an arbitrary state-space graph can be thought to be generated by a mechanical procedure, each of the numbers $0, 1, \dots, n, \dots$ is associated to exactly one of its nodes. Among the new neighbors of node x , its *first new neighbor* is the one to which the smallest number is associated.
- ¹⁷ See note 4.
- ¹⁸ Def. 28 only applies to *undecomposable* discrete systems, for the concept of a dynamically effective representation is only defined for such systems. However, by th. 1, we can naturally extend the definition to an arbitrary discrete system as follows. A discrete dynamical system DS is *computational*₂ iff for any of its constituents, there is a dynamically effective representation. Thus, according to this extended definition, by def. 13A, th. 3 and def. 23, the set of all computational₂ systems turns out to be identical to the set of all discrete dynamical systems.

REFERENCES

- Arnold, Vladimir I. (1977), *Ordinary Differential Equations*. Cambridge: The MIT Press.
- Boolos, George S., and Richard C. Jeffrey (1980), *Computability and Logic*. 2d ed. Cambridge: Cambridge University Press.
- Copeland, B. Jack (2002), "The Church-Turing Thesis", in Edward N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy (Fall 2002 Edition)*. URL = <<http://plato.stanford.edu/archives/fall2002/entries/church-turing/>>.
- Kulenovic, Mustafa R. S., and Orlando Merino (2002), *Discrete Dynamical Systems and Difference Equations with Mathematica*. Boca Raton: Chapman & Hall/CRC.

-
- Gandy, Robin (1980), "Church's Thesis and Principles for Mechanism", in J. Barwise, H. J. Keisler and K. Kunen (eds.), *The Kleene Symposium*. Amsterdam: North Holland Publishing Company, 123-148.
- Giunti, Marco (1992), *Computers, Dynamical Systems, Phenomena and the Mind*. Ph.D. dissertation. Bloomington, IN: Indiana University.
- (1995), "Dynamical Models of Cognition", in Robert F. Port, and Timothy van Gelder (eds.), *Mind as Motion: Explorations in the Dynamics of Cognition*. Cambridge: The MIT Press, 549-571.
- (1996), "Beyond Computationalism", in Garrison W. Cottrel (ed.), *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. Mahwah, NJ: L. Erlbaum Associates, 71-75.
- (1997), *Computation, Dynamics, and Cognition*. New York: Oxford University Press.
- (1998), "Is Computationalism the Hard Core of Cognitive Science?", in Vito M. Abrusci, Carlo Cellucci, Roberto Cordeschi, and Vincenzo Fano (eds.), *Prospettive della logica e della filosofia della scienza: Atti del convegno triennale della Società Italiana di Logica e Filosofia delle Scienze, Roma, 3-5 gennaio 1996*. Pisa: Edizioni ETS, 255-267.
- (2004), "Is Being Computational an Intrinsic Property of a Dynamical System?", forthcoming in Gianfranco Minati, and Eliano Pessa (eds.), *Proceedings of the Third National Conference on Systems Science (A.I.R.S.)*. New York: Kluwer Academic/Plenum Publishers. URL = <<http://edu.supereva.it/giuntihome.dadacasa/download/papers/intcom-trento-submitted1bis.doc>>
- Martelli, Mario (1999), *Introduction to Discrete Dynamical Systems and Chaos*. New York: Wiley.
- Sandefur, James T (1990), *Discrete Dynamical Systems: Theory and Applications*. New York: Oxford University Press.
- Szlenk, Wieslaw (1984), *An Introduction to the Theory of Smooth Dynamical Systems*. Chichester, England: John Wiley and Sons.
- Turing, Alan M. (1936), "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, series 2, 42:230-265.
- (1950), "Computing Machinery and Intelligence", *Mind* 59:433-460.