C Reversible computing

C.1 Reversible computing as a solution

Notice that the key quantity F_{E} in Eqn. II.1 depends on the energy *dissipated* as heat.[15] The $100k_{\mathrm{B}}T$ limit depends on the energy in the signal (necessary to resist thermal fluctuation causing a bit flip). Although most common computing operations must dissipate a minimum amount of energy (as given by VNL), there is nothing that says that information processing has to be done this way. There is no need to dissipate energy, and an arbitrarily large amount of it can be recovered for future operations ("arbitrary" in the sense that there is no inherent physical lower bound on the energy that must be dissipated and cannot be recovered). Accomplishing this becomes a matter of precise energy *management*: moving it around in different patterns, with as little dissipation as possible. Indeed, E_{sig} can be increased to improve reliability, provided we minimize dissipation of energy. This goal can be accomplished by making the computation *logically reversible* (i.e., each successor state has only one predecessor state).

All fundamental physical theories are Hamiltonian dynamical systems, and all such systems are time-reversible. That is, if $\psi(t)$ is a solution, then so is $\psi(-t)$. That is, in general, *physics is reversible*. Therefore, physical information cannot be lost, but we can lose track of it. This is entropy: "unknown information residing in the physical state." Note how this is fundamentally a matter of *information* and *knowledge*: processes are irreversible because information becomes inaccessible. Entropy is ignorance.

To avoid dissipation, don't erase information. The problem is to keep track of information that would otherwise be dissipated, to avoid squeezing information out of logical space (IBDF) into thermal space (NIBDF). This is accomplished by making computation *logically reversible* (it is already *physically* reversible). In effect, computational information is rearranged and recombined *in place*. (We will see lots of examples of how to do this.)

C.1.a Information Mechanics

In 1970s, Ed Fredkin, Tommaso Toffoli, and others at MIT formed the Information Mechanics group to the study the physics of information. As we will

[15] This section is based on Frank (2005b).

see, Fredkin and Toffoli described computation with idealized, perfectly elastic balls reflecting off barriers. The balls have minimum dissipation and are propelled by (conserved) momentum. The model is unrealistic but illustrates many ideas of reversible computing. Later we will look at it briefly (Sec. C.7). They also suggested a more realistic implementation involving "charge packets bouncing around along inductive paths between capacitors." Richard Feynman (Caltech) had been interacting with Information Mechanics group, and developed "a full quantum model of a serial reversible computer" (Feynman, 1986).

Charles Bennett (1973) (IBM) first showed how any computation could be embedded in an equivalent reversible computation. Rather than discarding information (and hence dissipating energy), it keeps it around so it can later "decompute" it back to its initial state. This was a theoretical proof based on Turing machines, and did not address the issue of physical implementation.

Bennett (1982) suggested *Brownian computers* (or *Brownian motion machines*) as a possible physical implementation of reversible computation. The idea is that rather than trying to *avoid* randomization of kinetic energy (transfer from mechanical modes to thermal modes), perhaps it can be *exploited*. This is an example of *respecting the medium* in embodied computation. A Brownian computer makes logical transitions as a result of thermal agitation. However, because it is operating at thermal equilibrium, it is about as likely to go backward as forward; it is essentially conducting a random walk, and therefore can be expected to take $\Theta(n^2)$ time to advance n steps from its initial state. A small energy input — a very weak external driving force — biases the process in the forward direction, so that it precedes linearly, but still very slowly. This means we will need to look at the relation between energy and computation speed (Sec. C.1.b). DNA polymerization provides an example. We can compare its energy dissipation, about $40k_{\mathrm{B}}T$ (~ 1 eV) per nucleotide, with its rate, about a thousand nucleotides per second.

Bennett (1973) also described a chemical Turing machine, in which the tape is a large macromolecule analogous to RNA. An added group encodes the state and head location, and for each transition rule there is a hypothetical enzyme that catalyzes the state transition. We will look at molecular computation in much more detail later in the class.

As in ballistic computing, Brownian computing needs logical reversibility. With no driving force, it is equally likely to move forward or backward, but any driving force will ensure forward movement. Brownian computing can

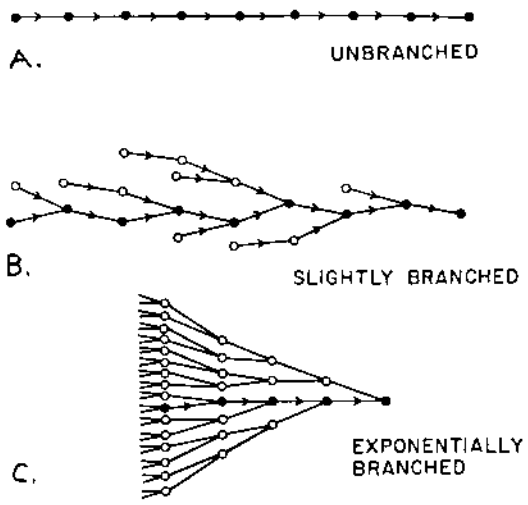

Figure II.7: Different degrees of logical reversibility. [from Bennett (1982)]

accommodate a small degree of irreversibility (see Fig. II.7). In these cases, there are a few backward detours (e.g., it might be equally likely to go in one forard path or two backward paths), but the process can still be biased forward. For forward computation on such a tree "the dissipation per step must exceed kT times the log of the mean number of immediate predecessors to each state" (Bennett, 1982, p. 923). Brownian computation cannot accommodate exponentially backward branching trees, since the computer will spend much more of its time going backward than going forward, and since one backward step is likely to lead to even more backward steps. These undesired states may outnumber desired states by factors of 2^{100}, requiring driving forces on the order of $100kT$. Why 2^{100}? Think of the number of possible predecessors to a state that does something like `x := 0`; assigning to a 64-bit variable has 2^{64} possible predecessor states. Consider also the number of ways of getting to the next statement after a loop. Next we consider the relation between energy dissipation and computation speed.

C.1.b Energy coefficient

We have seen that a greater driving force can lead to faster computation, therefore we define an *energy coefficient* that relates energy dissipation to

computation speed (Frank, 2005b). Let E_{diss} be the energy dissipated per operation and f_{op} be the frequency of operations; then the energy coefficient is defined:

$$c_{\text{E}} \stackrel{\text{def}}{=} E_{\text{diss}}/f_{\text{op}}.$$

For example, for DNA, $c_{\text{E}} \approx (40kT)/(1\text{kHz}) = 40\times 26$ meV/kHz ≈ 1 eV/kHz (since at room temperature, $k_{\text{B}}T \approx 26$ meV: see Sec. A, p. 30). If our goal, however, is to operate at GHz frequencies ($f_{\text{op}} \approx 10^9$) and energy dissipation below $k_{\text{B}}T$ (which is below VNL, but possible for reversible logic), then we need energy coefficients vastly lower than DNA. This is an issue, of course, for molecular computation.

C.1.c Adiabatic circuits

Since the 1980s, and especially in the 1990s there has been work in *adiabatic circuits*. An *adiabatic process* takes place without input or dissipation of energy, and *adiabatic circuits* minimize energy use by obeying certain circuit design rules. For example: (1) Never turn on a transistor when there is a voltage potential between the source and drain. (2) Never turn off a transistor when current is flowing through it. "[A]rbitrary, pipelined, sequential logic could be implemented in a fully-reversible fashion, limited only by the energy coefficients and leakage currents of the underlying transistors." As of 2004, about $c_{\text{E}} = 3$ meV/kHz was achievable, which is about 250× less than DNA.

> "It is difficult to tell for certain, but a wide variety of post-transistor device technologies have been proposed ... that have energy coefficients ranging from 10^5 to 10^{12} times lower than present-day CMOS! This translates to logic circuits that could run at GHz to THz frequencies, with dissipation per op that is still less (in some cases orders of magnitude less) than the VNL bound of $k_{\text{B}}T \ln 2$ ... that applies to all irreversible logic technologies. Some of these new device ideas have even been prototyped in laboratory experiments [2001]." (Frank, 2005b, p. 388)

Frank (2005b, p. 388) notes, "fully-reversible processor architectures [1998] and instruction sets [1999] have been designed and implemented in silicon." Reversible circuit design is, however, outside of the scope of this book.

C.2 Foundations of Conservative Computation

If we want to avoid the von Neumann-Landauer limit, then we have to do reversible computation (we cannot throw logical information away). Moreover, if we want to do fast, reliable computation, we need to use driving forces and signal levels well above this limit, but this energy cannot be dissipated into the environment. Therefore, we need to investigate a *conservative logic* by which energy and other resources are conserved.[16] What this means is that the mechanical modes (computation) must be separated from the thermal modes (heat) to minimize damping and fluctuation and the consequent thermalization of information (recall Sec. B.2).

According to Fredkin & Toffoli (1982), "Computation is based on the storage, transmission, and processing of discrete signals." They outline several physical principles implicit in the axioms of conventional *dissipative logic*:

P1 "The speed of propagation of information is bounded." That is, there is no action at a distance.

P2 "The amount of information which can be encoded in the state of a finite system is bounded." This is ultimately a consequence of thermodynamics and quantum theory.

P3 "It is possible to construct macroscopic, dissipative physical devices which perform in a recognizable and reliable way the logical functions AND, NOT, and FAN-OUT." This is a simple empirical fact (i.e., we build these things).

Since only macroscopic systems are irreversible, as we go to the microscopic level, we need to understand *reversible logic*. This leads to new physical principles of computing:

P4 "Identity of transmission and storage." From a relativistic perspective, information storage in one reference frame may be information transmission in another. For an example, consider leaving a note on a table in an airplane. In the reference frame of the airplane, it is information storage. If the airplane travels from one place to another, then in the reference plane of the earth it is information transmission.

[16] This section is based primarily on Fredkin & Toffoli (1982).

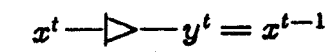

Figure II.8: Symbol for unit wire. (Fredkin & Toffoli, 1982)

P5 "Reversibility." This is because microscopic physics is reversible. Therefore, our computational primitives will need to be invertible.

P6 "One-to-one composition." Physically, fan-out is not trivial (even in conventional logic), so we cannot assume that one function output can be substituted for any number of input variables. Copying a signal can be complicated (and, in some cases, impossible, as in quantum computing). We have to treat fan-out as a specific signal-processing element.

P7 "Conservation of additive quantities." It can be shown that in a reversible systems there are a number of independent conserved quantities, and in many systems they are *additive* over the subsystems, which makes them more useful. Emmy Noether (1882–1935) proved a famous theorem: that any symmetry has a corresponding conservation law, and vice versa; that is, there is a one-to-one correspondence between physical invariances and conserved quantities. In particular, time invariance corresponds to conservation of energy, translational invariance corresponds to conservation of linear momentum, and rotational invariance corresponds to conservation of angular momentum. Conservative logic has to obey at least one additive conservation law.

P8 "The topology of space-time is locally Euclidean. "Intuitively, the amount of 'room' available as one moves away from a certain point in space increases as a power (rather than as an exponential) of the distance from that point, thus severely limiting the connectivity of a circuit."

We will see that two primitive operations are sufficient for conservative logic: the *unit wire* and the *Fredkin gate*.

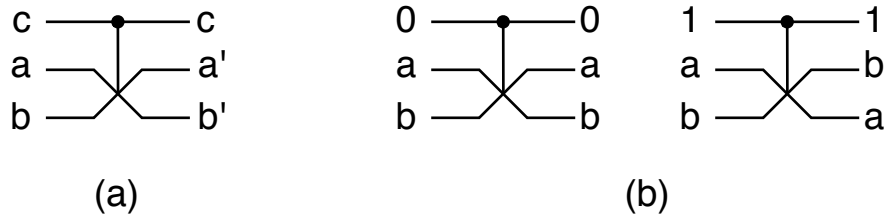

Figure II.9: Fredkin gate or CSWAP (conditional swap): (a) symbol and (b) operation.

C.2.a Unit wire

The basic operation of information storage/transmission is the *unit wire*, which moves one bit of information between two space-time points separated by one unit of time (Fig. II.8). The input value at time t, which is considered the wire's *state* at time t, becomes the output value at time $t + 1$. The unit wire is reversible and conservative (since it conserves the number of 0s and 1s in its input). (Note that there are mathematically reversible functions that are not conservative, e.g., Not.)

C.2.b Fredkin gate

A *conservative logic gate* is a Boolean function that is both invertible and conservative (preserves the number of 0s and 1s). Since the number of 1s and 0s is conserved, conservative computing is essentially *conditional rerouting*, that is, the initial supply of 0s and 1s is rearranged. Conventional models of computation are based on *rewriting* (e.g., Turing machines, the lambda calculus, register machines, term-rewriting systems, Post and Markov productions), but we have seen that overwriting dissipates energy (and is thus non-conservative). In conservative logic we *rearrange* bits without creating or destroying them. There is no infinite "bit supply" and no "bit bucket." In the context of the physics of computation, these are physically real, not metaphors!

A swap is the simplest operation on two bits, and the *Fredkin gate*, which is a conditional swap operation (also called CSWAP), is an example of a conservative logic operation on three bits. It is defined:

$$(0, a, b) \quad \mapsto \quad (0, a, b),$$

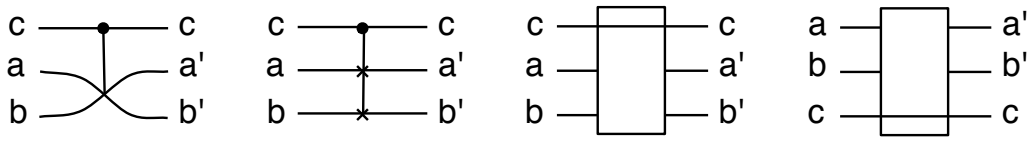

Figure II.10: Alternative notations for Fredkin gate.

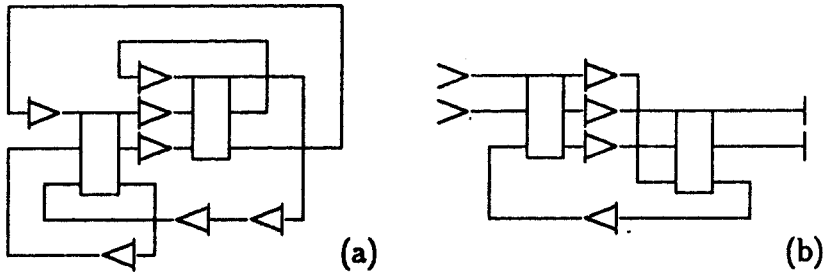

Figure II.11: "(a) closed and (b) open conservative-logic circuits." (Fredkin & Toffoli, 1982)

$$(1, a, b) \quad \mapsto \quad (1, b, a).$$

The first input is a *control* signal and the other two are *data* or *controlled* signals. Here, 1 signals a swap, but Fredkin's original definition used 0 to signal a swap. See Fig. II.9 and Fig. II.12(a) for the operation of the Fredkin gate; Fig. II.10 shows alternative notations. Check that the Fredkin gate is reversible and conservative. As we will see, the Fredkin gate is a universal Boolean primitive for conservative logic.

C.3 Conservative logic circuits

A *conservative-logic circuit* is a directed graph constructed from conservative logic gates connected by wires (see Fig. II.11 for examples). We can think of the gates as instantaneous and the unit wire as being a unit delay, of which we can make a sequence (or imagine intervening identity gates). A *closed circuit* is a *closed* (or *isolated*) physical system, whereas an *open circuit* has external inputs and outputs. The number of outputs must equal the number of inputs, or the circuit will not be reversible. An open circuit may be part of a larger

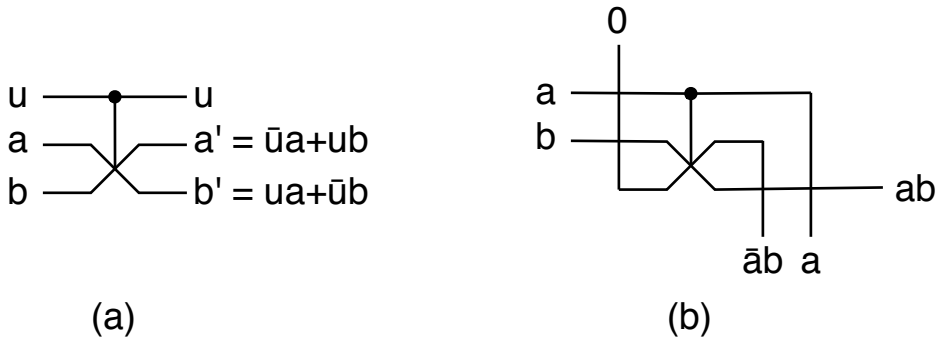

Figure II.12: (a) Logical behavior of Fredkin gate. (b) Implementation of AND gate by Fredkin gate by constraining one input to 0 and discarding two "garbage" outputs.

conservative circuit, or connected to the environment. A conservative-logic circuit is a *discrete-time dynamical system*, that is, it computes in discrete steps in each of which bits move through the gates and unit wires. The number N of unit wires in the circuit is its number of degrees of freedom (specifically, IBDF). The numbers of 0s and 1s at any time is conserved, $N = N_0 + N_1$.

C.4 Universality

Fig. II.12(b) illustrates how the Fredkin gate can be used to implement AND; other conventional gates, such as NOT, OR, and FAN-OUT can be implemented similarly. (You will show this in Exercises II.4 to II.5). Notice that the implementation of AND requires that we provide a constant 0 on the second data line, and the Fredkin gate produces two "garbage bits" whose values ($\overline{a}b$ and a) we might not need. Fig. II.14 shows a more complicated example, a 1-line to 4-line demultiplexer. Depending on the value of the address bits A_1A_0 it will direct the input X to Y_0, Y_1, Y_2, or Y_3. The circuit requires three constant 0 inputs and produces two garbage bits in addition the the desired outputs.

As a consequence, you can convert conventional logic circuits (constructed from AND, OR, NOT, etc.) into conservative circuits, but the process is not very efficient, because of the need for many ancillary (constant input) and garbage bits. It's better to design the conservative circuit from scratch. Nev-

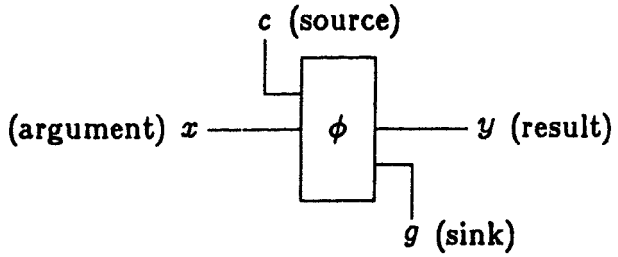

Figure II.13: "Realization of f by ϕ using source and sink. The function $\phi : (c, x) \mapsto (y, g)$ is chosen so that, for a particular value of $c, y = f(x)$." (Fredkin & Toffoli, 1982)

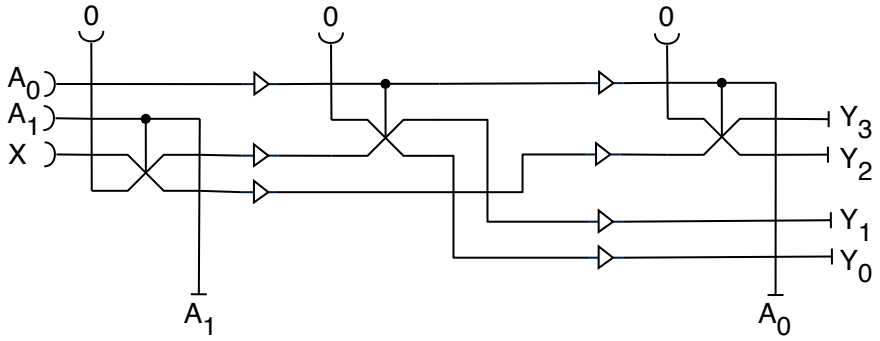

Figure II.14: 1-line-to 4-line demultiplexer. The address bits $A_1A_0 = 00, 01, 10, 11$ direct the data bit X into Y_0, Y_1, Y_2 or Y_3, respectively. Note that each Fredkin gate uses an address bit to route X into either of two wires. (Adapted from circuit in Fredkin & Toffoli (1982).)

ertheless, this shows that any conventional sequential circuit can be converted into a conservative logic circuit, provided there is a source for constants and a sink for garbage. As a consequence, the unit wire and Fredkin gate are a universal set of conservative operators, since they can be used to implement (for example) AND, NOT, and FAN-OUT, which are universal.

C.5 Constants and garbage

You have seen that the Fredkin gate can be used to compute non-invertible functions such as AND, if we are willing to provide appropriate constants (called "ancillary values") and to accept unwanted outputs. In general, an irreversible function can be embedded in a reversible by providing appropriate constants from a *source* and ignoring some of the outputs, the *sink*, which are considered *garbage* (Fig. II.13). That is, if we want to compute $f : x \mapsto y$, we provide appropriate constants c so that it can be embedded in a conservative computation $\phi : (c, x) \mapsto (y, g)$, which produces the desired output y along with garbage g. However, this garbage cannot be thrown away (which would dissipate energy), so it must be recycled in some way.

C.6 Garbageless conservative logic

To reuse the apparatus for a new computation, we would have to throw away the garbage and provide fresh constants, both of which would dissipate energy. This is a significant problem if dissipative circuits are naively translated to conservative circuits because: (1) the amount of garbage tends to increase with the number of gates, and (2) with the naive translation, the number of gates tends to increase exponentially with the number of input lines. However there is a way to make the garbage about the same size as the input, and thereby limit the dissipated energy.

First observe that a *combinational* conservative-logic network (one with no feedback loops) can be composed with its inverse to consume all the garbage (Fig. II.15). That is, if ϕ converts (c, x) into (y, g), then ϕ^{-1}, its inverse, will convert (y, g) back to (c, x). We can always implement ϕ^{-1} because the unit wire and the Fredkin gate are invertible (in fact, their own inverses). This in itself would not be useful, since in "decomputing" (y, g) back to (c, x) we have lost the result y of the computation. Therefore, observe that the desired output can be extracted by a "spy circuit" (Fig. II.16) interposed on a wire. It works because the Fredkin gate satisfies $(a, 0, 1) \mapsto (a, a, \bar{a})$.

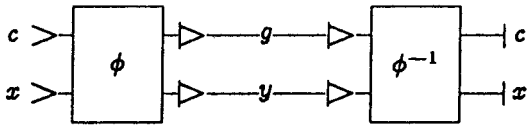

Figure II.15: Composition of combinational conservative-logic network with its inverse to consume the garbage. (Fredkin & Toffoli, 1982)

Figure II.16: The "spy circuit" for tapping into the output. (Fredkin & Toffoli, 1982)

We use this circuit on the bits of y between the computation ϕ and the decomputation ϕ^{-1}. Notice that the spy circuit requires two ancillary bits for each bit that it extracts; it outputs the desired value and its complement (presumably garbage).

We can use these ideas to design a general approach to garbageless computation (Fig. II.17). The desired computation has m input bits, $x_1, x_2, \ldots, x_m$, and n output bits $y_1, \ldots, y_n$. To do it reversibly requires (we suppose) h constants $c_1, \ldots, c_h$ and generates (necessarily) $h + m - n$ garbage bits (for the number of outputs of a reversible computation has to equal the number of inputs). Extracting the output requires the provision of $2n$ new constants and generates the n output bits and their n complements (which can be considered garbage). Initializing the machine for a new computation requires putting in the new input, which will dissipate energy, and restoring the output registers $y\bar{y}$ to their $00\cdots0011\cdots11$ state, which also dissipates energy. Therefore the energy dissipated will be proportional to the size of the input and output (specifically, $m + n$). The ancillary constants are automatically restored by decomputation.

Consider the more schematic diagram in Fig. II.18. Think of arranging tokens (representing 1-bits) in the input registers, both to represent the input

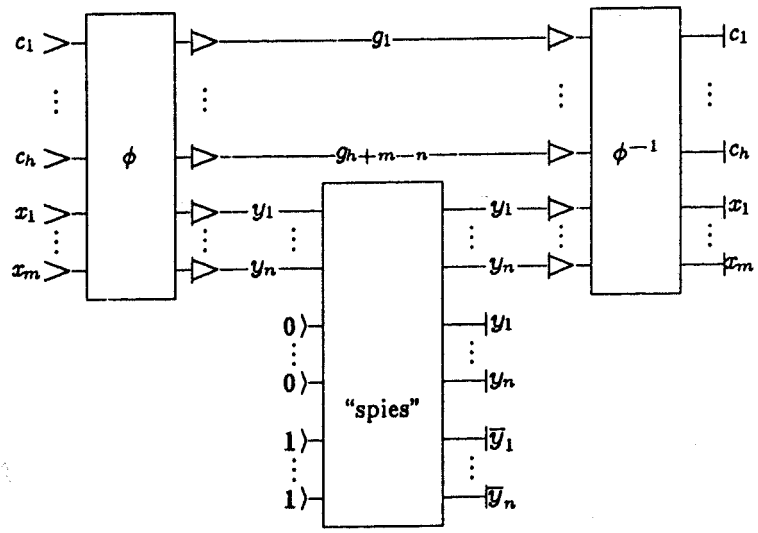

Figure II.17: Garbageless circuit. (Fredkin & Toffoli, 1982)

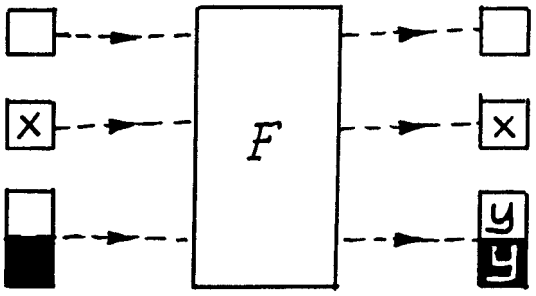

Figure II.18: "The conservative-logic scheme for garbageless computation. Three data registers are 'shot' through a conservative-logic black-box F. The register with the argument, x, is returned unchanged; the clean register on top of the figure, representing an appropriate supply of input constants, is used as a scratchpad during the computation (cf. the c and g lines in Figure [II.17]) but is returned clean at the end of the computation. Finally, the tokens on the register at the bottom of the figure are rearranged so as to encode the result y and its complement $\neg y$" (Fredkin & Toffoli, 1982)

x, but also to provide a supply of n of them in the black lower square. Next, run the computation (including both the forward and backward passes). The backward pass restores the input argument tokens to their initial positions. The $2n$-bit string $00\cdots0011\cdots11$ in the lower register has been rearranged to yield the result and its complement, $y\bar{y}$. Restoring the $0\cdots01\cdots1$ inputs for another computation dissipates energy, but the amount of energy depends on the size of the output (number of bits), not the amount of computation.[17]

C.7 Ballistic computation

"Consider a spherical cow moving in a vacuum..."

To illustrate how conservative computation could dissipate arbitrarily small amounts of energy, Fredkin and Toffoli developed an idealized model of dissipationless *ballistic computation*, often called *billiard ball computation*. It is based on the same assumptions as the classical kinetic theory of gasses: perfectly elastic spheres and surfaces. In this case we can think of pucks on frictionless table.

Fig. II.19 shows the general structure of a billiard ball computer. 1-bits are represented by the presence of a ball at a location, and 0-bits by its absence. Input is provided by simultaneously firing balls into the input ports for the 1s in the argument. Inside the box the balls ricochet off each other and off of fixed reflectors, which performs the computation. After a fixed time delay, the balls emerging (or not) from the output ports define the output. Obviously the number of 1s (balls) is conserved, and the computation is reversible because the laws of motion are reversible.

Since in this idealized model collisions are perfectly elastic, and there is no friction, no energy is dissipated, and the tiniest initial velocities are sufficient for a computation of arbitrary length. Therefore, there is no lower bound on the energy required for the computation. (Computation will go faster, of course, if the balls are shot in at higher velocity.) Since the laws of classical dynamics are reversible, the computation will be reversible, assuming of course that billiard balls can be made to compute at all!

In fact, they can, and Fig. II.20 shows the realization of the computational primitive, the *interaction gate*. If balls representing 1-bits are shot in from the left at p and q, then the balls emerging (or not) on the right will represent

[17] Finite loops can be unrolled, which shows that they can be done without dissipation. (Cf. also that billiard balls can circulate in a frictionless system.)

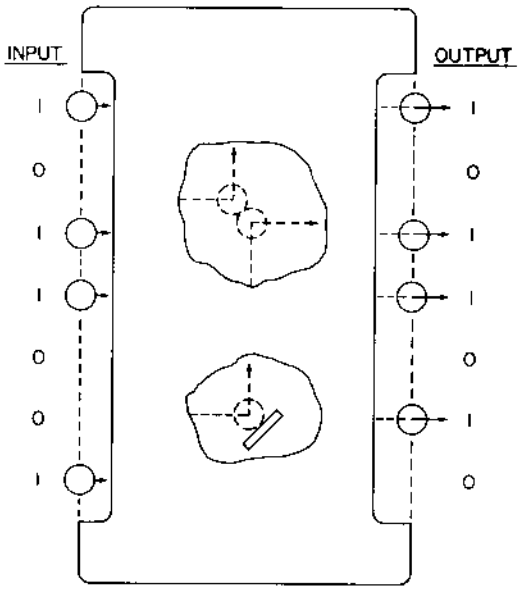

Figure II.19: Overall structure of ballistic computer. (Bennett, 1982)

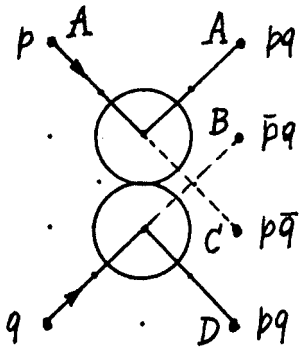

Figure II.20: "Billiard ball model realization of the interaction gate." (Fredkin & Toffoli, 1982)

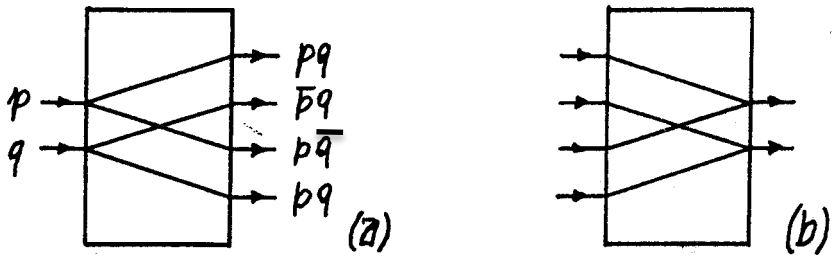

Figure II.21: "(a) The interaction gate and (b) its inverse." (Fredkin & Toffoli, 1982)

four logical possibilities, $p\,q$, $\bar{p}\,q$, $p\,\bar{q}$, and $\bar{p}\,\bar{q}$ (the latter represented by no balls emerging from the gate). (Of course, the gate is conservative: the number of balls entering it has to equal the number exiting.) Notice that the interaction gate is invertible, because if we put in on the right one of the four possible outputs (11, 10, 01, 00), we will get the corresponding input on the left ($p\,q$, $\bar{p}\,q$, $p\,\bar{q}$, $\bar{p}\,\bar{q}$, respectively). Check to make sure you see this (Ex. II.10). Fig. II.21 is a more abstract symbol for the interaction gate and its inverse.

The interaction gate is universal because it can compute both AND and NOT. However, we must make provisions for arbitrary interconnections in a planar grid. Therefore, we need to implement signal *crossover* and to control *timing* so that balls arrive at the correct time in order to interact. In fact, it's only necessary to deal with *non-trivial* crossover, for *trivial* crossover is when two balls cannot possibly be at the same place at the same time. Fig. II.22 shows mechanisms for realizing nontrivial crossover, delays, and direction changes. Notice that the "wires" in this computer are virtual, represented by the possible trajectories of the balls, and not physical objects. For reversible computing, the Fredkin gate is more relevant, and Fig. II.23 shows its realization in terms of multiple interaction gates. (The "bridge" indicates non-trivial crossover.) Since the Fredkin gate is universal, any reversible computation can be implemented on the billiard ball computer.

Of course, the billiard ball computer is an idealized model of computation, and like other abstract models, such as the Turing machine, it has practical limitations (Bennett, 1982). For example, minuscule errors of any sort (position, velocity, alignment) will accumulate rapidly (by about a factor of 2

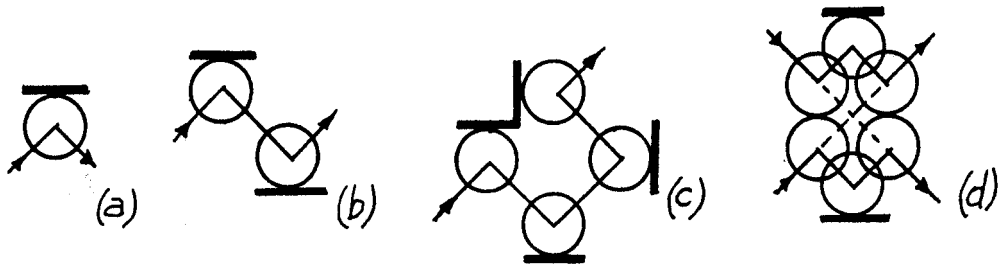

Figure II.22: "The mirror (indicated by a solid dash) can be used to deflect a ball's path (a), introduce a sideways shift (b), introduce a delay (c), and realize nontrivial crossover (d)." (Fredkin & Toffoli, 1982)

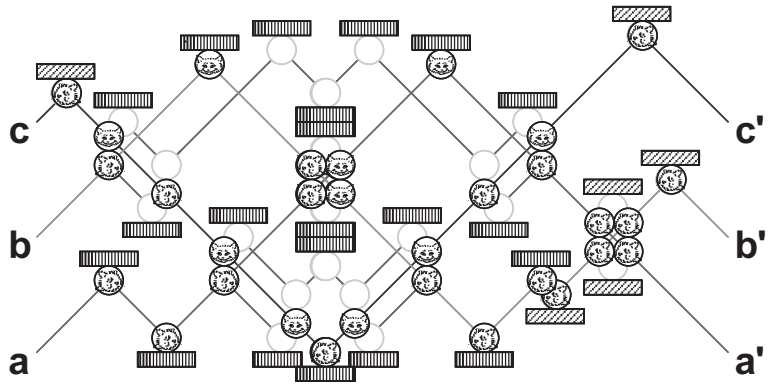

Figure II.23: Realization of the Fredkin gate in terms of multiple interaction gates. [NC]

at each collision). Therefore, an initial random error of $1/10^{15}$ in position or velocity (about what would be expected from Heisenberg uncertainty principle) would lead to a completely unpredictable trajectory after a few dozen collisions, leading to a Maxwell distribution of velocities, as in a gas. That is, errors grow exponentially in the length of a computation. "Even if classical balls could be shot with perfect accuracy into a perfect apparatus, fluctuating tidal forces from turbulence in the atmosphere of nearby stars would be enough to randomize their motion within a few hundred collisions" (Bennett, 1982, p. 910). Various solutions to these problems have been considered, but they all have limitations. Bennett (1982, p. 911) concludes, "In summary, although ballistic computation is consistent with the laws of classical and quantum mechanics, there is no evident way to prevent the signals' kinetic energy from spreading into the computer's other degrees of freedom." Of course, signals can be restored, but this introduces dissipation, and we are back where we began. Nevertheless, ballistic computation, as found in the billiard ball computer, illustrates some of the principles of reversible computing that are used in quantum computation, the topic of the next chapter.