MIT/LCS/TM-151

REVERSIBLE COMPUTING

Tommaso Toffoli

February 1980

REVERSIBLE COMPUTING*

Tommaso Toffoli

MIT Laboratory for Computer Science
545 Technology Sq., Cambridge, MA 02139

Abstract. The theory of reversible computing is based on invertible primitives and composition rules that preserve invertibility. With these constraints, one can still satisfactorily deal with both functional and structural aspects of computing processes; at the same time, one attains a closer correspondence between the behavior of abstract computing systems and the microscopic physical laws (which are presumed to be strictly reversible) that underly any concrete implementation of such systems.

Here, we integrate into a comprehensive picture a variety of concepts and results. According to a physical interpretation, the central result of this paper is that *it is ideally possible to build sequential circuits with zero internal power dissipation*. Even when these circuits are interfaced with conventional ones, power dissipation at the interface would be at most proportional to the number of input/output lines, rather than to the number of logic gates as in conventional computers.

Keywords. Reversible computing, computation universality, automata, computing networks, physical computing.

1. Introduction

Mathematical models of computation are abstract constructions, by their nature unfettered by physical laws. However, if these models are to give indications that are relevant to concrete computing, they must somehow capture, albeit in a selective and stylized way, certain general physical restrictions to which all concrete computing processes are subjected. For instance, the Turing machine, which embodies in a heuristic form the axioms of computability theory, avowedly accounts in its design[24] for the fact that the speed of propagation of information is bounded, and that the amount of information which can be encoded in the state of a finite system is bounded. However, other physical

*This research was supported by Grant N00014-75-C-0661, Office of Naval Research, funded by DARPA.

principles of comparable importance, such as the reversibility at a microscopic level of the dynamical laws (which imposes severe constraints on the operation of concrete computing primitives[12,23]) and the fact that the topology of spacetime is locally Euclidean (which severely limits the range of interconnection patterns for concrete computing structures[19]), are not yet adequately represented in the theory of computing. Conceivably, a better match between the abstract constructs of the theory and its applications would be attained if a suitable counterpart of these principles were incorporated in the theory.

Here, we shall be concerned with the issue of *reversibility*. Intuitively, a dynamical system is reversible if from any point of its state set one can uniquely trace a trajectory backward as well as forward in time. For a time-discrete system such as an automaton, this is equivalent to saying that its transition function is invertible, that is, bijective. The concept of reversibility has its origins in physics, and, in particular, in the study of continuous systems, such as those characterized by a differential equation, rather than discrete systems, which are characterized by a transition function. In the continuous case, a more technical definition is necessary; namely, a dynamical system is *reversible* if its dynamical semigroup can be expanded to a group[14]. The connection between these two definitions is immediate, since in the discrete case the transition function coincides with the generator of the system's semigroup.

It should be noted that reversibility does not imply invariance under time reversal; the latter is a more specialized notion which is definable in a nontrivial way only for dynamical systems having additional structure.

In the past, some misgivings were expressed concerning the computing capabilities of reversible automata. Such misgivings were cleared by the work of Bennett (reversible Turing machines[4]), Priese (reversible computers embedded in Thue systems[17]), Fredkin (conservative logic[7]), and Toffoli (reversible cellular automata[20]). Today, the concept of reversible computing appears to be not only productive from a theoretical viewpoint but also promising in terms of technological applications[8].

In fact, one of the strongest motivations for the study of reversible computing comes from the desire to reduce heat dissipation in computing machinery, and thus achieve higher density and speed. Briefly, while the laws of physics are presumed to be strictly reversible, abstract computing is usually thought of as an irreversible process, since it may involve the evaluation of many-to-one functions. Thus, as one proceeds down from an abstract computing task to a formal realization by means of a digital network and finally to an implementation in a physical system, at some level of this modeling hierarchy there must take place the transition from the irreversibility of the given computing process to

the reversibility of the physical laws. In the customary approach, this transition occurs at a very low level and is hidden—so to speak—in the "physics" of the individual digital gate;* as a consequence of this approach, the details of the work-to-heat conversion process are put beyond the reach of the conceptual model of computation that is used.

On the other hand, it is possible to formulate a more general conceptual model of computation such that the gap between the irreversibility of the desired behavior and the reversibility of a given underlying mechanism is bridged *in an explicit way* within the model itself. This we shall do in the present paper.

An important advantage of our approach is that any operations (such as the clearing of a register) that in conventional logic lead to the destruction of macroscopic information, and thus entail energy dissipation, here can be planned at the whole-circuit level rather than at the gate level, and most of the time can be replaced by an information-lossless variant. As a consequence, *it appears possible to design circuits whose internal power dissipation, under ideal physical circumstances, is zero.* The power dissipation that would arise at the interface between such circuits and the outside world would be at most proportional to the number of input/output lines, rather than to the number of logic gates.

2. Terminology and notation

A function $\phi: X \to Y$ is *finite* if X and Y are finite sets. A *finite automaton* is a dynamical system characterized by a transition function of the form $\tau: X \times Q \to Q \times Y$, where τ is finite.

Without loss of generality, one may assume that such sets as X, Y, and Q above be explicitly given as indexed Cartesian products of sets. We shall occasionally call *lines* the individual variables associated with the individual factors of such products. By convention, the Cartesian product of zero factors is identified with the *dummy* set $\{\lambda\}$ consisting of the *empty* word λ. In what follows, we shall assume *once and for all* that all factors of the aforementioned Cartesian products be identical copies of the *Boolean* set $\mathbf{B} = \{0, 1\}$. This assumption entails little loss of generality, and—at any rate—the theory of reversible computing could be developed along essentially the same lines if such assumption were dropped.

*Typically, the computation is logically organized around computing primitives that are not invertible, such as the NAND gate; in turn, these are realized by physical devices which, while by their nature obeying reversible microscopic laws, are made *macroscopically* irreversible by allowing them to convert some work to heat.

The concept of "function composition" is a fundamental one in the theory of computing. According to the ordinary rules for function composition, an output variable of one function may be substituted for any *number* of input variables of other functions, i.e., arbitrary "fan-out" of lines is allowed. However, the process of generating multiple copies of a given signal must be treated with particular care when reversibility is an issue (moreover, from a physical viewpoint this process is far from trivial). For this reason, in all that follows we shall restrict the meaning of the term "function composition" to *one-to-one* composition, where any substitution of output variables for input variables is one-to-one (in other words, no fan-out of lines is allowed). Any fan-out node in a given function-composition scheme will have to be treated as an explicit occurrence of a fan-out *function* of the form $\langle x \rangle \mapsto \langle x, \ldots, x \rangle$. Intuitively, the responsibility for providing fan-out is shifted from the composition rules to the computing primitives.

We shall be dealing with various classes of abstract computers (such as combinational networks, finite automata, Turing machines, and cellular automata) which constitute the main paradigms of the theory of computing. An abstract computer is, in essence, a function-composition scheme,* and a computation is a particular solution (which may be required to satisfy certain boundary conditions or other constraints) of such a scheme. While finite composition schemes are adequate for dealing with the most elementary aspects of computing, many computing processes of interest require an unbounded amount of resources and are more conveniently represented as taking place in infinite schemes.

It is customary to express a function-composition scheme in graphical form as a *causality network* (or *functional-dependence* network). This is basically an acyclic directed graph whose nodes are labeled by associating with each of them a particular finite function and whose arcs are colored by associating with each of them a particular variable. (Here, we can safely ignore certain slight technical differences between a causality network and a directed graph.)

By construction, causality networks are "loop-free," i.e., they contain no cyclic paths. A *combinational* network is a causality network that contains no infinite paths. Note that a finite causality network is always a combinational one.

With certain additional conventions, causality networks having a particular iterative structure can be represented more compactly as *sequential* networks (cf. Section 7).

We shall assume familiarity with the concept of "realization" of finite func-

*In what follows, we shall restrict our attention to function-composition schemes based on finite primitives.

tions and automata by means of, respectively, combinational and sequential networks. In what follows, a "realization" will always mean a *componentwise* one; that is, to each input (or output) line of a finite function there will correspond an input (or output) line in the combinational network that realizes it, and similarly for the realization of automata by sequential networks.

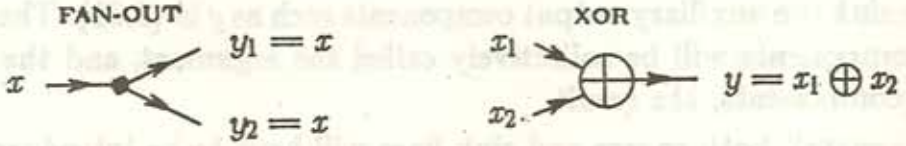
A causality network is *reversible* if it is obtained by composition of invertible primitives. Note that a reversible combinational network always defines an invertible function. Thus, in the case of combinational networks the structural aspect of "reversibility" and the functional aspect of "invertibility" coincide. A sequential network is *reversible* if its *combinational part* (i.e., the combinational network obtained by deleting the delay elements and thus breaking the corresponding arcs) is reversible.

3. Introductory concepts

As explained in Section 1, our overall goal is to develop an explicit realization of computing processes within the context of reversible systems. For the moment, the processes we shall deal with will be those described by finite functions, and the systems used for their realization will be reversible *combinational* networks. Later on, we shall consider sequential processes, characterized by the presence of internal states in addition to input and output states, and we shall study their realization by means of reversible *sequential* networks.

As an introduction, let us consider two simple functions, namely, FAN-OUT (3.1a) and XOR (3.1b):

$$\text{(a)}\quad \begin{array}{c|cc} x & y_1 & y_2 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array} \qquad \text{(b)}\quad \begin{array}{cc|c} x_1 & x_2 & y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \tag{3.1}$$



Neither of these functions is invertible. (Indeed, FAN-OUT is not *surjective*, since, for instance, the output (0, 1) cannot be obtained for any input value; and XOR

is not *injective*, since, for instance, the output 0 can be obtained from two distinct input values, $\langle 0,0\rangle$ and $\langle 1,1\rangle$). Yet, both functions admit of an invertible realization.

To see this, consider the *invertible* function XOR/FAN-OUT defined by the table

$$\begin{array}{cc} 00 & 00 \\ 01 & 11 \\ 10 & 10 \\ 11 & 01 \end{array} \quad , \tag{3.2}$$

which we have copied over with different headings in (3.3a), (3.3*b*), and (3.6*b*). Then, FAN-OUT can be realized by means of this function* as in (3.3a) (where we have outlined the relevant table entries), by assigning a value of 0 to the auxiliary input component c; and XOR can be realized by means of the same function as in (3.3*b*), by simply disregarding the auxiliary output component g. In more technical terms, (3.1a) is obtained from (3.3a) by componentwise *restriction*, and (3.1*b*) from (3.3*b*) by *projection*.

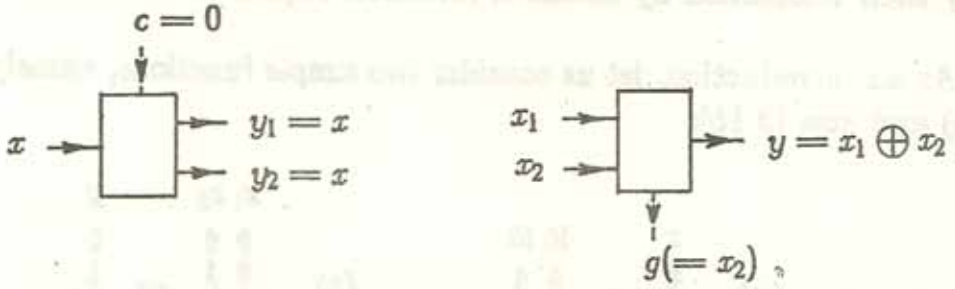
(a)

c	x	y_1	y_2
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

(b)

x_1	x_2	y	g
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

(3.3)



In what follows, we shall collectively call *the source* the auxiliary input components that have been used in a realization, such as component c in (3.3a), and *the sink* the auxiliary output components such as g in (3.3*b*). The remaining input components will be collectively called *the argument*, and the remaining output components, *the result*.

In general, both source and sink lines will have to be introduced in order

*Ordinarily, one speaks of a realization "by a network." Note, though, that a finite function by itself constitutes a trivial case of combinational network.

to construct an invertible realization of a given function.

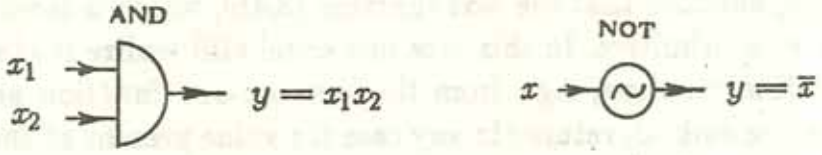
(a)

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

(b)

x	y
0	1
1	0

(3.4)



For example, from the *invertible* function AND/NAND defined by the table

000	000
001	001
010	010
011	011
100	100
101	101
110	110
111	011

(3.5)

the AND function (3.4a) can be realized as in (3.6a) with one source line and two sink lines.

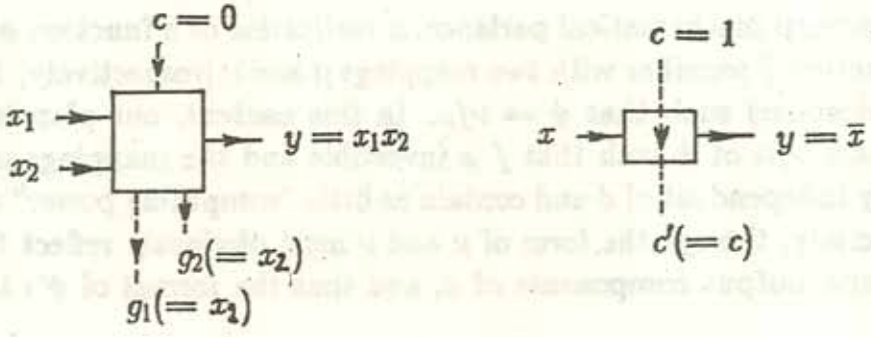
(a)

c	x_1	x_2	y	g_1	g_2
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

(b)

x	c	y	c'
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

(3.6)



Observe that in order to obtain the desired result the source lines must be fed with specified *constant* values, i.e., with values that do not depend on the argument. As for the sink lines, some may yield values that do depend on the argument—as in (3.6a)—and thus cannot be used as input constants for a new computation; these will be called *garbage* lines. On the other hand, some sink lines may return constant values; indeed, this happens whenever the functional relationship between argument and result is itself an invertible one. To give a trivial example, suppose that the NOT function (3.4*b*), which is invertible, were not available as a primitive. In this case one could still realize it starting from another invertible function, e.g., from the XOR/FAN-OUT function as in (3.6*b*); note that here the sink, c', returns in any case the value present at the source, c. In general, if there exists between a set of source lines and a set of sink lines an invertible functional relationship that is independent of the value of all other input lines, then this pair of sets will be called (for reasons that will be made clear in Section 5) a *temporary-storage channel.*

Using the terminology just established, we shall say that the above realization of the FAN-OUT function by means of an invertible combinational function is a realization *with constants*, that of the XOR function, *with garbage*, that of the AND function, *with constants and garbage*, and that of the NOT function, with *temporary storage* (for the sake of nomenclature, the source lines that are part of a temporary-storage channel will not be counted as lines of constants). In referring to a realization, features that are not explicitly mentioned will be assumed not to have been used; thus, a realization "with temporary storage" is one *without* constants or garbage. A realization that does not require any source or sink lines will be called an *isomorphic* realization.

4. The fundamental theorem

In the light of the particular examples discussed in the previous section, this section establishes a general method for realizing an *arbitrary* finite function ϕ by means of an *invertible* finite function f.

Here, we are concerned with realizations in the sense defined in Section 2. In more general mathematical parlance, a *realization* of a function ϕ consists of a new function f together with two mappings μ and ν (respectively, the *encoder* and the *decoder*) such that $\phi = \nu f \mu$. In this context, our plan is to obtain a realization $\nu f \mu$ of ϕ such that f is invertible and the mappings μ and ν are essentially independent of ϕ and contain as little "computing power" as possible. More precisely, though the form of μ and ν must obviously reflect the number of input and output components of ϕ, and thus the *format* of ϕ's truth table,

we want them to be otherwise independent of the particular *contents* of such truth table as ϕ is made to range over the set of all combinatorial functions.

In general, given any finite function one obtains a new one by assigning specified values to certain distinguished input lines (*source*) and disregarding certain distinguished output lines (*sink*). According to the following theorem, any finite function can be realized in this way starting from a suitable *invertible* one.

THEOREM 4.1 *For every finite function* $\phi: \mathbf{B}^m \to \mathbf{B}^n$ *there exists an invertible finite function* $f: \mathbf{B}^r \times \mathbf{B}^m \to \mathbf{B}^n \times \mathbf{B}^{r+m-n}$, *with* $r \leq n$, *such that*

$$f(\overbrace{0, \ldots, 0}^{r}, x_1, \ldots, x_m) = \phi_i(x_1, \ldots, x_m), \qquad (i = 1, \ldots, n). \tag{4.1}$$

Proof. Let ϕ be defined by a binary table of the following form

$$2^m \left\{ \begin{array}{|c|} \hline \overbrace{\quad X \quad}^{m} \\ \hline \end{array} \right. \rightarrow \begin{array}{|c|} \hline \overbrace{\quad Y \quad}^{n} \\ \hline \end{array} ,$$

where X denotes a listing of all 2^m m-tuples over B and Y denotes a listing of the corresponding values of ϕ, which are n-tuples over B. We shall define a function $f: \mathbf{B}^{n+m} \to \mathbf{B}^{n+m}$ by means of the following table

$$2^n \text{blocks} \left\{ \begin{array}{|c|c|} \hline \overbrace{0}^{n} & \overbrace{X}^{m} \\ \hline 1 & X \\ \hline \ldots & \ldots \\ \hline 2^n - 1 & X \\ \hline \end{array} \right. \rightarrow \begin{array}{|c|c|} \hline \overbrace{Y}^{n} & \overbrace{X}^{m} \\ \hline Y+1 & X \\ \hline \ldots & \ldots \\ \hline Y+2^n-1 & X \\ \hline \end{array} ,$$

where each block of the form k $(0 \leq k < 2^n)$ consists of 2^m identical n-tuples each representing the integer k written in base 2, while each block of the form $Y + k$ $(0 \leq k < 2^n)$ consists of the 2^m entries of Y each treated as a base-2 integer and incremented by $k \bmod 2^n$. (So the sequence of "$Y + k$" blocks differs from the "k" sequence only by a circular permutation.) By construction, each side of this table contains each element of $\mathbf{B}^{n+m}$ exactly once. Thus, f is invertible. Moreover, equation (4.1), with $r = n$, holds by construction. ∎

The meaning of Theorem 4.1 is illustrated in Figure 4.1 below. The operations of restriction and projection mentioned in Section 3 are respectively performed by an input encoder μ and an output decoder ν. While letting through the argument $\langle x_1, \ldots, x_m \rangle$ unchanged, the encoder supplies the r source lines with constant values, i.e., with values that do not depend on the argument itself. On the other hand, while letting through the result $\langle y_1, \ldots, y_n \rangle$ unchanged, the decoder absorbs whatever values come out of the the $m + r - n$ sink lines.

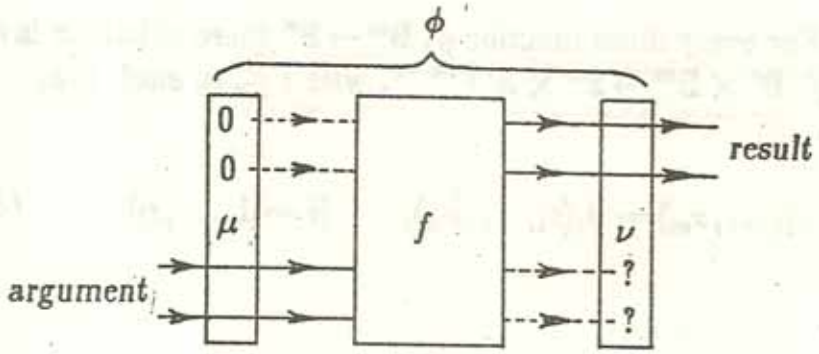


FIG. 4.1 *Any finite function ϕ can be written as the product of a trivial encoder μ, an invertible finite function f, and a trivial decoder ν.*

Intuitively, such a realization of ϕ by means of an invertible function f is "fair," in the sense that neither μ nor ν contribute to the "computing power" of f.

Since μ does not interact with the input signals, it will be more convenient to visualize it as a separate *source* of constant input values, as in Figure 4.2*b* rather than as an *input encoder*, as in Figure 4.1; likewise, ν is more conveniently visualized as a separate *sink* of output values rather than as an *output decoder*. To sum up, whatever can be computed by an arbitrary finite function according to the schema of Figure 4.2*a* can also be computed by an *invertible* finite function according to the schema of Figure 4.2*b*.

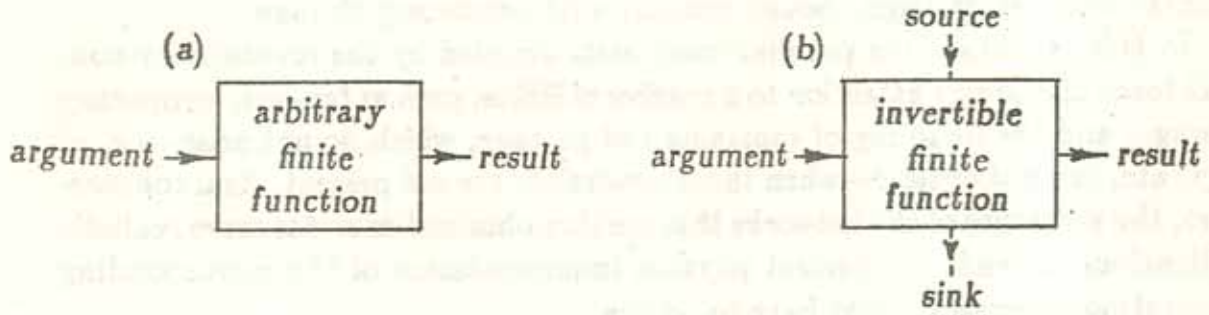


FIG. 4.2 *Any finite function (a) can be realized as an invertible finite function (b) having a number of auxiliary input lines which are fed with constants and a number of auxiliary output lines whose values are disregarded.*

Remark. The construction in the proof of Theorem 4.1 does not necessarily lead to a *minimal* realization, as often the number of source lines can be made strictly less than n and, correspondingly, the number of sink lines strictly less than m. Intuitively, in many cases the given function ϕ is such that much of the information contained in the argument is retained in the result, so that in order to guaranteee invertibility f need only preserve in the garbage signals a *fraction* of the total information.

5. Invertible primitives and reversible networks

In the previous section, each given ϕ was realized by a reversible combinational network consisting of a single occurrence of an *ad hoc* primitive f. In this section, we shall study the realization of arbitrary finite functions by means of reversible combinational networks constructed from given primitives; in particular, from a certain finite set U of very simple primitives.

If the given function ϕ is defined by means of an *arbitrary* combinational network (in what follows, we shall assume for simplicity that this network be based on the NAND element), a reversible realization of ϕ based on the set U can be obtained in a very simple way by subjecting the given network to straightforward translation rules. However, the reversible realization that is obtained in this way in general requires many more sink lines (and, consequently, many more source lines) than the realization of Section 4. On the other hand, starting from the same set U of primitives but using more sophisticated synthesis techniques it is possible to obtain a reversible realization of ϕ that does not require any more garbage lines than when realizing ϕ by means of an *ad hoc* primitive f. In fact, f itself—or, for that matter, any invertible finite function—can be synthesized

from U *without garbage*, though possibly with temporary storage.

In this synthesis, the peculiar constraints dictated by the reversibility context force one to pay attention to a number of issues, such as fan-out, temporary storage, and the handling of constants and garbage, which do not arise—or, at any rate, are not critical—when these constraints are not present. As a counterpart, the structure of the networks that are thus obtained provides more realistic indications of what an efficient physical implementation of the corresponding computing processes would have to be like.

It is well known that, under the ordinary rules of function composition, the two-input NAND element constitutes a universal primitive for the set of all combinational functions.

In the theory of reversible computing, a similar role is played by the AND/NAND element, defined by (3.5) and graphically represented as in Figure 5.1c. Referring to (3.6a), observe that $y = x_1 x_2$ (AND function) when $c = 0$, and $y = \overline{x_1 x_2}$ (NAND function) when $c = 1$. Thus, as long as one supplies a value of 1 to input c and disregards outputs g_1 and g_2, the AND/NAND element can be substituted for any occurence of a NAND gate in an ordinary combinational network.

In spite of having ruled out fan-out as an intrinsic feature provided by the composition rules, one can still achieve it as a *function* realized by means of an invertible primitive, such as the XOR/FAN-OUT element defined by (3.2) and graphically represented as in Figure 5.1b. In (3.3a), observe that $y_1 = y_2 = x$ when $c = 0$ (FAN-OUT function); and in (3.3b), that $y = x_1 \oplus x_2$ (XOR function).

Finally, recall that finite composition always yields invertible functions when applied to invertible functions (cf. Section 2).

Therefore, using the set of invertible primitives consisting of the AND/NAND element and the XOR/FAN-OUT element, any combinational network can be immediately translated into a reversible one which, when provided with appropriate input constants, will reproduce the behavior of the original network. Indeed, even the set U consisting of the single element AND/NAND is sufficient for this purpose, since XOR/FAN-OUT can be obtained from AND/NAND, with one line of temporarty storage, by taking advantage of the mapping $\langle 1, p, q\rangle \mapsto \langle 1, p, p \oplus q\rangle$.)

The element-by-element substitution procedure outlined above for constructing a reversible-network realization of a given combinational function is *wasteful*, in the sense that the number of source and sink lines that are introduced by this construction is roughly proportional to the number of computing elements that make up the network, and therefore in general much larger than the minimum required to compensate for the noninvertibility of the given function (cf. Section 4).

From the viewpoint of a physical implementation, where signals are encoded in some form of energy, each constant input entails the supply of energy of predictable form, or *work*, and each garbage output entails the removal of energy of unpredictable form, or *heat*. In this context, a realization with fewer source and sink lines might point the way to a physical implementation that dissipates less energy.

Our plan to achieve a less wasteful realization will be based on the following concept. While it is true that each garbage signal is "random," in the sense that it is not predictable without knowing the value of the argument, yet it will be correlated with other signals in the network. Taking advantage of this, one can augment the network in such a way as to make correlated signals interfere with one another and produce a number of *constant* signals instead of garbage. These constants can be used as source signals in other parts of the network. In this way, the overall number of both source and sink lines can be reduced. This process is analogous to the destructive interference of, say, sound waves. It is well-known that by superposing two correlated random signals one may obtain an overall signal which is less "noisy" that either component.

In the remainder of this section we shall show how, in the abstract context of reversible computing, destructive interference of correlated signals can be achieved in a systematic way. For a similar process of destructive interference to take place in concrete computers (thus leading to greatly reduced power dissipation), one would have to match the abstract invertible primitives with digital gates taht are macroscopically—as well as microscopically—reversible. The arguments in [7,8,23] strongly suggest that such gates are indeed physically realizable.

Returning to our mathematical exposition, we shall first show that any invertible finite function can be realized *isomorphically* from certain *generalized* AND/NAND primitives. Then, we shall show that any of these primitives can be realized from the AND/NAND element possibly with temporary storage but with no garbage.

For convenience, we shall say that an invertible finite function is *of order* n if it has n input lines and n output lines.

DEFINITION 5.1 Consider the set $\mathbf{B} = \{0,1\}$ with the usual structure of Boolean ring, with "$\oplus$" (*exclusive*-OR) denoting the addition operator, "$\ominus$" the additive-inverse operator (which in this case coincides with the identity operator), and juxtaposition (AND) the multiplication operator. For any $n > 0$, the *generalized* AND/NAND *function of order* n, denoted by $\theta^{(n)}: \mathbf{B}^n \to \mathbf{B}^n$, is

defined by

$$\theta^{(n)}: \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ \ominus x_n \oplus x_1 x_2 \cdots x_{n-1} \end{pmatrix}. \tag{5.1}$$

Remark. (a) The $\ominus$ sign in (5.1), which is redundant (since $\ominus x_n = x_n$), has been introduced for ease of comparison with the arguments of [23]. (*b*) For any $n > 0$, $\theta^{(n)}$ is invertible and coincides with its inverse. (c) For $i = 1, 2, \ldots, n-1$, the i-th component of $\theta^{(n)}$, i.e., $\theta_i^{(n)}$, coincides with the *projection* operator for the corresponding argument, i.e., $\theta_i^{(n)}(x_1, \ldots, x_n) = x_i$. (*d*) The last component of $\theta^{(n)}$, i.e., $\theta_n^{(n)}$, coincides with the NOT function for $n = 1$ (note that, by convention, $x_1 \cdots x_i = 1$ when $i = 0$), and with the exclusive-OR of its two arguments for $n = 2$. (e) For all other values of n, $\theta_n^{(n)}$ is still linear in the n-th argument, but is *nonlinear* in the first $n - 1$ arguments.

We have already encountered $\theta^{(1)}$ under the name of the NOT element, $\theta^{(2)}$ under the name of the XOR/FAN-OUT element, and $\theta^{(3)}$ under the name of the AND/NAND element. The generalized AND/NAND functions are graphically represented as in Figure 5.1*d*.

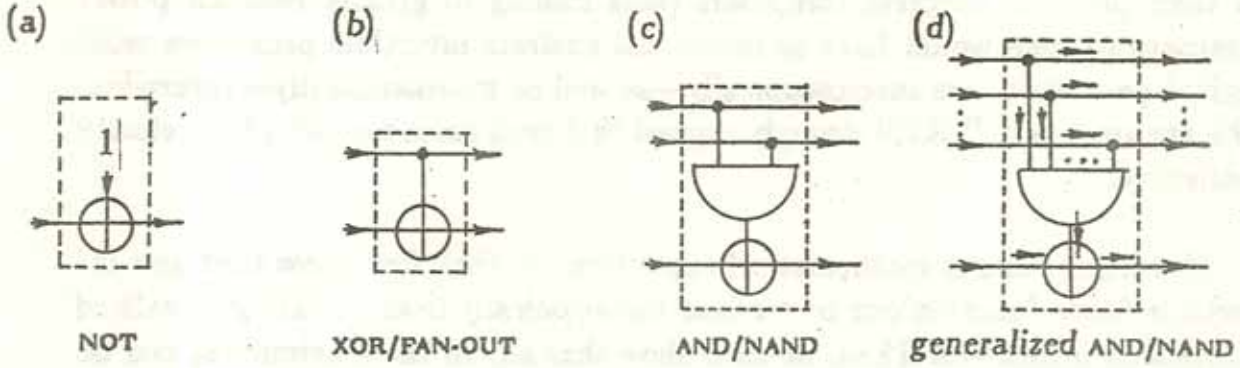


FIG. 5.1 *Graphic representation of the generalized* AND/NAND *functions.* WARNING: This representation is offered only as a mnemonic aid in recalling a function's truth table, and is not meant to imply any "internal structure" for the function, or suggest any particular implementation mechanism. *(a) $\theta^{(1)}$, which coincides with the* NOT *element; (b) $\theta^{(2)}$, which coincides with the* XOR/FAN-OUT *element; (c) $\theta^{(3)}$, which coincides with the* AND/NAND *element; and, in general, (d) $\theta^{(n)}$, the generalized* AND/NAND *function of order n. The bilateral symmetry of these symbols recalls the fact that each of the corresponding functions coincides with its inverse.*

An invertible function acts as a permutation on the elements of its domain, and it is well known that any permutation can be written as a product of *elementary* permutations, i.e., of permutations that exchange exactly two elements. In our attempt to synthesize arbitrary invertible functions of order n we shall consider, as building blocks, elementary permutations on $\mathbf{B}^n$ and even more basic permutations on $\mathbf{B}^n$ called *atomic* permutations.

DEFINITION 5.2 In the truth table for the AND/NAND element (3.5), observe that the only difference between the left- and the right-hand side of the table is that exactly two rows (namely, $\langle 0,1,1\rangle$ and $\langle 1,1,1\rangle$) which have a *Hamming distance* of 1 (i.e., differ in exactly one position) have been exchanged. An *atomic permuter* is any function having this property.

Any atomic permuter of order n can be constructed from the generalized AND/NAND element of order n by appending NOT elements to some of the input lines and to each of the corresponding output lines, as in Figure 5.2a. Graphically, this permuter will be represented as in Figure 5.2b, where some of the inputs to the AND-gate symbol are negated (as denoted by a little circle).

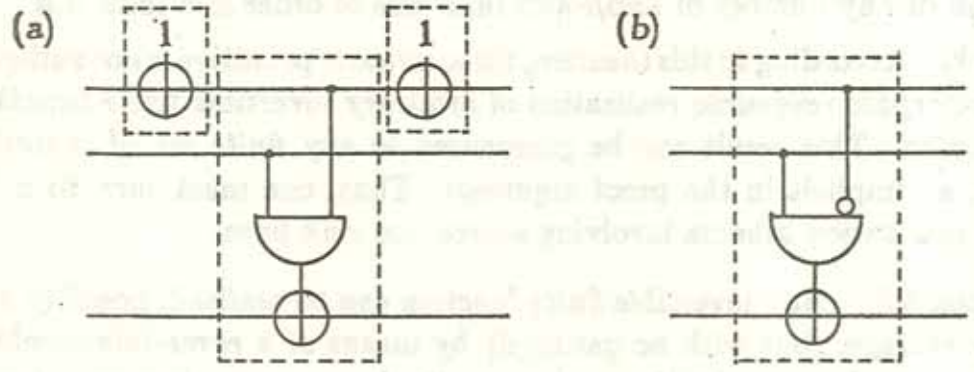


FIG. 5.2 *(a) Construction and (b) graphic representation of a particular atomic permuter.*

THEOREM 5.1 *Any invertible finite function of order n can be obtained by composition of atomic permuters of order n, and therefore by composition of generalized* AND/NAND *functions of order $\leq n$.*

Proof. It will be sufficient to show that one can obtain any elementary permutation on $\mathbf{B}^n$.

In a table of all elements of $\mathbf{B}^n$, a sequence of table entries $a_1, a_2, \ldots, a_i$ (these are all n-tuples) are said to form a *Gray-code path* if any two entries that are adjacent in this sequence are related by an atomic permutation. Consider the pair of entries x and y that we wish to exchange, and consider a Gray-code path

from x to y (such a path exists for any pair x, y). It is easy to verify that by means of sequence of atomic permutations item x can be moved to the end of the path, while the remainder of the path is shifted one position to the left but is otherwise unchanged. In a similar way, y can be brought to the beginning of the path. Thus x and y can be exchanged by means of a sequence of atomic permutations without affecting the rest of the table.∎

Remark. Note that the realization referred to by Theorem 5.1 is an *isomorphic* one (unlike that of Section 4, which makes use of source and sink lines).

THEOREM 5.2 *There exist invertible finite functions of order n which cannot be obtained by composition of generalized* AND/NAND *functions of order strictly less than n.*

Proof. In the context of the proof of Theorem 5.1, when $\theta^{(i)}$ is applied to any i components of B^n this set is divided into 2^{n-i} disjoint collections of 2^i n-tuples, and each collection is permuted by $\theta^{(i)}$ in an identical fashion. Thus, only even permutations can be obtained when $i < n$. Since the product of even permutations is even, only even permutations can be obtained by one-to-one composition of any number of AND/NAND functions of order less than n.∎

Remark. According to this theorem, the AND/NAND primitive is not sufficient for the *isomorphic* reversible realization of arbitrary invertible finite functions of larger order. This result can be generalized to any *finite* set of invertible primitives, as implicit in the proof argument. Thus, one must turn to a less restrictive realization schema involving source and sink lines.

THEOREM 5.3 *Any invertible finite function can be realized, possibly with temporary storage,* [but with no garbage!] *by means of a reversible combinational network using as primitives the generalized* AND/NAND *elements of order* ≤ 3.

Proof. In view of Theorem 5.1, it will be sufficient to realize (possibly with temporary storage), for each n, all atomic permuters of order n. Since these can be realized isomorphically from $\theta^{(1)}$ and $\theta^{(n)}$ (cf. Figure 5.2), it will be sufficient to realize $\theta^{(n)}$ itself. We shall proceed by recursion; namely, given $\theta^{(n-1)}$, $\theta^{(n)}$ can be realized with one line of temporary storage as follows.

Construct the network of Figure 5.3, which contains two occurrences of $\theta^{(n-1)}$ and one occurrence of $\theta^{(3)}$. Observe that $c' \equiv c$, since every generalized AND/NAND element coincides with its inverse (and thus the second occurrence of $\theta^{(n-1)}$ cancels the effect of the first). Therefore, the pair $\langle\{c\}, \{c'\}\rangle$ constitutes a temporary-storage channel. When $c = 0$, the remaining variables behave as the corresponding ones of $\theta^{(n)}$.∎

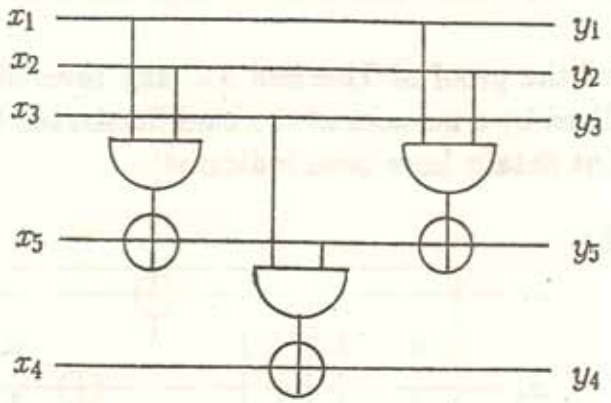


FIG. 5.3 *Realization with temporary storage of $\theta^{(n)}$ from $\theta^{(n-1)}$ (and $\theta^{(3)}$). In this network, when $c = 0$, also $c' = 0$, and the remaining components behave as the corresponding ones of $\theta^{(n)}$.*

In the above construction, it is clear that one line of temporary storage is added every time that one realizes the AND/NAND function of the next higher order. Thence the following theorem.

THEOREM 5.4. *In Theorem 5.3, let n be the order of the given invertible function, and m the number of source (as well as sink) lines required required for temporary-storage channels in the realization. Then m need not exceed $n - 3$. If only $\theta^{(3)}$ is given as a primitive, then m need not exceed $3n - 3$.*

Remark. The second part of this theorem reflects the fact that additional temporary-storage lines are needed to realize NOT from AND/NAND (cf. (3.6)).

The proof of Theorem 5.3 establishes a general mechanisms for bringing about destructive interference of garbage. With reference to Figure 5.3, which can serve as an outline for the general case, observe that the left portion of the network is accompanied by its "mirror image" on the right. The left portion computes an intermediate result (on the line running from c to c') that is needed as an input to the lower portion and is returned by it unchanged. Having performed its function, this intermediate result is then "undone" by the right portion, so that no garbage is left.

The reader may refer to [3,7] for more specific examples of destructive interference of garbage.

Taken together, Theorems 5.3 and 5.4 have an interesting interpretation from the viewpoint of complexity theory. They imply a trade-off between the number of available primitives and the availability of an appropriate amount of a resource that, as we shall presently explain, can be intuitively identified with

temporary storage (hence the term "temporary-storage channel" introduced in Section 3.)

Following the lines of the proof of Theorem 5.3, any invertible function f (Figure 5.4a) can be realized by a network of the kind illustrated in Figure 5.4b (where only some relevant details have been indicated).

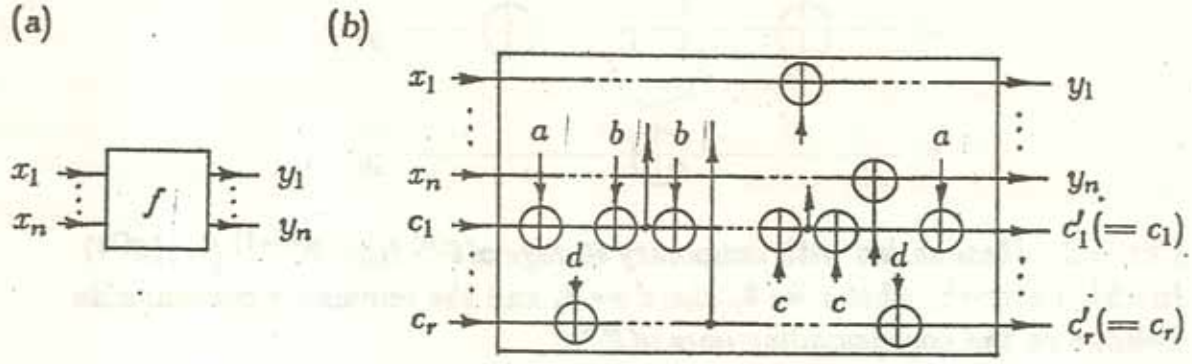


FIG. 5.4 *If instead of an ad hoc reversible mechanism (a) for the invertible function f one seeks a mechanism based on the* AND/NAND *primitive, as in (b), auxiliary constant input signals are required. Such signals are returned unchanged at the output.*

Let us consider the process of traversing the box of Figure 5.4b from left to right. Each line running through the box represents the evolution of a binary storage element. In general, all the constants entering the box will be repeatedly written over as they traverse the box itself; yet, they will emerge from the box with the original values. In this context, a temporary storage channel (such as the pair $\langle\{c_1, \ldots, c_r\}, \{c'_1, \ldots, c'_r\}\rangle$) represents a "scratchpad" register which is initialized to an assigned state (say, all 0's) and is invariably restored to the original state before the end of the computation. (Though more general, this behavior is analogous to that of the reversible Turing machines described by Bennett[4] and briefly discussed in Section 7.)

In this context, Theorem 5.2 can be interpreted as saying that, for a given choice of reversible primitives, it may be impossible to carry out the computation of a given invertible finite function if one is restricted to working on a register of size just enough to contain the argument (cf. the limitations of linear-bounded automata). On the other hand, Theorem 5.3 says that this difficulty disappears as soon as one permits the use of an auxiliary temporary-storage register of appropriate size.

The following list (cf. Figure 5.5) sums up in a schematic way the input/output resources of which a reversible network must avail itself in order

to be able to compute a finite function ϕ.

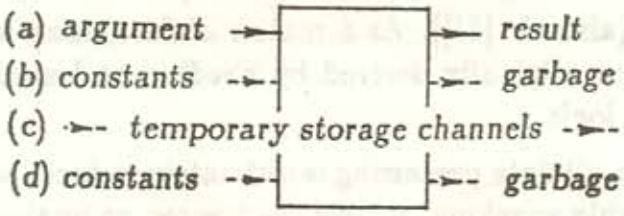


FIG. 5.5 *Classification of input and output lines in a reversible combinational network, according to their function. (a) Argument and result of the intended computation. (b) Constant and garbage lines to account for the noninvertibility of the given function. (c) "Temporary storage" registers required when only a restricted set of primitives is available. (d) Additional constant and garbage lines required when in designing the network one chooses not to take full advantage of the correlation between internal streams of data, and thus looses opportunities to bring about destructive interference of garbage.*

(a) If ϕ is invertible, and the use of *ad hoc* primitives is allowed, then no source or sink lines are necessary.

(*b*) If ϕ is not invertible, it is necessary to provide source lines to be fed with specified constants and/or sink lines which will produce garbage.

(*c*) Independently of the invertibility of ϕ, if only a restricted set of primitives is allowed it may be necessary to supply temporary-storage channels (i.e, additional source lines to be fed with constants and an equal number of sink lines which will return constants).

(*d*) If, in order to simplify the mechanism, one chooses to ignore that certain streams of garbage within the mechanism are correlated and thus could be subjected to destructive interference, then further source and sink lines will be required, as in (*b*).

6. Conservative logic

In view of the considerable exertions that were necessary in the previous sections in order to obtain a bona fide realization of mechanisms having universal logic capabilities and at the same time satisfying the reversibility constraint, one might wonder whether nontrivial computation would be possible at all if substantial further constraints were imposed.

Actually, it turns out that universal logic capabilities can still be obtained even if one restricts one's attention to combinational networks that, in addition

to being reversible, conserve in the output the number of 0's and 1's that are present at the input. The study of such networks is part of a discipline called *conservative logic** [7] (also cf. [11]). As a matter of fact, most of the results of Sections 4 and 5 were originally derived by Fredkin and associates in the context of conservative logic.

In conservative logic, all data processing is ultimately reduced to *conditional routing* of signals. Roughly speaking, signals are treated as unalterable objects that can be moved around in the course of a computation but never created or destroyed. The physical significance of this principle will be discussed shortly.

The basic primitive of conservative logic is the *Fredkin gate* (cf. [76]), defined by the table

c	x_1	x_2		c'	y_1	y_2
0	0	0		0	0	0
0	0	1		0	1	0
0	1	0		0	0	1
0	1	1	$\rightarrow$	0	1	1
1	0	0		1	0	0
1	0	1		1	0	1
1	1	0		1	1	0
1	1	1		1	1	1

(6.1)

This computing element can be visualized as a device that performs conditional crossover of two *data* signals a and b according to the value of a *control* signal c (Figure 6.1a). When $c = 1$ the two data signals follow parallel paths, while when $c = 0$ they cross over (Figure 6.1*b*).

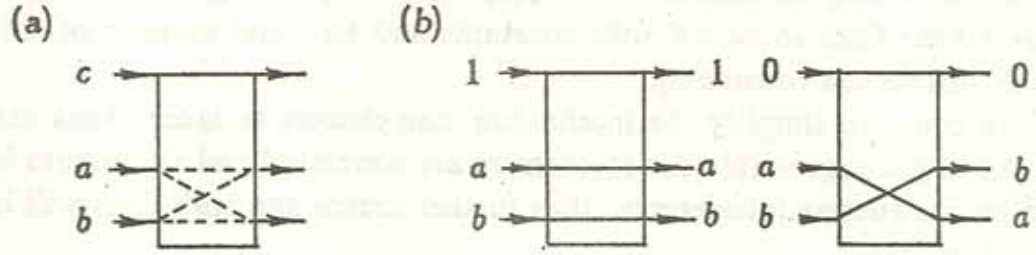


FIG. 6.1 *(a) Symbol and (b) operation of the Fredkin gate.*

In order to prove the universality of this gate as a logic primitive for reversible computing, it is sufficient to observe that AND can be obtained from the mapping

*Besides a combinational primitive (the *Fredkin gate* discussed below), conservative logic also posits a communication/memory primitive (the *unit wire*). Together, these two primitives are sufficient to account for sequential computation. Moreover, they provide a basis for a theory of computation complexity which incorporates certain important physical-like constraints that are usually neglected.

$\langle p, q, 0\rangle \mapsto \langle p, pq, \bar{p}q\rangle$, and NOT and FAN-OUT from the mapping $\langle p, 1, 0\rangle \mapsto \langle p, p, \bar{p}\rangle$.

There exists only one other 3-input, 3-output conservative-logic element that can be used as a universal primitive; this is the *symmetric majority/parity gate* (SMP) defined by

$$\begin{array}{ccc|ccc} x & y & z & x' & y' & z' \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \qquad (6.2)$$

Intuitively, in the SMP gate the three signals are rotated in one direction if their overall parity is even, and in the other direction if it is odd. While the SMP gate can be realized isomorphically by means of Fredkin gates, a realization of the latter by the former requires a temporary-storage channel. In this sense, the Fredkin gate is the *most elementary* conservative-logic primitive—since no 2-input, 2-output invertible function is universal.*

Finally, the Fredkin gate can be realized *isomorphically* by means of AND/NAND gates, as shown in Figure 6.2.

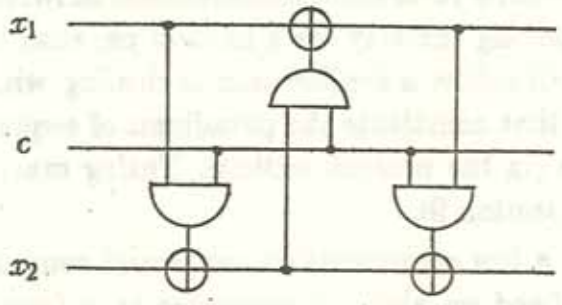


FIG. 6.2 *Isomorphic realization of the Fredkin gate by means of* AND/NAND *gates.*

It is important to realize that, far from representing merely an elegant *tour-de-force* in the theory of reversible computing, conservative logic provides an essential connection between that theory and the physics of computing circuits. In

*An even more elementary conservative-logic primitive, namely, the *interaction gate*[7], can be obtained if one considers invertible function whose domain and codomain coincide with a *proper subset* of a set of the form B^n.

fact, while in any reversible system whatsoever there are a number of conserved quantities, in *physical* system many of these quantities are required to have a special structure. For instance, energy, momemtum, and angular momentum are *additive* (this reflects certain symmetries of space and time). Intuitively, reversibility by itself is not sufficient to give enough physical "flavor" to a theory of computing.

In a conservative logic circuit, the number of 1's, which is conserved in the operation of the circuit, is the sum of the number of 1's in different parts of the circuit. Thus, this quantity is additive, and can be shown to play a formal role analogous to that of energy in physical systems. Other connections between conservative logic and physics will discussed in more detail in [7]

In conclusion, conservative logic represents a substantial step in the development of a model of computation that adequately reflects the basic laws of physics.

7. Reversible sequential computing

In Sections 4 and 5, we started from a certain computing object (viz., a finite function), and we discussed the conditions for its *reversible* realization first (a) as an object of the same nature (viz., an invertible finite function) treated as a "lumped" system, thus stressing *functional* aspects, and then (*b*) as a "distributed" system (viz., a reversible combinational network), thus stressing *structural* aspects and paving the way for a natural physical implementation.

By and large, we shall follow a similar plan in dealing with the more complex computing objects that constitute the paradigms of sequential computing, namely, *finite automata* (in the present section), *Turing machines* (Section 8), and *cellular automata* (Section 9).

First, we shall make a few commemts on sequential computing in general.

In Section 2, we defined an abstract computer as a function-composition scheme. Some of these schemes possess a regular structure of a certain kind, namely, they are *time-iterative*,* and with appropriate conventions can be represented much more compactly in terms of *sequential* networks (cf. [9]).

*If the *partial-order* relation between arcs that is induced by the function-composition operation is interpreted as a *causal* relationship between signals, then certain other relations between arcs are naturally interpreted as referring to *spatial* and *temporal* relationships. In this context, a causal network may possess certain automorphisms that may be interpreted as *time shifts*, and in this case the network is *time-iterative*; similarly, it may possess automorphisms corresponding to *spatial* shifts, and in this case it is *space-iterative*.

Consider, for definiteness, the iterative function-composition scheme

$$\begin{cases} q_{i+1} = x_i \oplus q_i \\ y_{i+1} = x_i \oplus q_i \end{cases} \quad (-\infty < i < +\infty), \tag{7.1}$$

explicitly represented by the infinite causal network of Figure 7.1.

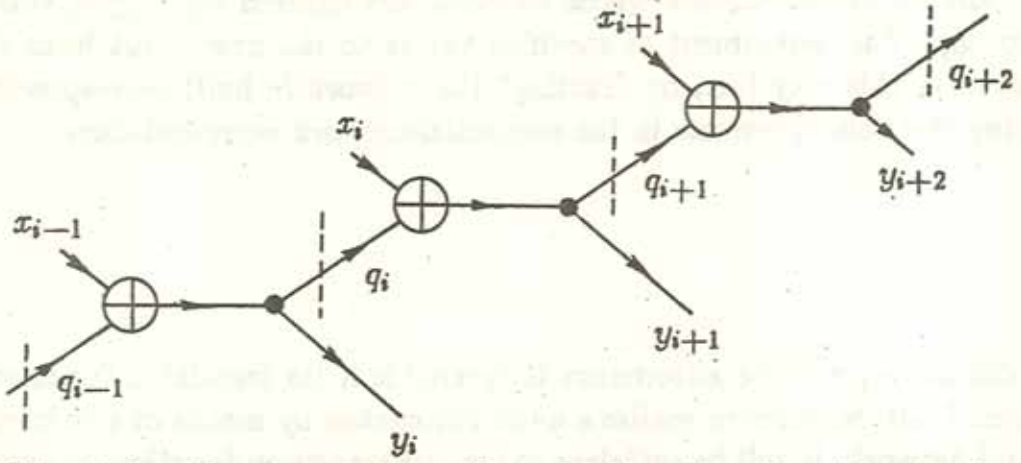


FIG. 7.1 *A function-composition scheme having a time-iterative structure.*

The same scheme can be represented by the finite sequential network of Figure 7.2. By comparing this figure with Figure 7.1, it is clear that the role of the so-called "delay elements" in a sequential network is to mark out those places that correspond to the boundary between one stage and the next in the equivalent time-iterative combinational network.

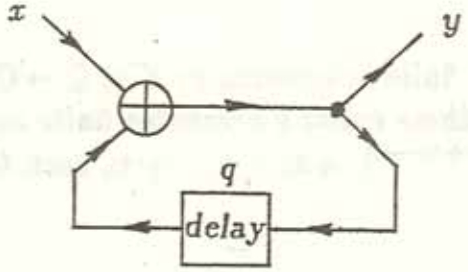


FIG. 7.2 *The same function-composition scheme as represented by a finite sequential network.*

In the previous sections, for the sake of realization arguments we have treated a finite function as a combinational network consisting of a single node. Similarly, we shall treat a finite automaton as a sequential network whose combinational part consists of a single node, representing the automaton's transition function. Thus, for example, the network of Figure 7.2 can be identified with the finite automaton $\langle x, q \rangle \mapsto \langle x \oplus q, x \oplus q \rangle$.

Finally, recalling that a computation is a solution of a function-composition scheme (cf. Section 2), let us note that one is usually not interested in arbitrary solutions; rather, one seeks solutions corresponding to particular boundary conditions (defined, for instance, by assigning specified values to the input lines). Moreover, in the case of a time-iterative scheme, one is usually interested only in that portion of the scheme whose elements are indexed by $i \geq i_0$ (for an arbitrary i_0). The assignment of specified values to the new input lines that are created in this way (i.e., by "cutting" the network in half) corresponds to *initializing* the delay elements in the sequential-network representation.

By definition, a finite automaton is *reversible* if its transition function is invertible. Thus, in order to realize a finite automaton by means of a reversible sequential network, it will be sufficient to take its transition function, construct a reversible realization of it, and use this as the combinational part of the desired sequential network. The problem of reversibly realizing an arbitrary finite function has been solved in Section 4. Thus, we have the following theorem.

THEOREM 7.1. *For every finite automaton* $\tau: X \times Q \to Q \times Y$, *where* $X = B^m$, $Y = B^n$, *and* $Q = B^u$, *there exists a reversible finite automaton* $t: (B^r \times B^m) \times B^u \to B^u \times (B^n \times B^{r+m-n})$, *with* $r \leq n + u$, *such that*

$$t_i(\overbrace{0, \ldots, 0}^{r}, x_1, \ldots, x_m, q_1, \ldots, q_u) = \tau_i(x_1, \ldots, x_m, q_1, \ldots, q_u), \quad (i = 1, \ldots, u+n).$$

In other words, whatever can be computed by an arbitrary finite automaton according to the scheme of Figure 7.3a can also be computed by a *reversible* finite automaton according to the schema of Figure 7.3b.

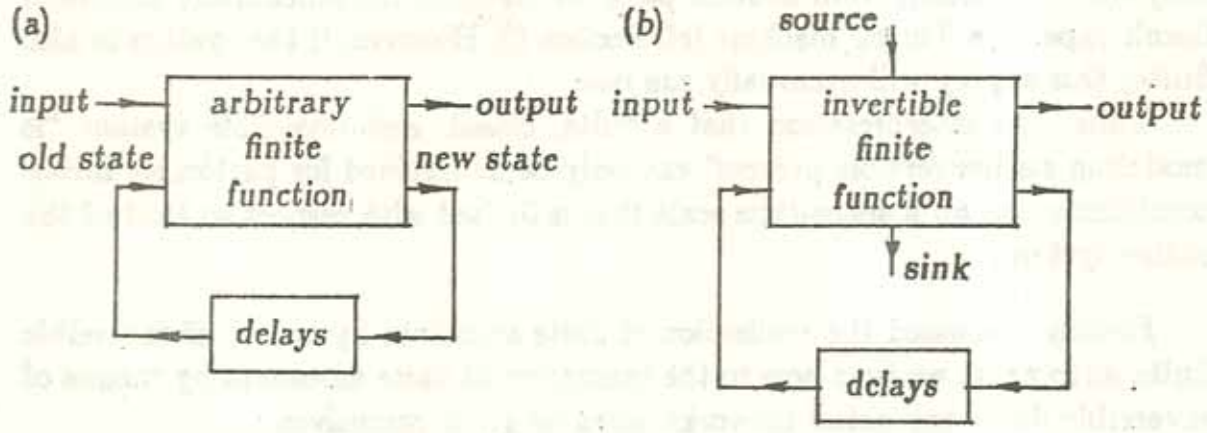


FIG. 7.3 *Any finite automaton (a) can be realized as a reversible finite automaton (b) having a number of auxiliary input lines (source) which are fed with constants and a number of auxiliary output lines (sink) whose values are disregarded.*

The following theorem, which from a mathematical viewpoint is trivial, is a restatement of the second law of thermodynamics. Recall that an automaton is *closed* if $X = \{\lambda\}$ (i.e., there are no input lines); *open*, otherwise.

THEOREM 7.2 *If a closed, finite automaton is not reversible, then it does not admit of a reversible realization that is both closed and finite.*

Proof. Given a nonreversible automaton $\tau: Q \to Q$, assume that $t: P \to P$ be a reversible realization of τ. That is, we assume the existence of an invertible finite function t and of two maps μ and ν (respectively, the *encoder* and the *decoder*) such that, for all $i \geq 0$,

$$\nu t^i \mu = \tau^i. \tag{7.2}$$

Since automaton t is finite and reversible, for any $p \in P$ there exists an $i_p > 0$ such that $t^{i_p}(p) = p$. Thus, for any $q \in Q$, $\nu t^{i_{\mu(q)}}\mu(q) = \nu\mu(q)$. But, in view of (7.2), $\nu\mu(q) = q$ and $\nu t^{i_{\mu(q)}}\mu(q) = \tau^{i_{\mu(q)}}(q)$. As a consequence, $\tau^{i_{\mu(q)}}(q) = q$ for all q. On the other hand, since automaton τ is finite and nonreversible, there must be at least one $\bar{q} \in Q$ such that $\tau^i(\bar{q}) \neq \bar{q}$ for all $i > 0$. Thus, the original assumption leads to a contradiction. ∎

Remark. According to Theorem 4.1, noninvertible functions can be computed by a reversible system provided that a supply of constants is made available. In an open system, this supply may come from some input lines. In a

closed system, in order to perform a computation in one part of the system one may draw constants from another part, for instance, the indefinitely extended blank tape of a Turing machine (cf. Section 8). However, if the system is also finite, this supply will eventually run out.

Thus, the interpretation that a finite, closed, and reversible system "is modeling an irreversible process" can only be maintained for particular initial conditions and on a space-time scale that is limited with respect to that of the entire system.

Having discussed the realization of finite automata by means of reversible finite automata, we turn now to the realization of finite automata by means of reversible finite sequential networks based on given primitives.

It is clear that all the arguments of Sections 5 and 6 concerning finite functions immediately apply to the transition function of any given finite automaton. In particular, in analogy with Figure 5.5, every finite automaton can be realized by a finite, reversible sequential network based on, say, the AND/NAND primitive and having the following form (Figure 7.4)

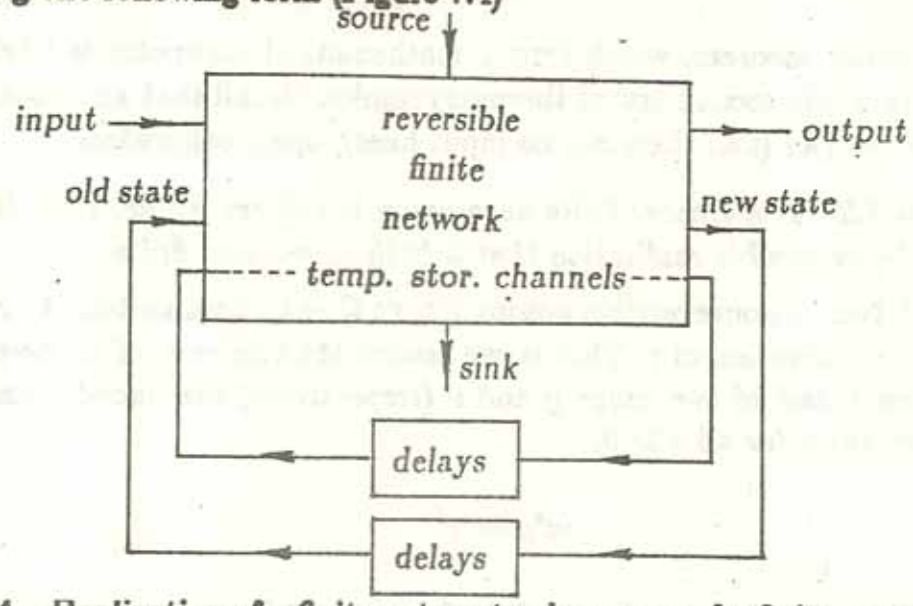


FIG. 7.4 *Realization of a finite automaton by means of a finite, reversible sequential network.*

In order to insure the desired behavior, the state components represented by the temporary-storage channels must be initialized *once and for all* with appropriate values (typically, all 0's), while the source must be fed with appropriate constants (typically, all 0's) *at every sequential step.*

In a conventional computer, power dissipation is proportional to the number of logic gates. On the other hand, the number of constants/garbage lines in

Figure 7.4 is at worst proportional to the number of input/output lines (cf. Theorems 4.1 and 5.3). From the viewpoint of a physical implementation, where signals are encoded in some form of energy, the above schema can be interpreted as follows: *Using invertible logic gates, it is ideally possible to build a sequential computer with zero internal power dissipation.* The only source of power dissipation arises outside the circuit, typically at the input/output interface, if the user chooses to connect input or output lines to nonreversible digital circuitry. Even in this case, power dissipation is at most proportional to the number of argument/result lines,* rather than to the number of logic gates (as in ordinary computers), and is thus independent of the "complexity" of the function being computed. This constitutes the central result of the present paper.

8. Reversible Turing machines

We shall assume the reader to be familiar with the concept of *Turing machine.* From our viewpoint, a Turing machine is a closed, time-discrete dynamical system having three state components, i.e., (a) an infinite *tape,* (*b*) the internal state of a finite automaton called *head,* and (c) a *counter* whose content indicates on which tape square the head will operate next. Let T, H, and C be the sets of tape, head, and counter states, respectively. A Turing machine is *reversible* if its transition function $\tau: T \times H \times C \to T \times H \times C$ is invertible.

It is well known that for every recursive function there exists a Turing machine that computes it, and, in particular, that there exist computation-universal Turing machines. Are these capabilities preserved if one restricts one's attention to the class of *reversible* Turing machines?

The answer to the above question is positive. In fact, in [4] Bennett exhibits a procedure for constructing, for any Turing machine and for certain quite general computation formats, a reversible Turing machine that performs essentially the same computations. (The concept of "reversibility" used by Bennett is weaker than ours, insofar as the transition function and its inverse are defined only on distinguished subsets of tape/head/counter states, and even among these states there are some with no predecessors and others with no successors; however, the transition rules of Bennett's machine can easily be augmented so as to satisfy

*In fact, the free energy that must be supplied is at most that in which the input constants are encoded, and the thermal energy that must be removed is at most that of the garbage outputs. According to Theorem 4.1, the number of constant lines need not be greater than that of result lines, and the number of garbage lines need not be greater than that of argument lines.

our stronger definition. We shall assume this augmenting procedure to have been carried out.)

In order to obtain the desired behavior, Bennett's machine is initialized so that all of the tape is blank except for one connected portion representing the computation's argument, and the head is set to a distinguished "initial" state and positioned by the argument's first symbol. At the end of the computation, i.e., when the head enters a distinguished "terminal" state, the result will appear on the tape alongside with the argument, and the rest of the tape will be blank.

Thus, a number of tape squares that are initially blank will eventually contain the result. These squares fulfill a role similar to that played by the constants/garbage lines in Section 5, in the sense that they provide a sufficient supply of "predictable" input values (blanks) at the beginning of the computation, and collect the required amount of "random" output values (in this case, a copy of the argument—cf. the first row of (4.2)) at the end of the computation.

Moreover, during the computation a number of originally blank tape squares may be written over and eventually erased. These squares fulfill a role similar to that played by the temporary-storage lines in Section 5 (cf. the discussion centered on Figure 5.4).

It is clear that, like the constants in the reversible combinational networks of Section 5, the blanks in Bennett's machine play an essential role in the computation, since without their presence one could not achieve *universality* and *reversibility* at the same time. Intuitively, computation in reversible systems requires *a higher degree of "predictability" about the environment's initial conditions* than computation in nonreversible ones.

Owing to the "hybrid" nature of a Turing machine, which involves several kinds of computing primitives (an active device, viz., the head, a passive device, viz., the tape, and mechanisms for head-tape interaction and for head movement), it is difficult to discuss the realizability of a Turing machine by means of a reversible, distributed computing mechanism before having expressed the above primitives in terms of fewer and more elementary ones. For this reason, we shall deal with this problem at the end of the next section, when suitable tools will have been introduced.

9. Reversible cellular automata

We shall assume the reader to have some familiarity with the concept of *cellular automaton* (cf. [21] for details and references). From a physical viewpoint, cellular automata are in many respects more satisfactory models of computing

processes than Turing machines; in particular, they allow one to represent by means of the same mathematical machinery and at the same level of abstraction not only a computer proper but also its environment, the interface with its "users," and the "users" themselves[22]. A cellular automaton is in essence a k-dimensional ($k \geq 1$) array of identical, uniformly interconnected finite automata, and constitutes a closed, time-discrete dynamical system whose state set is a countable Cartesian product of finite sets and whose transition function is continuous (in the topological sense) and commutes with the translations (i.e., with the elements of the array's symmetry group). A cellular automaton is *reversible* if its transition function is invertible.

It is well known that there exist cellular automata that are computation- and construction-universal. Are these capabilities preserved if one restricts one's attention to the class of *reversible* cellular automata? The answer to the above question is positive. In fact, in [20] Toffoli exhibits a procedure for constructing, for *any* cellular automaton (presented as an infinite, space-iterative sequential network), a reversible cellular automaton that realizes it. (Note that until then the existence of computation- and construction-universal reversible cellular automata—which is thus established—had been doubted or, by erroneous arguments, outright denied.)

Assuming the original cellular automaton $\hat{M}$ to be *one-dimensional* and with the von Neumann neighborhood (Figure 9.1a), with Toffoli's construction one obtains a *two-dimensional* cellular automaton M in which the flow of information between the elements of the array (or *cells*) can be visualized as in Figure 9.1b.

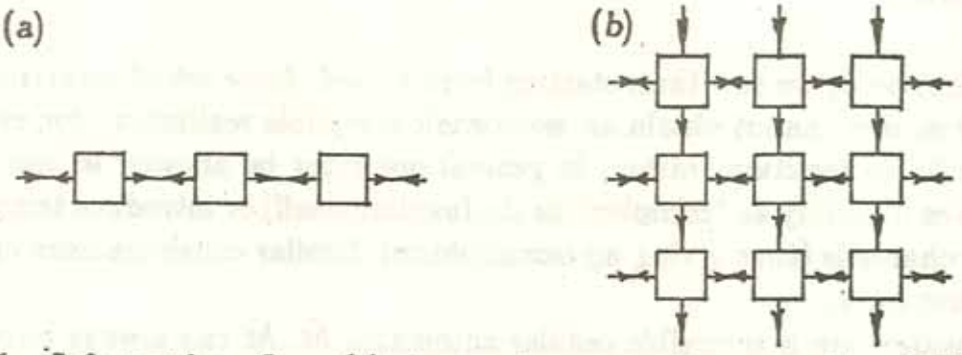


FIG. 9.1 *Information flow (a) in a given one-dimensional cellular automaton $\hat{M}$, and (b) in the corresponding two-dimensional, reversible realization M.*

Each row of M constitutes a reversible (infinite) automaton which is, of course, open. Let us consider a particular row ρ. In analogy with Figure 7.3, let us interpret the arrows entering a row from above (in Figure 9.1) as source lines,

and the ones leaving it from below as *sink* lines. By construction, when the source lines are fed with particular constant values (all 0's) and the values on the sink lines are ignored, the "row" automaton will behave as $\hat{M}$. In order for M to constitute a reversible realization of $\hat{M}$, one must guarantee that ρ's source lines are fed with 0's at all time steps $t \geq 0$. But, by construction, if a row is initialized to a distinguished *blank* state and its source lines are fed with 0's, then the row will remain in the blank state and its sink lines will produce 0's. Thus, if all of the rows that lie above ρ are initialized to the blank state (Figure 9.2), an inexhaustible supply of 0's will "rain down" into ρ. At the same time, ρ's sink lines will produce garbage that will spread down and sideways through the rows below ρ, while ρ itself will reproduce $\hat{M}$'s behavior.

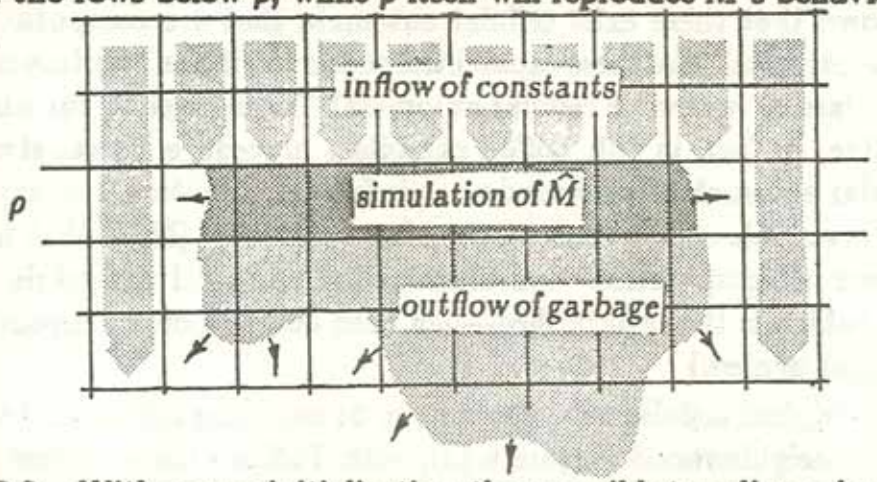


FIG. 9.2 *With proper initialization, the reversible two-dimensional cellular automaton M realizes the one-dimensional automaton $\hat{M}$, which may be irreversible.*

In Section 5, we saw that, starting from a fixed, finite set of invertible logic primitives, one cannot obtain an *isomorphic* invertible realization for every invertible finite function; rather, in general one must be allowed to use ad-hoc primitives (possibly as "complex" as the function itself) or introduce temporary-storage channels (thus giving up isomorphism). Similar considerations apply to finite automata.

Consider now a reversible cellular automaton $\hat{M}$. $\hat{M}$ can always be realized as a reversible sequential network *with temporary storage*. Is it possible to construct an *isomorphic* reversible realization of $\hat{M}$ if one is given a free hand in the choice of primitives? Surprisingly, the answer to this question is negative (Theorem 9.1 below).

LEMMA 9.1. *Let m be the number of binary state variables associated with each cell of a cellular automaton $\hat{M}$, and n the size of its neighborhood. A neces-*

sary condition for the isomorphic realizability of $\hat{M}$ as a reversible sequential network is that $m \geq n$.

Proof. We shall sketch the proof for a one-dimensional cellular automaton $\hat{M}$ with $m = 1$, $n = 2$, and neighborhood index $\langle 0, -1 \rangle$ (i.e., the next state of cell i will depend on the current state of cell i itself and of cell $i - 1$, that is, its "left" neighbor . Assume that the required realization M exists; it will have the form of Figure 9.3a.

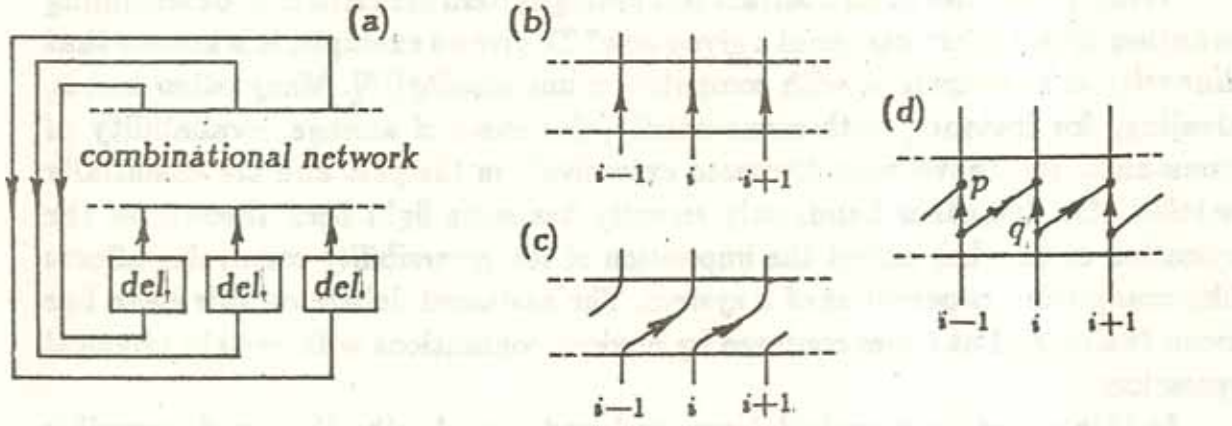


FIG. 9.3 *(a) Overall structure of the (assumed) isomorphic reversible realization of $\hat{M}$. Details of the combinational part of such realization: (b) paths from each cell to itself, (c) paths from each cell to its right neighbor, and (d) nodes implied by the existence of such paths.*

Since each cell is "affected" by itself, there will be paths in the combinational part of M as in Figure 9.3*b*; similarly, since each cell is "affected" by its left neighbor, there will be paths as in Figure 9.3c. Thus, there will be nodes as in Figure 9.3*d*. These nodes cannot stand for invertible primitives (note fan-in at p and fan-out at q) unless further arcs exist (namely, an additional arc entering p and one leaving q). In turn, these arcs imply the existence of new nodes—and so on *ad infinitum*—since connecting these arcs to one another would violate the partial ordering ("causality") associated with function composition). Clearly, it is impossible to complete the construction without introducing either cycles or paths of infinite length.∎

THEOREM 9.1. *There exist reversible cellular automata that do not admit of an isomorphic reversible realization.*

Proof. In [2], Amoroso and Patt exhibit reversible cellular automata (more

amply discussed in [21]) with $m = 1$ and $n = 4$. The thesis follows from Lemma 9.1.∎

Finally, it is well known that any Turing machine can be embedded in a suitable cellular automaton. Thus, according to the foregoing discussion, any Turing machine can be realized by an infinite *reversible* sequential nework.

10. Conclusions and perspectives

What properties of an abstract computing system are critical in determining whether or not it can carry out a given task? To give an example, it is known that *linearity* is incompatible with computation universality[15]. Many other issues, dealing, for instance, with monotonicity, finiteness of storage, availability of constants, etc., have been discussed extensively in the past and are essentially settled. On the other hand, only recently has some light been thrown on the question of to what extent the imposition of the *reversibility* constraint affects the computing capabilities of a system. The sustained debate on this issue has been fueled and at times confused by obvious connections with certain physical questions.

In this paper, we have laid down the foundations for the *theory of reversible computing*, i.e., a theory of computing based on invertible primitives and composition rules that preserve invertibility, and we have shown that this theory can deal satisfactorily with both the functional and the structural aspects of computing processes. In particular, those structures that constitute the traditional paradigms of the theory of computing, such as combinational and sequential networks, finite automata, Turing machines, and cellular automata, have been ascertained to have reversible counterparts of equal computing power. (Analogous considerations apply to systems of difference equations, as will be shown in a later paper.) Thus, the choice to use reversible mechanisms in describing computing processes is a viable one. What can be gained from this choice?

In the synthesis of an abstract computing system for a given task, the requirement that the system be reversible can in general be met only at the cost of greater structural complexity. In other words, one may need more (but not many more) gates and wires. However, the system's very reversibility promises to be a key factor in leading to a more efficient physical realization, since, at the microscopic level, the "primitives" and the "composition rules" available in the physical world resemble much more closely those used in the theory of reversible computing than those used in traditional logic design. For example, it appears that conservative logic might be completely modeled by processes of elastic collision between identical particles[7].

Besides questions of efficient implementation of computing systems, there are several other research lines to which the concepts of reversible computing may give a substantial contribution. Here, we shall limit ourselves to naming a few on which some preliminary investigations have been carried out, such as error detection; numerical analysis and, in particular, the integration of differential equations; number theory and the factoring of large integers; second-order automata and state-reduction techniques; synchronization of computing processes; and cellular automata and the modeling of fundamental physical laws.

The main reason why the concept of reversibility is so pervasive is the following. In every reversible system there are a number of quantities (functions of the system's state) which are *conserved*, i.e., which are constant along each of the system's trajectories (cf. the "integrals of the motion" of theoretical mechanics). Both in mathematical and in physical research, experience has shown that many interesting properties of a system can usually be expressed in terms of conserved quantities, and on this basis one can often make nontrivial statements about the behavior of a system or a class of systems without having to go into a detailed, case-by-case analysis of their operation.

Thus, in the more abstract context of computing, the laws of "conservation of information" may play a role analogous to those of conservation of energy and momentum in physics.

11. Historical and bibliographical notes

The computing element known as the "Fredkin gate" was apparently first described by Quine as an abstract primitive, and was known to Petri[10]. Some conservative (but not reversible) circuits using complementary signal streams were discussed by von Neumann[25] as early as 1952. More recently, Kinoshita and associates[11] worked out a classification of logic elements that "conserve" 0's and 1's; their work, motivated by research in magnetic-bubble circuitry, mentions the possibility of more energy-efficient computation, but has apparently little concern for reversibility.

The AND/NAND element was known to Petri[10], and was mentioned *en passant*, in the context of reversibility arguments, by Landauer[12]. The generalized AND/NAND functions were introduced by Toffoli[20] for proving the universality of reversible cellular automata.

Doubts about the computational capabilities of reversible cellular automata were first expressed by Moore[16], and more formally stated as a conjecture by Burks[5]. Other parties in the debate were Smith[18], Amoroso and Patt[2], Aladyev[1], and Di Gregorio and Trautteur[6].

Distributed systems which are not reversible *per se*, but in which computation-universal, reversible structures can be embedded by proper initialization, were described by Priese[17].

The idea that universal computing capabilities could be obtained from reversible, dissipationless (and, of course, nonlinear) physical circuits apparently first occurred to von Neumann, as reported in a posthumous paper[26]. This idea was also aired by Bennett[4], with some support from his construction of reversible Turing machines. A more formal proof of the physical realizability of such circuits was given by Toffoli[23], and some concrete though not yet entirely practical approaches are outlined in [8]. A much more promising approach, based on the interaction gate, has been recently suggested by Fredkin[7].

Owing to the interdisciplinary nature of the subject, it is quite difficult to locate, evaluate, and credit all research results that may somehow be relevant to reversible computing. We shall appreciate any contributions to the above bibliographical notes.

Acknowledgments

As acknowledged in the text, many ideas discussed in the present paper were originated by Prof. Edward Fredkin. Aside from this, I wish to thank him for much useful advice and encouragment.

List of references

[1] Aladyev, Viktor, "Some Questions Concerning Nonconstructibility and Computability in Homogeneous Structures," *Izv. Akad. Nauk. Estonian SSR*, Fiz.-Mat. **21** (1972), 210-214.

[2] Amoroso, Serafino, and Patt, Y. N., "Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellation Structures," *J. Comptr. Syst. Sci.* **6** (1972), 448-464.

[3] Barton, Edward, "A Reversible Computer Using Conservative Logic," *6.895 Term Paper*, MIT Dept. of Electr. Eng. Comp. Sci. (1978).

[4] Bennett, C. H., "Logical Reversibility of Computation," *IBM J. Res. Dev.* **6** (1973), 525-532.

[5] Burks, A. W., "On Backwards-Deterministic, Erasable, and Garden-of-Eden Automata," *Tech. Rep. no. 012520-4-T*, Comp. Comm. Sci. Dept., Univ. of Michigan (1971).

[6] Di Gregorio, Salvatore, and Trautteur, Giorgio, "On Reversibility in Cellular Automata," *J. Comptr. Syst. Sci.* 11 (1975), 382-391.

[7] Fredkin, Edward, and Toffoli, Tommaso, "Conservative Logic," (in preparation). Some of the material of this paper is tentatively available in the form of unpublished notes from Prof. Fredkin's lectures, collected and organized by Bill Silver in a *6.895 Term Paper*, "Conservative Logic," MIT Dept. of Electr. Eng. Comp. Sci. (1978).

[8] Fredkin, Edward, and Toffoli, Tommaso, "Design Principles for Achieving High-Performance Submicron Digital Technologies," *Proposal to DARPA*, MIT Lab. for Comp. Sci. (1978).

[9] Hennie, C. H., *Iterative Arrays of Logical Circuits*, John Wyley and Sons (1961).

[10] Holt, Anatol, Personal communication (1979).

[11] Kinoshita, Kozo, *et al.*, "On Magnetic Bubble Circuits," *IEEE Trans. Computers* C-25 (1976), 247-253.

[12] Landauer, Rolf, "Irreversibility and Heat Generation in the Computing Process," *IBM J.* 5 (1961), 183-191.

[13] Landauer, Rolf, "Fundamental Limitations in the Computational Process," *Tech. Rep. RC 6048*, IBM Thomas J. Watson Res. Center (1976).

[14] Mackey, G. W., *Mathematical Foundations of Quantum Mechanics*, W. A. Benjamin (1963).

[15] Meyer, J. D., and Zeigler, B. P., "On the Limits of Linearity," *Theory of Machines and Computation*, 229-242, Academic Press (1971).

[16] Moore, E. F., "Machine Models of Self-Reproduction," *Proc. Symp. Appl. Math.* (Amer. Math. Soc. 14 (1962), 17-33.

[17] Priese, Lutz, On a Simple Combinatorial Structure Sufficient for Sublying Nontrivial Self-Reproduction, *J. Cybernetics* 6 (1976), 101-137.

[18] Smith, A. R. III, "Cellular Automata Theory," *Tech. Rep. no. 2*, Stanford Electr. Lab., Stanford Univ. (1969).

[19] Sutherland, I. E., and Mead, C. M., "Microelectronics and Computer Science," *Scientific American* 237:3 (Sept. 1977), 210-262.

[20] Toffoli, Tommaso, "Computation and Construction Universality of Reversible Cellular Automata," *J. Comput. Syst. Sci.* 15 (1977), 213-231.

[21] Toffoli, Tommaso, "Cellular Automata Mechanics" (Ph. D. Thesis), *Tech. Rep. no. 208*, Logic of Computers Group, Univ. of Michigan (1977).

[22] Toffoli, Tommaso, "The Role of the Observer in Uniform Systems," *Applied General Systems Research* (ed. G. J. Klir), 395–400 (Plenum

Press, 1978).

[23] TOFFOLI, Tommaso, "Bicontinuous Extensions of Invertible Combinatorial Functions," *Tech. Memo MIT/LCS/TM-124*, MIT Lab. for Comp. Sci. (1979) (to appear in *Math. Syst. Theory*).

[24] TURING, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proc. London Math. Soc.*, ser. 2, 43 (1936), 544-546.

[25] VON NEUMANN, John, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Automata Studies* (edited by C. E. Shannon and J. McCarthy), 43-98, Princeton Univ. Press (1956).

[26] WIGINGTON, R. L., "A New Concept in Computing," *Proc. IRE* 47 (1961), 516-523.