# Foundational principles of reversible and quantum computing

## Reversible Computing

Luca Paolini
paolini@di.unito.it
Università di Torino
Dipartimento di Informatica

October 30, 2013

### Entropy

The information-entropy à la Shannon is the average information content of an information entity (a channel signal, a variable, ...).

**Definition** 1 Let $B$ be a (not necessarily finite) type whose values are labeled $b^1, b^2, \ldots$. Let $\xi$ be a random variable of type $B$ that is equal to $b^i$ with probability $p_i$. The entropy of $\xi$ is defined as $-\sum_i p_i \log p_i$.

**Definition** 2 Consider a function $f : B_1 \to B_2$ where $B_2$ is a (not necessarily finite) type whose values are labeled $b_2^1, b_2^2, , \ldots$. The output entropy of the function is given by $-\sum q_j \log q_j$ where $q_j$ indicates the probability of the output of the function to have value $b_2^j$.

**Definition** 3 We say a function is information-preserving whenever its output entropy is equal to the entropy of its input.

• For instance, let $true, false$ be the values of type $Bool$. Let us consider a variable $\xi$ of type $Bool \times Bool$. The information content of this variable depends on the probability distribution of the four possible $Bool \times Bool$ values. If we have a computational situation in which the pair $(false, false)$ could occur with probability $1/2$, the pairs $(false, true)$ and $(true, false)$ can each occur with probability $1/4$, and the pair $(true, true)$ cannot occur, the information content of $\xi$ would be:

$$1/2 \log 2 + 1/4 \log 4 + 1/4 \log 4 + 0 \log 0 = 1.5 \text{bits of information.}$$

• If, however, the four possible pairs had an equal probability, the same formula would calculate the information content to be 2 bits, which is the maximal amount for a variable of type $Bool \times Bool$ .

• The minimum entropy $0$ corresponds to a variable that happens to be constant with no uncertainty.

Now consider functions.

- Consider the $Bool \to Bool$ function NOT. Let $p_F$ and $p_T$ be the probabilities that the input is $false$ or $true$ respectively. The outputs occur with the reverse probabilities, i.e., $p_T$ is the probability that the output is $false$ and $p_F$ is the probability that the output is $true$. Hence the output entropy of the function is $-p_F \log p_F - p_T \log p_T$ which is the same as the input entropy and the function is information-preserving.
- As another example, consider the $Bool \to Bool$ function $\text{CONST}_T(x) = true$ which discards its input. The output of the function is always $true$ with no uncertainty, which means that the output entropy is $0$, and that the function is not information-preserving.
- As a third example, consider the function AND and let the inputs occur with equal probabilities, viz. let the entropy of the input be 2. The output is false with probability $3/4$ and true with probability $1/4$, which means that the output entropy is about $0.8$ and the function is not information-preserving.
- As a final example, consider the $Bool \to Bool \times Bool$ function fan-out$(x) = (x, x)$ which duplicates its input. Let the input be $false$ with probability $p_F$ and $true$ be probability $p_T$. The output is $(false, false)$ with probability $p_F$ and $(true, true)$ with probability $p_T$ which means that the output entropy is the same as the input entropy and the function is information-preserving.

## Reversible Logic

As we pack more and more logic elements into smaller and smaller volumes and clock them at higher and higher frequencies, we dissipate more and more heat. This creates at least three problems:

- Energy costs money (throughput).
- Portable systems exhaust their batteries (stockpiling).
- Systems overheat (disposal).

When a computational system erases a bit of information, it must dissipate $\ln 2 \times kT$ energy, where $k$ is Boltzmann's constant and $T$ is the temperature.
For $T = 300$ Kelvins (room temperature), this is about $2.9 \times 10^{-}21$ joules (roughly the kinetic energy of a single air molecule at room temperature).

Today's computers erase a bit of information (in the sense used here) every time they perform a logic operation. These logic operations are therefore called "irreversible." This erasure is done very inefficiently, and much more than kT is dissipated for each bit erased.

Reversible Logic is also known as Charge Recovery Logic or Adiabatic Logic.

> "Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics. 'He thought,' as Feynman once put it, 'that he understood paper.' But he was mistaken. Real, quantum-mechanical paper is wildly different from the abstract stuff that the Turing machine uses. The Turing machine is entirely classical, and does not allow for the possibility the paper might have different symbols written on it in different universes, and that those might interfere with one another."

The above quote by David Deutsch, originally stated in the context of quantum computing, stems from the observation that even the most abstract models of computation embody some laws of Physics. Indeed, conventional classical models of computation, including boolean logic, the Turing machine, and the $\lambda$-calculus, are founded on primitives which correspond to irreversible physical processes.
For example, a NAND gate is an irreversible logical operation in the sense that its inputs cannot generally be recovered from observing its output, and so is the operation of overriding a cell on a Turing machine tape with a new symbol, and so is a $\beta$-reduction which typically erases or duplicates values in a way that is destructive and irreversible. Reversible computation models have been studied in widely different areas ranging from cellular automata, program transformation concerned with the inversion of programs, reversible programming languages, the view-update problem in bidirectional computing and model transformation, static prediction of program properties, digital circuit design, to quantum computing.

## References

Slides in this Section are based on:

Holger Bock Axelsen and Robert Glück. What do reversible programs compute? In *Proceedings of the 14th international conference on Foundations of software science and computational structures: part of the joint European conferences on theory and practice of software*, FOSSACS'11/ETAPS'11, pages 42–56. Springer-Verlag, 2011

where more references can be found.

## Triple-format TM

The computation models that form the basis of programming languages are usually deterministic in one direction (forward), but non-deterministic in the opposite (backward) direction.

Reversible computing is the study of computation models wherein all computations are organized two-way deterministically, without any logical information loss.

**Definition** 4 **(3TM)** A (non-deterministic) turing machine $T$ is a tuple $(Q, \Sigma, \delta, b, q_s, q_f)$ where

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set of tape symbols and $b \in \Sigma$ is the blank symbol,
- $\delta \subseteq (Q \times [(\Sigma \times \Sigma) \cup \{L, N, R\}] \times Q) = \Delta$ is a partial relation defining the transition relation,
- $q_s \in Q$ is the starting state, and $q_f \in Q$ is the final state.

There must be no transitions leading out of $q_f$. There must be no transition leading into $q_s$. Symbols $L, N, R$ represent the three shift directions.

It is easy to see how to extend the definition to $k$-tape machines by letting $\delta \subseteq (Q \times [(\Sigma \times \Sigma)^k \cup \{L, N, R\}^k] \times Q)$.

**Definition** 5 **(Configuration)** . The configuration of a TM is a tuple

$$(q, (l, s, r)) \in Q \times (\Sigma^* \times \Sigma \times \Sigma^*) = \mathcal{C},$$

where $q \in Q$ is the internal state, $l, r \in \Sigma^*$ are the parts of the tape to the left and right of the tape head represented as strings, and $s \in \Sigma$ is the symbol being scanned by the tape head[a].

A TM $T = (Q, \Sigma, \delta, b, q_s, q_f)$ in configuration $C \in \mathcal{C}$, leads to configuration $C' \in \mathcal{C}$, written as $T \vdash C \rightsquigarrow C'$, defined for $s, s' \in \Sigma$, $l, r \in \Sigma^*$ and $q, q' \in Q$ by

$$\begin{array}{lll}
T \vdash (q, (l, s, r)) & \rightsquigarrow (q', (l, s', r)) & \text{if } (q, (s, s'), q') \in \delta, \\
T \vdash (q, (ls', s, r)) & \rightsquigarrow (q', (l, s', sr)) & \text{if } (q, L, q') \in \delta, \\
T \vdash (q, (l, s, r)) & \rightsquigarrow (q', (l, s, r)) & \text{if } (q, N, q') \in \delta, \\
T \vdash (q, (l, s, s'r)) & \rightsquigarrow (q', (ls, s', r)) & \text{if } (q, R, q') \in \delta.
\end{array}$$

[a] When describing tape contents we shall use the empty string $\epsilon$ to denote the infinite string of blanks $b^\omega$, and shall usually omit it when unambiguous.

## Reversible Computing

**Definition** 6 Let $T = (Q, \Sigma, \delta, b, q_s, q_f)$ be a TM.

- $T$ is locally forward deterministic iff for any distinct pair of transition rule triples $(q_1, a_1, q_1'), (q_2, a_2, q_2') \in \delta$, if $q_1 = q_2$ then $a_1 = (s_1, s_1')$ and $a_2 = (s_2, s_2')$, and $s1 \neq s2$.
- $T$ is locally backward deterministic iff for any distinct pair of triples $(q_1, a_1, q_1'), (q_2, a_2, q_2') \in \delta$, if $q_1' = q_2'$ then $a_1 = (s_1, s_1')$ and $a_2 = (s_2, s_2')$, and $s_1' \neq s_2'$.

**Definition** 7 A TM $T$ is reversible whenever it is locally forward and backward deterministic.

The reversible Turing machines (RTMs) are thus a proper subset of the set of all TM, with an easily decidable property.

**Lemma 1** If $T$ is a RTM then the induced computation step relation $T \vdash \cdot \rightsquigarrow \cdot$ is an injective function on configurations.

Proof. Trivial, since the backward determinism and conditions on $q_s, q_f$.

- Consider a TM with states $\{q_s, q_f\}$, with alphabet $\{0, 1, b\}$ and with transitions

$$\{(q_s, (1, 0), q_s); (q_s, (0, 0), q_f)\}.$$

It is a RTM?

It is easy to check that this TM is not injective on configurations, viz. $(q_s, (\epsilon, 0, \epsilon))$ and $(q_s, (\epsilon, 1, \epsilon))$ reduce both to $(q_F, (\epsilon, 0, \epsilon))$.

We shall consider the relationship mainly between deterministic and reversible Turing machines. Thus, all TMs are assumed to be fwd deterministic.

## I/O semantics

**Definition** 8 A tape containing one finite, blank-free string $s \in (\Sigma\backslash\{b\})^*$ is said to be given in standard configuration for a TM $(Q, \Sigma, \delta, b, q_s, q_f)$ iff the tape head is positioned to the immediate left of $s$ on the tape, i.e. if for some $q \in Q$, the configuration of the TM is $(q, (\epsilon, b, s))$.

The semantics $[\![T]\!]$ of a TM $T = (Q, \Sigma, \delta, b, q_s, q_f)$ is given by the relation

$$[\![T]\!] = \{(s, s') \in ((\Sigma\backslash\{b\})^* \times (\Sigma\backslash\{b\})^*) \mid T \vdash (q_s, (\epsilon, b, s)) \rightsquigarrow^* (q_f, (\epsilon, b, s'))\}.$$

To differentiate between extensional and intensional aspects, we shall write $T(x)$ to mean the computation of $[\![T]\!](x)$ by the specific machine $T$. We say that $T$ computes function $f$ iff $[\![T]\!] = f$.

Thus, the string transformation semantics of a TM $T$ has type

$$[\![T]\!] : \Sigma^* \rightharpoonup \Sigma^*.$$

**Theorem 1.** If $T$ is an RTM then $[\![T]\!]$ is injective.

Proof. The proof follows by Lemma 1. Remark that we are just considering a functional relation (a partial function).

## Inversion

It is well-known that if $f$ is a computable injective function then $f^{-1}$ is still computable.

**Lemma 2** Given a TM $T$ computing an injective function $[\![T]\!]$ there exists a TM $M(T)$, such that $[\![M(T)]\!] = [\![T]\!]^{-1}$.

**Lemma 3** Given a RTM $T = (Q, \Sigma, \delta, b, q_s, q_f)$, the RTM $T^{-1} = (Q, \Sigma, inv(\delta), b, q_f, q_s)$ computes the inverse function of $[\![T]\!]$, i.e. $[\![T^{-1}]\!] = [\![T]\!]^{-1}$, where $inv : \Delta \to \Delta$ is defined as

$$inv(q, (s, s'), q') = (q', (s', s), q) \quad inv(q, L, q') = (q', R, q)$$
$$inv(q, N, q') = (q', N, q) \quad inv(q, R, q') = (q', L, q)$$

Questions: Are RTM turing-complete? Are RTM and TM equivalent?



## Reversibilization

**Lemma 4 Landauer embedding** Given a 1-tape TM $T = (Q, \Sigma, \delta, b, q_s, q_f)$, there is a 2-tape RTM $L(T)$ such that $[\![L(T)]\!] : \Sigma^* \rightharpoonup \Sigma^* \times R^*$, and $[\![L(T)]\!] \stackrel{\text{def}}{=} \lambda x.([\![T]\!](x), trace(T, x))$, where $trace(T, x)$ is a complete trace of the specific rules from $\delta$ (enumerated as $R$) that are applied during the computation $T(x)$.

The trace is machine-specific. Given functionally equivalent TMs $T_1$ and $T_2$, i.e., $[\![T_1]\!] = [\![T_2]\!]$, it will almost always be the case that $[\![L(T_1)]\!] \neq [\![L(T_2)]\!]$.
The addition of the trace also changes the space consumption of the original program.

It is preferable that an injectivization generates extensional garbage data (specific to the function) rather than intensional garbage data (specific to the machine), since we would like to talk about semantics and ignore the mechanics and data-representation. This is attained in a Lemma, known colloquially as "Bennett's trick."

**Lemma 5 Bennet embedding** Given a 1-tape TM $T = (Q, \Sigma, \delta, b, q_s, q_f)$, there exists a 3-tape RTM $B(T)$, s.t. $[\![B(T)]\!] = \lambda x.(x, [\![T]\!](x))$.

While the construction (shown below) is defined for 1-tape machines, it can be extended to Turing machines with an arbitrary number of tapes.
It is important to note that neither Landauer embedding nor Bennett's method are semantics preserving as both reversibilizations lead to garbage: $[\![L(T)]\!] \neq [\![T]\!] \neq [\![B(T)]\!]$. References are:


- Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, 1961
- Charles H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, 1973

## Robustness

The Turing machines are remarkably computationally robust. Using multiple symbols, tapes, heads etc. has no impact on computability.

**Theorem** Let $T$ be a $k$-tape, $m$-symbol RTM. Then there exists a $1$-tape, $3$-symbol RTM $T'$ s.t. $[\![T]\!](x_1, \ldots, x_k) = (y_1, \ldots, y_k)$ iff $[\![T]\!](e(\langle x_1, ..., x_k\rangle)) = e(\langle y_1, ...y_k\rangle)$, where $\langle\cdot\rangle$ is the character-sequentialization of tape contents, and $e(\cdot)$ is a binary encoding.

Robusteness-proofs are not too different from the classic ones.

## Expressiveness

By Theorem 1 the RTMs compute only injective functions, but we can say something more.

**Theorem** Given a 1-tape $TM$ $S_1$ s.t. $[\![S_1]\!]$ is injective, and given a 1-tape $TM$ $S_2$ s.t. $[\![S_2]\!] = [\![S_1]\!]^{-1}$, there exists a 3-tape RTM $T$ s.t. $[\![T]\!] = [\![S_1]\!]$.

**Theorem 5.** The RTMs can compute exactly all injective computable (partial) functions. That is, given a 1-tape TM $T$ such that $[\![T]\!]$ is an injective function, there is a 3-tape RTM $T'$ such that $[\![T]\!] = [\![T']\!]$.

Proof. We construct and concatenate three RTMs (see Fig. 1 below, for a graphical representation.) First, construct $B(T)$ by applying Lemma 5 directly to $T$:

$$[\![B(T))]\!] = \lambda x.(x, [\![T]\!](x)), \qquad B(T) \in RTM$$

Second, construct the machine $B(M(T))^{-1}$ by successively applying the transformations of Lemmas 2, 5 and 3 to $T$ :

$$[\![B(M(T))^{-1}]\!] = (\lambda y.(y, [\![T]\!]^{-1}(y)))^{-1}, \qquad B(M(T))^{-1} \in RTM$$

Third, we can construct an RTM S, s.t. $[\![S]\!] = \lambda(a, b).(b, a)$, that is, a machine to exchange the contents of two tapes (in standard configuration). To see that $[\![B(M(T))^{-1} \circ S \circ B(T)]\!] = [\![T]\!]$, we apply the machine to an input, $x$:

$$\begin{aligned}
&[\![B(M(T))^{-1} \circ S \circ B(T)]\!](x) \\
&= [\![B(M(T))^{-1} \circ S]\!](x, [\![T]\!](x)) \\
&= [\![B(M(T))^{-1}]\!]([\![T]\!](x), x)) \\
&= (\lambda y.(y, [\![T]\!]^{-1}(y)))^{-1}([\![T]\!](x), x) \\
&= (\lambda y.(y, [\![T]\!]^{-1}(y)))^{-1}([\![T]\!](x), [\![T]\!]^{-1}([\![T]\!](x))) \\
&= [\![T]\!](x).
\end{aligned}$$

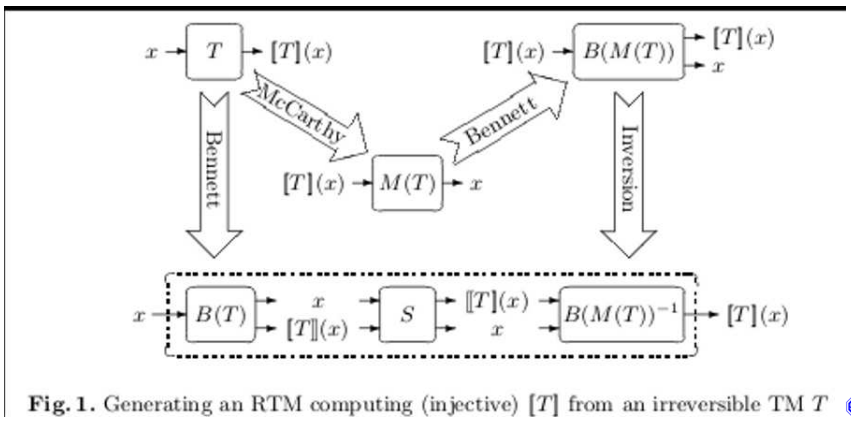



**Fig. 1.** Generating an RTM computing (injective) $[T]$ from an irreversible TM $T$

Thus, the RTMs can compute exactly all the injective computable functions. This suggests that the RTMs have the maximal computational expressiveness we could hope for (in any (effective) reversible computing model).

## Universality

A universal machine is a machine that can simulate the functional behaviour of any other machine.

**Definition** 9 **Classical universality.** A TM $U$ is classically universal iff for all TMs $T$, all inputs $x \in \Sigma^*$, and Gödel number $\ulcorner T \urcorner \in \Sigma^*$ representing $T$: $[\![U]\!](\ulcorner T \urcorner, x) = [\![T]\!](x)$.

The actual Gödel numbering $\ulcorner \_ \urcorner : TMs \to \Sigma^*$ for a given universal machine is not important, but we do require that it is computable and injective (up to renaming of symbols and states).
Because $[\![U]\!]$ in this definition is a non-injective function, it is clear that no classically universal RTM exists! Maybe, the appropriate question to ask is whether the RTMs are classically universal for just their own class, i.e. where the interpreted machine $T$ is restricted to being an RTM.

The answer is, again, NO:

**Remark** Different programs may compute the same function, so there exists RTMs $T_1 = T_2$ such that $[\![T_1]\!](x) = [\![T_2]\!](x)$, but $U \in RTM$ must be inherently non-injective, and therefore $[\![U]\!]$ cannot be computed by any RTM.

The classic definition of universality is therefore unsuitable if we want to capture a similar notion wrt RTMs.

**Definition** 10 (Universality). A TM $U_{TM}$ is universal iff for all TMs $T$ and all inputs $x \in \Sigma^*$,

$$[\![U_{TM}]\!](\ulcorner T \urcorner, x) = (\ulcorner T \urcorner, [\![T]\!](x)).$$

This is equivalent to the original definition of classical universality.
Given UTM universal $U_{TM}$, $snd \circ U_{TM}$ is classically universal, where $snd$ is a TM s.t. $[\![snd]\!] = \lambda(x, y).y$. The converse is easy.

**Definition** 11 An RTM $U_{RTM}$ is RTM-universal iff for all RTMs $T$ and all inputs $x \in \Sigma^*$,

$$[\![U_{RTM}]\!](\ulcorner T \urcorner, x) = (\ulcorner T \urcorner, [\![T]\!](x)).$$

**Theorem** There exists an RTM-universal RTM $U_R$.

Proof. We show that an RTM $U_R$ exists, such that for all RTMs $T$ , $[\![U_R]\!](\ulcorner T \urcorner, x) = (\ulcorner T \urcorner, [\![T]\!](x))$. Clearly, $[\![U_R]\!]$ is a computable function, since $T$ is a TM (so $[\![T]\!]$ is computable), and $\ulcorner T \urcorner$ is given as input. We show that $[\![U_R]\!]$ is injective.
Assuming $(\ulcorner T_1 \urcorner, x_1) \neq (\ulcorner T_2 \urcorner, x_2)$ we show that $(\ulcorner T_1 \urcorner, [\![T_1]\!](x_1) \neq (\ulcorner T_2 \urcorner, [\![T_2]\!](x_2)$.
Either $\ulcorner T_1 \urcorner \neq \ulcorner T_2 \urcorner$ or $x_1 \neq x_2$ or both. Because the program text is passed through to the output, the first and third cases are trivial. Assuming that $x_1 \neq x_2$ and $\ulcorner T_1 \urcorner = \ulcorner T_2 \urcorner$, we have that $[\![T_1]\!] = [\![T_2]\!]$, i.e. $T_1$ and $T_2$ are the same machine, and so compute the same function. Because they are RTMs this function is injective (by Theorem 1), so $x_1 \neq x_2$ implies that $[\![T_1]\!](x_1) \neq [\![T_2]\!](x_2)$.
Therefore, $[\![U_R]\!]$ is injective, and by Theorem 5 computable by some RTM $U_R$.

## Universality (2)

Given an irreversible TM computing the function of RTM-universality, Theorem 5 provides us with a possible construction for an RTM-universal RTM. In any case the construction uses the very inefficient generate-and-test inverter by McCarthy. We can do better.

**Lemma** There exists an RTM pinv, such that pinv is a program inverter for RTM programs,

$$[\![\texttt{pinv}]\!](\ulcorner T_1 \urcorner) = \ulcorner T_1^{-1} \urcorner.$$

This states that the RTMs are expressive enough to perform the program inversion of Lemma 3. For practical Gödelizations this will only take linear time.

**Theorem** Let $U$ be a classically universal TM s.t. $[\![U]\!](\ulcorner T \urcorner, x) = [\![T]\!](x)$ for all TM $T$. We can define the following RTM-universal machine $U_R$

$$U_R = \texttt{pinv}_1 \circ (B(U))^{-1} \circ S_{23} \circ \texttt{pinv}_1 \circ B(U),$$

where $\texttt{pinv}_1$ is an RTM that applies RTM program inversion on its first argument, $[\![\texttt{pinv}_1]\!](p, x, y) = ([\![\texttt{pinv}]\!](p), x, y)$ , and $S_{23}$ is an RTM that swaps its second and third arguments, $[\![S_{23}]\!] = \lambda(x, y, z).(x, z, y)$.

Proof. We must show that $[\![U_R]\!](\ulcorner T \urcorner, x) = (\ulcorner T \urcorner, [\![T]\!](x))$ for any RTM $T$. To show this, we apply $U_R$ to an input $(\ulcorner T \urcorner, x)$.

$$\begin{aligned}
[\![U_R]\!](\ulcorner T \urcorner, x) &= [\![\texttt{pinv}_1 \circ (B(U))^{-1} \circ S_{23} \circ \texttt{pinv}_1 \circ B(U)]\!](\ulcorner T \urcorner, x) \\
&= [\![\texttt{pinv}_1 \circ (B(U))^{-1} \circ S_{23} \circ \texttt{pinv}_1]\!](\ulcorner T \urcorner, x, [\![T]\!](x)) \\
&= [\![\texttt{pinv}_1 \circ (B(U))^{-1} \circ S_{23}]\!](\ulcorner T^{-1} \urcorner, x, [\![T]\!](x)) \\
&= [\![\texttt{pinv}_1 \circ (B(U))^{-1}]\!](\ulcorner T^{-1} \urcorner, [\![T]\!](x), x) \\
&= [\![\texttt{pinv}_1]\!](\ulcorner T^{-1} \urcorner, [\![T]\!](x)) \\
&= (\ulcorner T \urcorner, [\![T]\!](x))
\end{aligned}$$

The reasoning is presented in the figure.

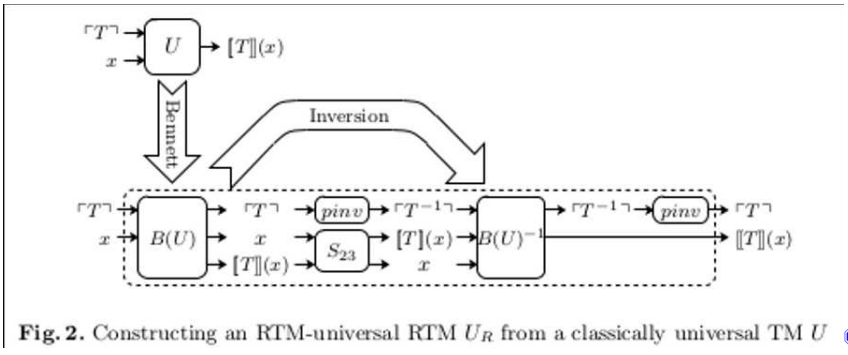



**Fig. 2.** Constructing an RTM-universal RTM $U_R$ from a classically universal TM $U$

Note that this implies that RTMs can simulate themselves exactly as time efficiently as the TMs can simulate themselves, but the space usage of the constructed machine will, by using Bennett's method, be excessive. However, there is nothing that forces us to start with an irreversible (universal) machine, when constructing an RTM-universal RTM, nor are reversibilizations necessarily required ...

### Janus

**Definition** 12 A (reversible) programming language $R$ is called r-Turing complete iff for all RTMs $T$ computing function $[\![T]\!]$, there exists a program $p \in R$, such that $[\![p]\!]_R = [\![T]\!]$.

Tipically this can be proved likewise the classic proofs, viz. by programming a "universal interpreter" and by remarking that the application of the "universal interpreter" to a representation of $T$ is still a program of the considered programming language.

This has been done for the language Janus ...



## Reversible Circuit 17 / 30

### References

Slides in this Section are based on:

Tommaso Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644. Springer-Verlag, 1980

where more references can be found. See also:

E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

## Reversible Computing

Concrete computation - whether by man or by machine - is a physical activity, and is ultimately governed by physical principles. One important role for mathematical theories of computation is to grasp in their axioms, in a stylized way, certain facts about the ultimate physical realizability of computing processes.
... albeit phisics theories which are (today) unsuitable for the realization of computating devices can be mathematically interesting by itself.

TM embodies in a heuristic form the axioms of computability theory.

In contrast to the opinion of David Deutsch, Edward Fredkin and Tommaso Toffoli sustain what follows.

> From Turing's original discussion (1936) it is clear that he intended to capture certain general physical constraints to which all concrete computing processes are subjected, as well as certain general physical mechanisms of which computing processes can undoubtedly avail themselves.

At the core of Turing's arguments, or, more generally, of Church's thesis, are the following physical assumptions.

- P1. The speed of propagation of information is bounded. (No "action at a distance": causal effects propagate through local interactions.)
- P2. The amount of information which can be encoded in the state of a finite system is bounded. (This is suggested by both thermodynamical and quantum-mechanical considerations.)
- 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 statement of technological fact.)

Additional axioms can be suggested from empirical observation (physical, chemical, biological, ...) to improve, the computational implementation, in some way.

Different axioms can be suggested from new empirical theories (not only of physics) to include new kind of computational devices between the our computing machinary.

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 microscopic laws of physics are presumed to be strictly reversible, abstract computing allow irreversible processes.



## Fan-out (Duplicazione)

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).

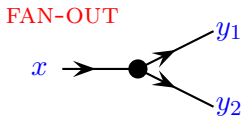
For this reason, in all that follows we shall restrict the meaning of the term "function composition" to one-to-one composition, where any substitution of output variables for input variables is one-to-one.

Thus, any fan-out node in a given function-composition scheme will have to be treated as an explicit occurrence of a fanout function of the form $\langle x \rangle \mapsto \langle x, ..., x \rangle$. Intuitively, the responsibility for providing fanout is shifted from the composition rules to the computing primitives.

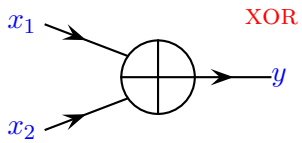
## Example 1

Not all boolean circuit are reversible.

| $x$ | $y_1$ | $y_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |




| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

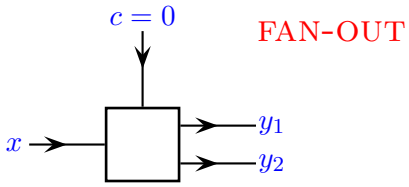



Neither of these functions is invertible.

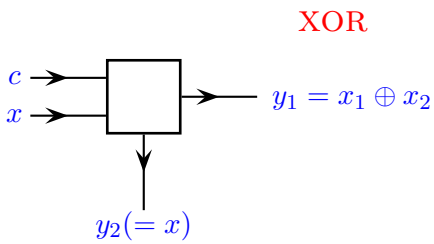
- FAN-OUT is not surjective, since, for instance, the output $\langle 0, 1\rangle$ cannot be obtained for any input value;
- XOR is not injective, since, for instance, the output $0$ can be obtained from two distinct input values, $\langle 0, 0\rangle$ and $\langle 1, 1\rangle$.

Yet, both functions admit an invertible realization.

| $c$ | $x$ | $y_1$ | $y_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |




| $c$ | $x$ | $y_1$ | $y_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

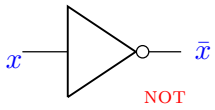



In what follows, we collectively call the source the auxiliary input components that have been used in a realization, such as component $c$ (for the gate FAN-OUT), and the sink the auxiliary output components such as $g$ (for the gate XOR).
The remaining input components will be collectively called the argument, and the remaining output components, the result.

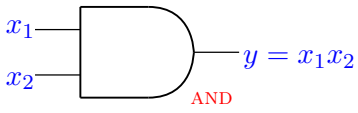
In general, both source and sink lines will have to be introduced in order to construct an invertible realization of a given function.

## Example 2

| $x$ | $y$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

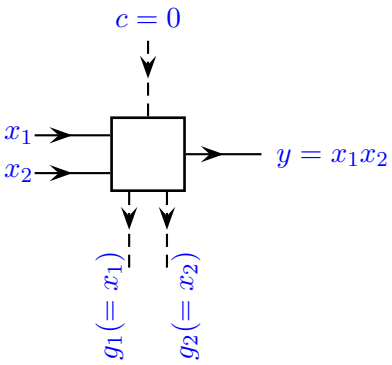



| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



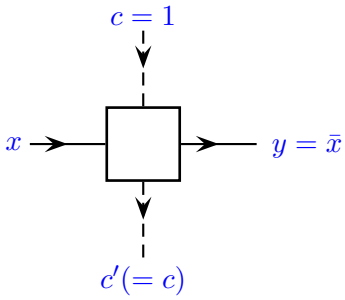

For example, from the invertible function AND/NAND defined by the table the AND function can be realized with one source line and two sink lines.

| $c$ | $x_1$ | $x_2$ | $y$ | $g_1$ | $g_2$ |
|---|---|---|---|---|---|
| 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 |




Reusing the XOR/FANOUT:

| $x$ | $c$ | $y$ | $c'$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



## Gate Lines

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 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, 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; note that here the sink, $c'$, returns in any case the value present at the source, $c$.

**Definition** 13 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 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).



## Reversibility Theorem

According to the following theorem, any finite function can be realized in this way starting from a suitable invertible one.

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

$$f\langle \underbrace{0, \ldots, 0}_{r}, x_1, \ldots, x_m\rangle = \phi_i\langle x_1, \ldots, x_m\rangle, \qquad (i = 1, \ldots, n).$$

Proof. It suffices to add the input to the output.

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).

## Invertible primitives

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 boolean functions.

**Lemma** There are primitive sets of universal reversible gates.

Proof. In the theory of reversible computing, a similar role is played by the AND/NAND element: if $c = 0$ then $y = x_1 x_2$ (their AND), if $c = 1$ then $y = \overline{x_1 x_2}$. 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 logic circuit.
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: if $c = 0$ then $x = y_1 = y_2$.

Recall that finite composition always yields invertible functions when applied to invertible functions.

Therefore, using the set of invertible primitives consisting of the XOR/FAN-OUT element and the AND/NAND element, any classical circuit 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 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 temporary storage, by taking advantage of the mapping $\langle 1, p, q \rangle \mapsto \langle 1, p, p \oplus q \rangle$.

In the element-by-element substitution procedure outlined above, the number of source and sink lines that are introduced is roughly proportional to the number of computing elements that make up the original network.

To achieve a less wasteful realization:

- one can augment the network 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.

## Conservative Logic

Universal logic capabilities can still be obtain even if one restricts attention logic circuit 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**.
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 basic primitive of conservative logic is the Fredkin gate, 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 | 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 |

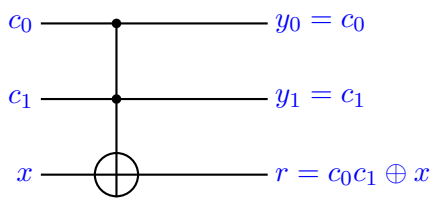
This computing element can be visualized as a device that performs conditional crossover of two data signals $x_1$ and $x_2$ according to the value of a control signal $c$.
In order to prove the universality of this gate as a logic primitive for reversible computing, it is sufficient to observe that AND can be obtained from the mapping $\langle p, q, 0\rangle \mapsto \langle p, pq, \bar{p}q\rangle$, and NOT and FAN-OUT from the mapping $\langle p, 1, 0\rangle \mapsto \langle p, p, \bar{p}\rangle$.

The conservative logic models some preservation of physics, in particular a physical realization of the Fredkin gate based on elastic collisions in the billiard ball model of computing.

## Toffoli Gate

Toffoli Gate is also called CCNOT (acronym of controlled-controlled-not).

| $c_0$ | $c_1$ | $x$ | $y_0$ | $y_1$ | $r$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |




It is easy to see that the Toffoli gate is universal, since AND and NOT can be repesented by:

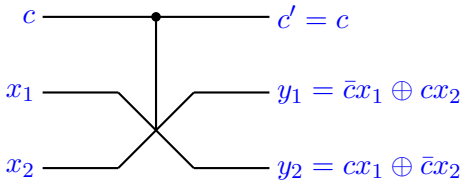
- if $c_0 = 1$ and $c_1 = 1$ then $r = \bar{x}$,
- if $x = 0$ then $r = c_0c_1$.



## Fredkin Gate

A (variant of) Fredkin Gate (also CSWAP gate, viz. controlled-swap) is also universal.

| $c$ | $x_1$ | $x_2$ | $c'$ | $y_1$ | $y_2$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |



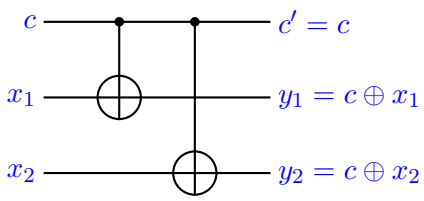

It is easy to see that the Fredkin gate is universal, since AND and NOT can be repesented by:

- if $x_2 = 0$ then $y_2 = cx_1$;
- if $x_1 = 0$ and $x_2 = 1$ then $y_2 = \bar{c}$

## Feynman Gate

Last, Feynman-gate is another useful reversible gate.

| $c$ | $x_1$ | $x_2$ | $c'$ | $y_1$ | $y_2$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |




It is easy to see that the Feynman gate is not universal, because AND cannot be represented.[a]

A NOT gate can be implemented as follows:
if $c = 0$ then $y_1 = \bar{x}_1$ (and $y_2 = \bar{x}_2$).

[a] An argument against XOR and XNOR as universal gates. An XOR gate is a parity generator. Cascading parity generators always produce parity generators. AND and OR are not parity functions. An XOR gate can be used as an inverter. An XNOR gate is an XOR followed by an inverter, so it is also a parity generator.



## . . . readings

Interesting books are

Kalyan S. Perumalla. *Introduction to Reversible Computing*. Chapman Hall, CRC Computational Science, 2013

Alexis De Vos. *Reversible Computing: Fundamentals, Quantum Computing, and Applications*. Wiley, 2010