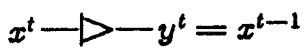

Figure II.10: Symbol for unit wire. (Fredkin & Toffoli, 1982)

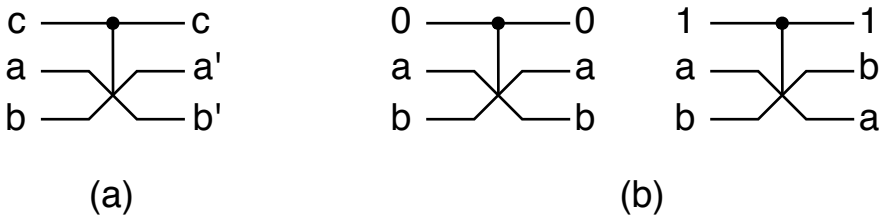

Figure II.11: Fredkin gate or CSWAP (conditional swap): (a) symbol and (b) operation.

C.3 Unit wire

¶1. Information storage in one reference frame may be information transmission in another.
E.g., leaving a note on a table in an airplane (at rest with respect to earth or not, or to sun, etc.).

¶2. The *unit wire* moves one bit of information from one space-time point to another space-time point separated by one unit of time. See Fig. II.10.

¶3. **State:** "The value that is present at a wire's input at time t (and at its output at time $t + 1$) is called the *state* of the wire *at time* t."

¶4. It is invertible 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.4 Fredkin gate

¶1. **Conservative logic gate:** Any Boolean function that is invertible and conservative.

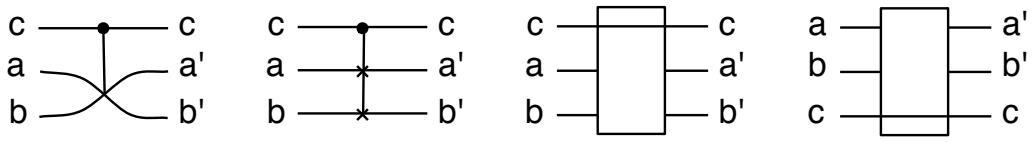

Figure II.12: Alternative notations for Fredkin gate.

¶2. **Conditional rerouting:** Since the number of 1s and 0s is conserved, conservative computing is essentially *conditional rerouting*

¶3. **Rearranging vs. rewriting:** Conventional models of computation are based on *rewriting*
(e.g., TMs, lambda calculus, register machines, term rewriting systems, Post and Markov productions).
But we have seen that overwriting dissipates energy (and is non-conservative).

¶4. In conservative logic we *rearrange* bits without creating or destroying them.
(No infinite "bit supply" and no "bit bucket.")

¶5. **Fredkin gate:** The *Fredkin gate* is a conditional swap operation (also called CSWAP):

$$\begin{aligned}(0, a, b) &\mapsto (0, a, b),\\ (1, a, b) &\mapsto (1, b, a).\end{aligned}$$

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.11 and Fig. II.14. Fig. II.12 shows alternative notations for the Fredkin gate.

¶6. Note that it is reversible and conservative.

¶7. **Universal:** The Fredkin gate is a universal Boolean primitive for conservative logic.

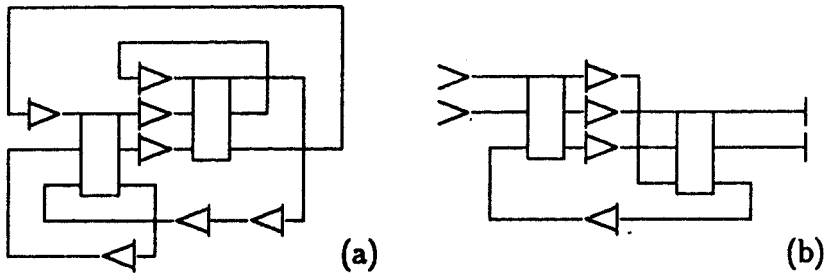

Figure II.13: "(a) closed and (b) open conservative-logic circuits." (Fredkin & Toffoli, 1982)

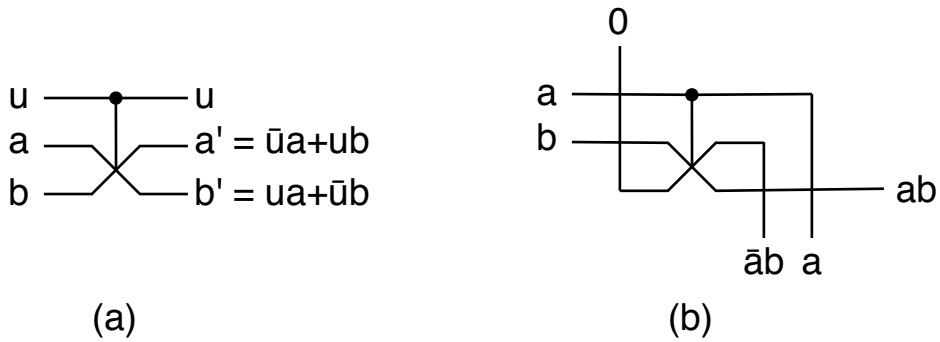

Figure II.14: (a) Logical behavior of Fredkin gate. (b) Implementation of AND gate by Fredkin gate by constraining one input to 0 and discarding two "garbage" outputs.

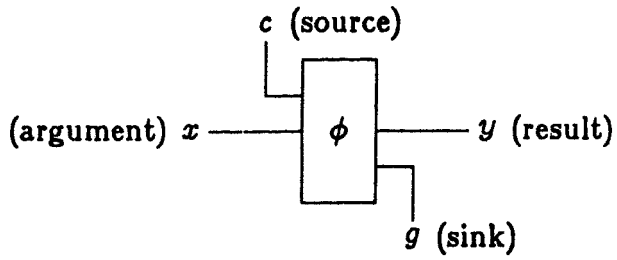

Figure II.15: "Realization of f by ϕ using source and sink. The function $\phi : (c, x) \mapsto (y, g)$ is chosen so that, for a particular value of $c, y = f(x)$." (Fredkin & Toffoli, 1982)

C.5 Conservative logic circuits

¶1. "A conservative-logic circuit is a directed graph whose nodes are conservative-logic gates and whose arcs are wires of any length [Fig. II.13]."

¶2. We can think of the gate as instantaneous and the unit wire as being a unit delay, of which we can make a sequence (or imagine intervening identity gates).

¶3. **Closed vs. open:** A *closed circuit* is a *closed* (or *isolated*) physical system.
An *open circuit* has external inputs and outputs.

¶4. The number of outputs must equal the number of inputs.

¶5. It may be part of a larger conservative circuit, or connected to the environment.

¶6. **Discrete-time dynamical system:** A conservative-logic circuit is a *discrete-time dynamical system.*

¶7. **Degrees of freedom:** The number N of unit wires in the circuit is its number of DoF.
The numbers of 0s and 1s at any time is conserved, $N = N_0 + N_1$.

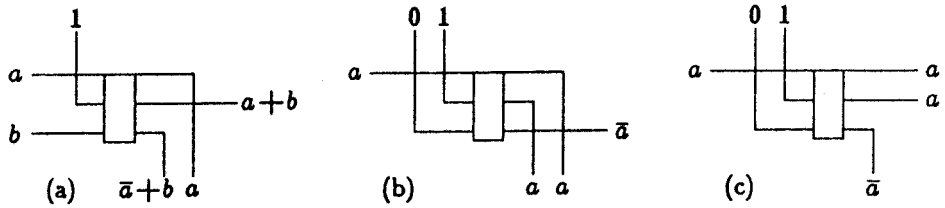

Figure II.16: "Realization of the (a) OR, (b) NOT, and (c) FAN-OUT functions by means of the Fredkin gate." (Fredkin & Toffoli, 1982)

C.6 Constants and garbage

¶1. 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 (see Fig. II.14).

¶2. In general, one function can be embedded in another by providing appropriate constants from a *source* and ignoring some of the outputs, the *sink*, which are considered *garbage*.

¶3. However, this garbage cannot be thrown away (which would dissipate energy), so it must be recycled in some way.

C.7 Universality

¶1. **OR, NOT, and FAN-OUT:** Fig. II.16 shows Fredkin realizations of other common gates.

¶2. **Demultiplexer example:** Fig. II.17 shows a 1-line to 4-line demultiplexer.

¶3. Hence you can convert conventional logic circuits into conservative circuits, but the process is not very efficient. It's better to design the conservative circuit from scratch.

¶4. **Universality:** "any computation that can be carried out by a conventional sequential network can also be carried out by a suitable

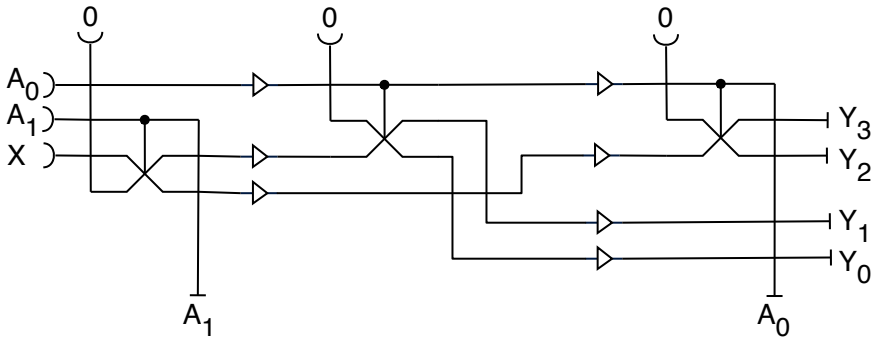

Figure II.17: 1-line-to 4-line demultiplexer. The address bits $A_1A_0 = 00, 01, 10, 11$ direct the data bit X into Y_0, Y_1, Y_2 or Y_3, respectively. Note that each Fredkin gate uses an address bit to route X into either of two wires. (Adapted from circuit in Fredkin & Toffoli (1982).)

---

conservative-logic network, provided that an external supply of constants and an external drain for garbage are available."
(Will see how to relax these constraints: Sec. C.8)

C.8 Garbageless conservative logic

¶1. To reuse the apparatus for a new computation, we will have to throw away the garbage and provide fresh constants, both of which will dissipate energy.

¶2. **Exponential growth of garbage:** 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.
"This is so because almost all boolean functions are 'random', i.e., cannot be realized by a circuit simpler than one containing an exhaustive look-up table."

¶3. However there is a way to make the garbage the same size as the input

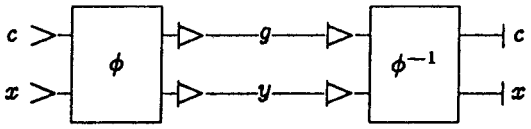

Figure II.18: Composition of combinational conservative-logic network with its inverse to consume the garbage. [fig. from Fredkin & Toffoli (1982)]

Figure II.19: The "spy circuit" for tapping into the output. Note that in the diagram the 0 and 1 constant inputs are switched (or, equivalently, the a and $\overline{a}$ outputs are switched). See also the FAN-OUT circuit in Fig. II.16. [fig. from Fredkin & Toffoli (1982)]

(in fact, identical to it).

¶4. First observe that a *combinational* conservative-logic network (one with no feedback loops) can be composed with its inverse to consume all garbage (Fig. II.18).

¶5. The desired output can be extracted by a "spy circuit" (Fig. II.19).

¶6. Fig. II.20 shows the general arrangement for garbageless computation. This requires the provision of n new constants (n = number output lines).

¶7. Consider the more schematic diagram in Fig. II.21.

¶8. Think of arranging tokens (representing 1-bits) in the input registers, both to represent the input x, but also a supply of n of them in the black lower square.

¶9. Run the computation.

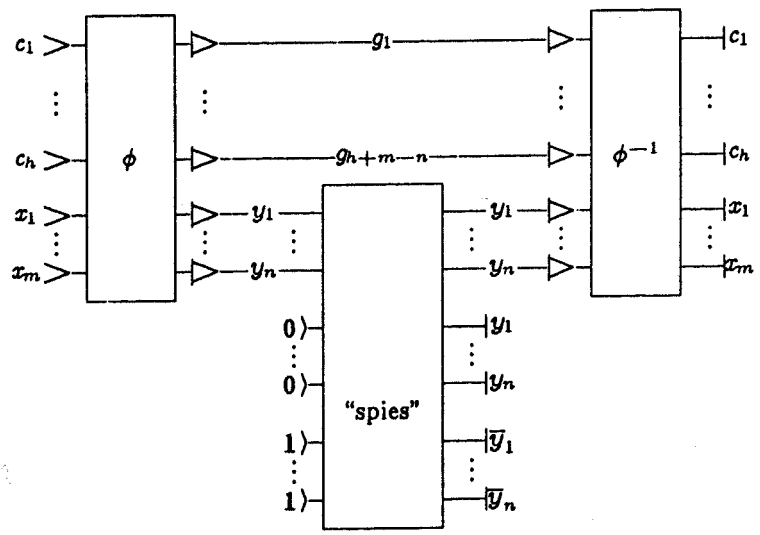

Figure II.20: Garbageless circuit. (Fredkin & Toffoli, 1982)

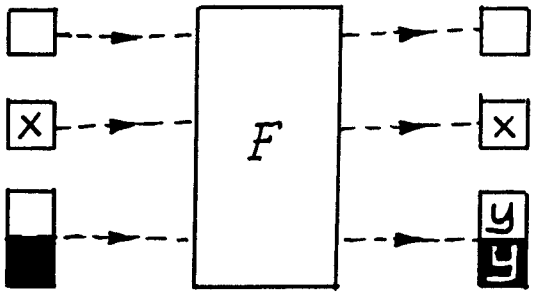

Figure II.21: "The conservative-logic scheme for garbageless computation. Three data registers are 'shot' through a conservative-logic black-box F. The register with the argument, x, is returned unchanged; the clean register on top of the figure, representing an appropriate supply of input constants, is used as a scratchpad during the computation (cf. the c and g lines in Figure [II.20]) 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)

¶10. The input argument tokens have been restored 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 $y\bar{y}$.

¶11. Restoring the $0\cdots01\cdots1$ inputs for another computation dissipates energy.

¶12. **Feedback:** 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.)

C.9 Ballistic computation

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

¶1. **Billiard ball model:** To illustrate dissipationless *ballistic computation*, Fredkin and Toffoli defined a *billiard ball* model of computation.

¶2. 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.

¶3. Fig. II.22 shows the general structure of the billiard ball model.

¶4. 1s are represented by the presence of a ball at a location, and 0s by their absence.

¶5. Input is provided by simultaneously firing balls into the input ports for the 1s in the argument.

¶6. Inside the box the balls ricochet off each other and fixed reflectors, which performs the computation.

¶7. After a fixed time delay, the balls emerging (or not) from the output ports define the output.

¶8. Obviously the number of 1s (balls) is conserved.

¶9. The computation is reversible because the laws of motion are reversible.

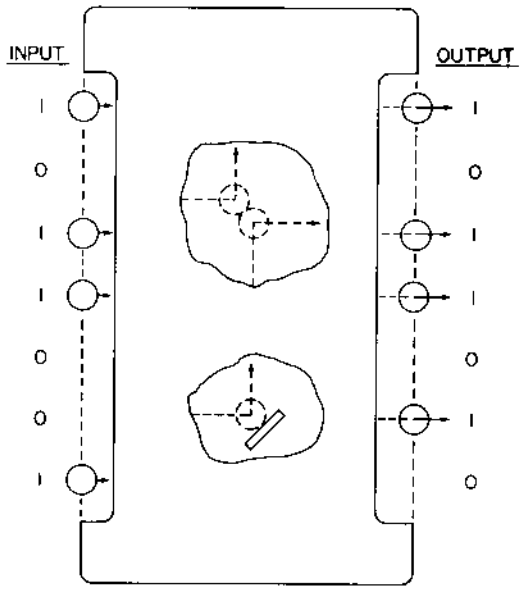

Figure II.22: Overall structure of ballistic computer. (Bennett, 1982)

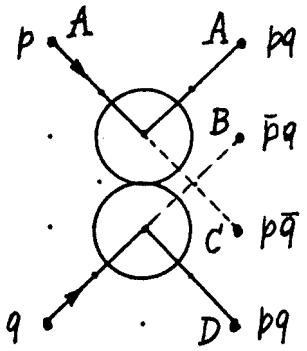

Figure II.23: "Billiard ball model realization of the interaction gate." (Fredkin & Toffoli, 1982)

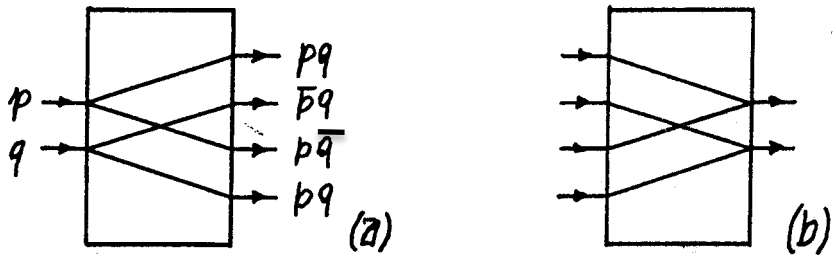

Figure II.24: "(a) The interaction gate and (b) its inverse." (Fredkin & Toffoli, 1982) Note that the second pq from the bottom should be $p\bar{q}$.

¶10. **Interaction gate:** Fig. II.23 shows the realization of the computational primitive, the *interaction gate*.

¶11. Fig. II.24 is the symbol for the interaction gate and its inverse.

¶12. **Universal:** The interaction gate is universal because it can compute both AND and NOT.

¶13. **Interconnections:** However, we must make provisions for arbitrary interconnections in a planar grid. So need to implement signal *crossover* and control *timing*.
(This is *non-trivial* crossover; *trivial* crossover is when two balls cannot possibly be at the same place at the same time.)

¶14. Fig. II.25 shows mechanisms for realizing these functions.

¶15. Fig. II.26 shows a realization of the Fredkin gate in terms of multiple interaction gates. (The "bridge" indicates non-trivial crossover.)

¶16. **Practical problems:** Minuscule errors of any sort (position, velocity, alignment) will accumulate rapidly (by about a factor of 2 at each collision).

¶17. E.g., initial random error of $1/10^{15}$ in position or velocity (about what would be expected from uncertainty principle) would lead to a completely unpredictable trajectory after a few dozen collisions.
It will lead to a Maxwell distribution of velocities, as in a gas.

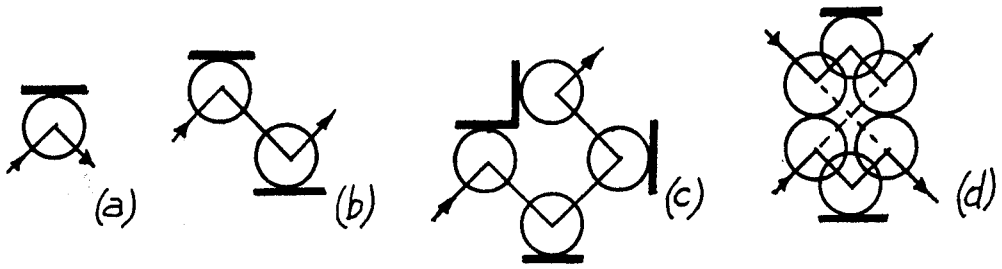

Figure II.25: "The mirror (indicated by a solid dash) can be used to deflect a ball's path (a), introduce a sideways shift (b), introduce a delay (c), and realize nontrivial crossover (d)." (Fredkin & Toffoli, 1982)

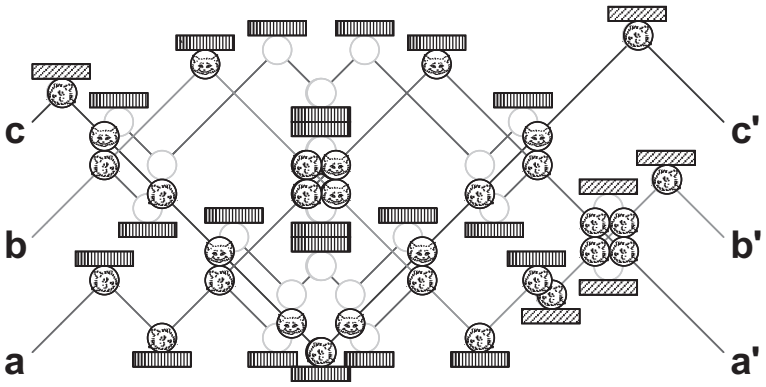

Figure II.26: Realization of the Fredkin gate in terms of multiple interaction gates. [NC]

¶18. "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)

¶19. Various solutions have been considered, but they all have limitations.

¶20. "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." (Bennett, 1982, p. 911)

¶21. Signals can be restored, but this introduces dissipation.

D Sources

Bennett, C. H. The Thermodynamics of Computation — a Review. *Int. J. Theo. Phys.*, 21, 12 (1982), 905–940.

Berut, Antoine, Arakelyan, Artak, Petrosyan, Artyom, Ciliberto, Sergio, Dillenschneider, Raoul and Lutz, Eric. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature* 483, 187–189 (08 March 2012). doi:10.1038/nature10872

Frank, Michael P. Introduction to Reversible Computing: Motivation, Progress, and Challenges. CF '05, May 4–6, 2005, Ischia, Italy.

Fredkin, E. F., Toffoli, T. Conservative logic. *Int. J. Theo. Phys.*, 21, 3/4 (1982), 219–253.