Unconventional Computation

Bruce MacLennan
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville

Version of

October 24, 2019

Contents

Chapter I

Introduction

A What is unconventional computation?

Unconventional computation and its synonym, *nonstandard computation*, are negative terms; they refer to computation that is not "conventional" or "standard." Conventional computation is the kind we are all familiar with, which has dominated computer science since the 1950s. We can enumerate its common characteristics:

- Information representation and processing is digital and in fact binary.
- Computers are organized according to the von Neumann architecture, which separates memory from a processor that fetches data from memory for processing and then returns it to memory.
- Memory is organized as one or more arrays of units (bytes or words) indexed by natural numbers.
- The processor executes instructions from memory, sequentially, in discrete steps, and one at a time; therefore instructions are encoded in binary.
- The sequential binary logic of the processor is implemented by means of electronically-controlled electronic switches.

- Programs are written as hierarchically organized sequences of instructions, which perform computation, input, output, and control of program execution.
- Programs can decide between alternative sequences of instructions.
- Programs can repeat sequences of instructions until some condition is satisfied.
- Programs can be hierarchically organized into subprograms, subprograms of subprograms, etc.

Unconventional computers, then, are different in at least one of these characteristics. For example, analog computers represent information by continuous physical quantities rather than bits, and some analog computers operate in continuous time (obeying differential equations) rather than in discrete sequential steps. Nonelectronic computers may use light, fluids, microorganisms, or DNA and other molecules as a computational medium. In molecular computers, vast numbers of operations take place asynchronously in parallel. Ballistic computers implement reversible computation, so they may dissipate arbitrarily small amounts of energy, but cannot use common programming operations, such as storing into a memory. Functional programming does not use rewritable variables, and computers designed for functional programming, such as reduction machines, do not have a programmer-accessible storable memory. Quantum computers do not do one thing at a time, but execute an exponentially large number of operations simultaneously in quantum superposition. Alternative computational paths can be executed in superposition and otherwise iterative processes can be executed all at once. Neuromorphic computers, which are inspired by the brain, don't have an addressible memory in the usual sense (they learn by analog changes in vast numbers of connection strengths, so that individual memories are distributed over large overlapping sets of connections). They are also massively parallel and use analog computation. Programs for unconventional computers can take unconventional forms, such as recursive function definitions, ordinary or partial differential equations, quantum operations, neural network architectures, sequences of chemical reactions, genetic regulatory networks, or even continuous images. These are just a few of the ways in which computation can be "unconventional" or "nonstandard." You may be asking, however, why we would want to compute in these unconventional ways, given the outstanding success of conventional computation. That is our next topic.

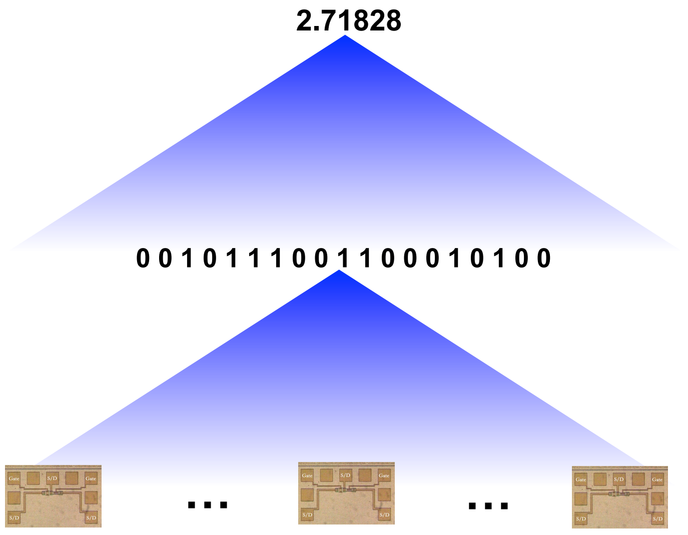


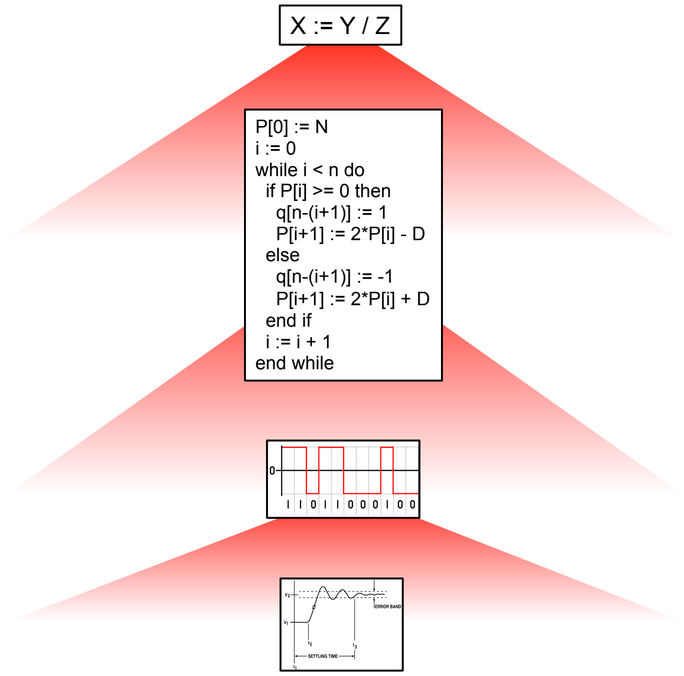
Figure I.1: Hierarchy of spatial scales in conventional computing.

B Post-Moore's Law computing

Although estimates differ, it is clear that the end of Moore's Law is in sight; there are physical limits to the density of binary logic devices and to their speed of operation.[1] This will require us to approach computation in new ways, which will present significant challenges, but can also broaden and deepen our concept of computation in natural and artificial systems.

In the past there has been a significant difference in scales between computational processes and the physical processes by which they are realized. For example, there are differences in spatial scale: the data with which programs operate (integers, floating point numbers, characters, pointers, etc.) are represented by large numbers of physical devices comprising even larger numbers of physical particles (Fig. I.1). Also, there are differences in time scale: elementary computational operations (arithmetic, instruction sequencing, memory access, etc.), are the result of large numbers of state changes at the device level (typically involving a device moving from one saturated state to another) (Fig. I.2). However, increasing the density and speed of

[1]This section is adapted from MacLennan (2008). Moore's Law was presented in Moore (1965).



```
P[0] := N
i := 0
while i < n do
  if P[i] >= 0 then
    q[n-(i+1)] := 1
    P[i+1] := 2*P[i] - D
  else
    q[n-(i+1)] := -1
    P[i+1] := 2*P[i] + D
  end if
  i := i + 1
end while
```

Figure I.2: Hierarchy of temporal scales in conventional computing.

computation will force it to take place on the scale (spatial and temporal) near that of the underlying physical processes. With fewer hierarchical levels between computations and their physical realizations, and less time for implementing computational processes, computation will have to become more like the underlying physical processes. That is, *post-Moore's Law computing* will depend on a greater assimilation of computation to physics.

In discussing the role of physical embodiment in the "grand challenge" of unconventional computing, Stepney (2004, p 29) writes,

> Computation is physical; it is necessarily embodied in a device whose behaviour is guided by the laws of physics and cannot be completely captured by a closed mathematical model. This fact of embodiment is becoming ever more apparent as we push the bounds of those physical laws.

Traditionally, a sort of Cartesian dualism has reigned in computer science; programs and algorithms have been conceived as idealized mathematical objects; software has been developed, explained, and analyzed independently of hardware; the focus has been on the *formal* rather than the *material.* Post-Moore's Law computing, in contrast, because of its greater assimilation to physics, will be less idealized, less independent of its physical realization. On one hand, this will increase the difficulty of programming since it will be dependent on (or, some might say, contaminated by) physical concerns. On the other hand, as will be explored in this book, it also presents many opportunities that will contribute to our understanding and application of information processing in the future.

C Super-Turing vs. non-Turing

In addition to the *practical* issues of producing more powerful computers, unconventional computation poses the question of its *theoretical* power.[2] Might some forms of unconventional computation provide a means of *super-Turing computation* (also called *hypercomputation*), that is, of computation beyond what is possible on a Turing machine? This is an important theoretical question, but obsession with it can divert attention from more significant issues in unconventional computing. Therefore it is worth addressing it before we get into specific unconventional computing paradigms.

[2] This section is adapted from MacLennan (2009b).

C.1 The limits of Turing computation

C.1.a Frames of Relevance

It is important to remember that Church-Turing (CT) computation is a *model* of computation, and that computation is a physical process taking place in certain physical objects (such as computers). Models are intended to help us understand some class of phenomena, and they accomplish this by making *simplifying assumptions* (typically *idealizing assumptions*, which omit physical details taken to be of secondary importance). For example, we might use a linear mathematical model of a physical process even though we know that its dynamics is only approximately linear; or a fluid might be modeled as infinitely divisible, although we know it is composed of discrete molecules. We are familiar also with the fact that several models may be used for a single system, each model suited to understanding certain aspects of the system but not others. For example, a circuit diagram shows the electrical interconnections among components (qua electrical devices), and a layout diagram shows the components' sizes, shapes, spatial relationships, etc.

As a consequence of its simplifying assumptions, each model comes with a (generally unstated) *frame of relevance*, which delimits (often vaguely) the questions that the model can answer accurately. For example, it would be a mistake to draw conclusions from a circuit diagram about the size, shape, or physical placement of circuit components. Conversely, little can be inferred about the electrical properties of a circuit from a layout diagram.

Within a (useful) model's frame of relevance, its simplifying assumptions are sensible (e.g., they are good approximations); outside of it they may not be. That is, within its frame of relevance a model will give us good answers (not necessarily 100% correct) and help us to understand the characteristics of the system that are most relevant *in that frame*. Outside of its intended frame, a model might give good answers (showing that its *actual* frame can be larger than its *intended* frame), but we cannot assume that to be so. Outside of its frame, the answers provided by a model may reflect the simplifying assumptions of the model more than the system being modeled. For example, in the frame of relevance of macroscopic volumes, fluids are commonly modeled as infinitely divisible continua (an idealizing assumption), but if we apply such a model to microscopic (i.e., molecular scale) volumes, we will get misleading answers, which are a consequence of the simplifying assumptions.

C.1.b The frame of relevance of Church-Turing computation

It is important to explicate the frame of relevance of Church-Turing (CT) computation, by which I mean not just Turing machines, but also equivalent models of computation, such as the lambda calculus and Post productions, as well as other more or less powerful models based on similar assumptions (discussed below). (Note however that the familiar notions of equivalence and power are themselves dependent on the frame of relevance of these models, as will be discussed.) The CT frame of relevance becomes apparent if we recall the original questions the model was intended to answer, namely questions of effective calculability and formal derivability. As is well known, the CT model arises from an idealized description of what a mathematician could do with pencil and paper. Although a full analysis of the CT frame of relevance is beyond the scope of this chapter (but see MacLennan, 1994b, 2003, 2004), I will mention a few of the idealizing assumptions.

Within the CT frame of relevance, something is computable if it can be computed with finite but unbounded resources (e.g., time, memory). This is a reasonable idealizing assumption for answering questions about formal derivability, since we don't want our notion of a proof to be limited in length or "width" (size of the formal propositions). It is also a reasonable simplifying assumption for investigating the limits of effective calculability, which is a idealized model of arithmetic with paper and pencil. Again, in the context of the formalist research program in mathematics, there was no reason to place an a priori limit on the number of steps or the amount of paper (or pencil lead!) required. Note that these *are* idealizing assumptions: so far as we know, physical resources are not unbounded, but these bounds were not considered relevant to the questions that the CT model was originally intended to address; in this frame of relevance "finite but unbounded" is a good idealization of "too large to be worth worrying about."

Both formal derivation and effective calculation make use of finite formulas composed of discrete tokens, of a finite number of types, arranged in definite structures (e.g., strings) built up according to a finite number of primitive structural relationships (e.g., left-right adjacency). It is further assumed that the types of the tokens are positively determinable, as are the primitive interrelationships among them. Thus, in particular, we assume that there is no uncertainty in determining whether a token is present, whether a configuration is one token or more than one, what is a token's type, or how the tokens are arranged, and we assume that they can be rearranged with perfect accuracy according to the rules of derivation. These are reasonable assumptions in the study of formal mathematics and effective calculability, but it is important to realize that they are *idealizing assumptions*, for even mathematicians can make mistakes in reading and copying formulas and in applying formal rules!

Many of these assumptions are captured by the idea of a *calculus*, but a phenomenological analysis of this concept is necessary to reveal its background of assumptions (MacLennan, 1994b). Briefly, we may state that both information representation and information processing are assumed to be *formal*, *finite*, and *definite* (MacLennan, 2003, 2004). These and other assumptions are taken for granted by the CT model because they are reasonable in the context of formal mathematics and effective calculability. Although the originators of the model discussed some of these simplifying assumptions (e.g., Markov, 1961), many people today do not think of them as assumptions at all, or consider that they might not be appropriate in some other frames of relevance.

It is important to mention the concept of time presupposed in the CT model, for it is not discrete time in the familiar sense in which each unit of time has the same duration; it is more accurate to call it *sequential time* (van Gelder, 1997). This is because the CT model does not take into consideration the time required by an individual step in a derivation or calculation, so long as it is finite. Therefore, while we can *count* the number of steps, we cannot translate that count into real time, since the individual steps have no definite duration. As a consequence, the only reasonable way to compare the time required by computational processes is in terms of their asymptotic behavior. Again, sequential time is reasonable in a model of formal derivability or effective calculability, since the time required for individual operations was not relevant to the research programme of formalist mathematics (that is, the timing was irrelevant in that frame of relevance), but it can be very relevant in other contexts, as will be discussed.

Finally I will mention a simplifying assumption of the CT model that is especially relevant to hypercomputation, namely, the assumption that computation is equivalent to evaluating a well-defined function on an argument. Certainly, the mathematical function, in the full generality of its definition, is a powerful and versatile mathematical concept. Almost any mathematical object can be treated as a function, and functions are essential to the description of processes and change in the physical sciences. Therefore, it was natural, in the context of the formalist program, to focus on functions in the investigation of effective calculation and derivation. Furthermore, many early applications of computers amounted to function evaluations: you put in a deck of punched cards or mounted a paper or magnetic tape, started the program, it computed for awhile, and when it stopped you had an output in the form of cards, tape, or paper. Input — compute — output, that was all there was to it. If you ran the program again with a different input, that amounted to an independent function evaluation. The only relevant aspect of a program's behavior was the input-output correspondence (i.e., the mathematical function).

This view can be contrasted with newer ones in which, for example, a computation involves continuous, non-terminating interaction with its environment, such as might be found in control systems and autonomous robotics. Some new models of computation have moved away from the idea of computation as the evaluation of a fixed function (Eberbach et al., 2003; Milner, 1993; Milner et al., 1992; Wegner, 1997, 1998; Wegner & Goldin, 2003). In the CT frame of relevance, however, the natural way to compare the "power" of models of computation was in terms of the classes of functions they could compute, a single dimension of power now generalized into a partial order of set inclusions (but still based on a single conception of power: computing a class of functions).[3]

C.2 New computational models

A reasonable position, which many people take explicitly or implicitly, is that the CT model is a perfectly adequate model of everything we mean by "computation," and therefore that any answers that it affords us are definitive. However, as we have seen, the CT model exists in a frame of relevance, which delimits the kinds of questions that it can answer accurately, and, as I will show, there are important computational questions that fall outside this frame of relevance.

C.2.a Natural computation

Natural computation may be defined as computation occurring in nature or inspired by computation in nature. The information processing and control

[3]I note in passing that this approach raises all sorts of knotty cardinality questions, which are inevitable when we deal with such "large" classes; therefore in some cases results depend on a particular axiomatization or philosophy of mathematics.

that occurs in the brain is perhaps the most familiar example of computation in nature, but there are many others, such as the distributed and self-organized computation by which social insects solve complicated optimization problems and construct sophisticated, highly structured nests. Also, the DNA of multicellular organisms defines a developmental program that creates the detailed and complex structure of the adult organism. For examples of computation inspired by that in nature, we may cite artificial neural networks, genetic algorithms, artificial immune systems, and ant swarm optimization, to name just a few. Next I will consider a few of the issues that are important in natural computation, but outside the frame of relevance of the CT model.

One of the most obvious issues is that, because computation in nature serves an adaptive purpose, it must satisfy stringent *real-time constraints.* For example, an animal's nervous system must respond to a stimulus — fight or flight, for example — in a fraction of a second. Also, in order to control coordinated sensorimotor behavior, the nervous system has to be able to process sensory and proprioceptive inputs quickly enough to generate effector control signals at a rate appropriate to the behavior. And an ant colony must be able to allocate workers appropriately to various tasks in real time in order to maintain the health of the colony.

In nature, asymptotic complexity is generally irrelevant; the constants matter and input size is generally fixed or varies over a relatively limited range (e.g., numbers of sensory receptors, colony size). Whether the algorithm is linear, quadratic, or exponential is not so important as whether it can deliver useful results in required real-time bounds in the cases that actually occur. The same applies to other computational resources. For example, it is not so important whether the number of neurons required varies linearly or quadratically with the number of inputs to the network; what matters is the absolute number of neurons required for the actual number of inputs, and how well the system will perform with the number of inputs and neurons it actually has.

Therefore, in natural computation, what does matter is how the real-time response rate of the system is related to the real-time rates of its components (e.g., neurons, ants) and to the actual number of components. This means that it is not adequate to treat basic computational processes as having an indeterminate duration or speed, as is commonly done in the CT model. In the natural-computation frame of relevance, knowing that a computation will *eventually* produce a correct result using *finite but unbounded resources* is largely irrelevant. The question is whether it will produce a *good-enough* result using available resources subject to real-time constraints.

Many of the inputs and outputs to natural computation are *continuous in magnitude* and *vary continuously* in real time (e.g., intensities, concentrations, forces, spatial relations). Many of the computational processes are also continuous, operating in continuous real time on continuous quantities (e.g., neural firing frequencies and phases, dendritic electrical signals, protein synthesis rates, metabolic rates). Obviously these real variables can be approximated arbitrarily closely by discrete quantities, but that is largely irrelevant in the natural-computation frame of relevance. The most natural way to model these systems is in terms of continuous quantities and processes.

If the answers to questions in natural computation seem to depend on "metaphysical issues," such as whether only Turing-computable reals exist, or whether all the reals of standard analysis exist, or whether non-standard reals exist, then I think that is a sign that we are out of the model's frame of relevance, and that the answers are more indicative of the model itself than of the modeled natural-computation system. For models of natural computation, naive real analysis, like that commonly used in science and engineering, should be more than adequate; it seems unlikely that disputes in the foundations of mathematics will be relevant to our understanding how brains coordinate animal behavior, how ants and wasps organize their nests, how embryos self-organize, and so forth.

C.2.b Cross-frame comparisons

This example illustrates the more general pitfalls that arise from *cross-frame comparisons.* If two models have different frames of relevance, then they will make different simplifying and idealizing assumptions; for example objects whose existence is assumed in one frame (such as standard real numbers) may not exist in the other (where all objects are computable). Therefore, a comparison requires that one of the models be translated from its own frame to the other (or that both be translated to a third), and, in doing this translation, assumptions compatible with the new frame will have to be made. For example, if we want to investigate the computational power of neural nets in the CT frame (i.e., in terms of classes of functions of the integers), then we will have to decide how to translate the naive continuous variables of the neural net model into objects that exist in the CT frame. For instance, we might choose fixed-point numbers, computable reals (represented in some way by finite programs), or arbitrary reals (represented by infinite discrete structures). We then discover (as reported in the literature: e.g., Maass & Sontag, 1999a; Siegelmann & Sontag, 1994a), that our conclusions depend on the choice of numerical representation (which is largely irrelevant in the natural-computation frame). That is, our conclusions are more a function of the specifics of the cross-frame translation than of the modeled systems.

Such results tell us nothing about, for example, why brains do some things so much better than do contemporary computers, which are made of much faster components. That is, in the frame of natural computation, the issue of the representation of continuous quantities does not arise, for it is irrelevant to the questions addressed by this frame, but this issue is crucial in the CT frame. Conversely, from within the frame of the CT model, much of what is interesting about neural net models (parallelism, robustness, real-time response) becomes irrelevant. Similar issues arise when the CT model is taken as a benchmark against which to compare unconventional models of computation, such as quantum and molecular computation.

C.2.c Relevant issues for natural computation

We have seen that important issues in the CT frame of relevance, such as asymptotic complexity and the computability of classes of functions, are not so important in natural computation. What, then, are the relevant issues?

One important issue in natural computation is *robustness*, by which I mean effective operation in the presence of noise, uncertainty, imprecision, error, and damage, all of which may affect the computational process as well as its inputs. In the CT model, we assume that a computation should produce an output exactly corresponding to the evaluation of a well-defined function on a precisely specified input; we can, of course, deal with error and uncertainty, but it's generally added as an afterthought. Natural computation is better served by models that incorporate this indefiniteness a priori.

In the CT model, the basic standard of correctness is that a program correctly compute the same outputs as a well-defined function evaluated on inputs in that function's domain. In natural computation, however, we are often concerned with *generality and flexibility*, for example: How well does a natural computation system (such as a neural network) respond to inputs that are *not* in its intended domain (the domain over which it was trained or for which it was designed)? How well does a neural control system respond to unanticipated inputs or damage to its sensors or effectors? A related issue is *adaptability*: How well does a natural computation system change its behavior (which therefore does not correspond to a fixed function)?

Finally, many natural computation systems are not usefully viewed as computing a function at all. As previously remarked, with a little cleverness anything can be viewed as a function, but this is not the simplest way to treat many natural systems, which often are in open and continuous interaction with their environments and are effectively nonterminating. In natural computation we need to take a more biological view of a computational system's "correctness" (better: *effectiveness*). It will be apparent that the CT model is not particularly well suited to addressing many of these issues, and in a number of cases begs the questions or makes assumptions incompatible with addressing them. Nevertheless, real-time response, generality, flexibility, adaptability, and robustness in the presence of noise, error, and uncertainty are important issues in the frame of relevance of natural computation.

C.2.d Nanocomputation

Nanocomputation is another domain of computation that seems to be outside the frame of relevance of the CT model. By nanocomputation I mean any computational process involving sub-micron devices and arrangements of information; it includes quantum computation (Ch. III) and *molecular computation* (e.g., DNA computation), in which computation proceeds through molecular interactions and conformational changes (Ch. IV).

Due to thermal noise, quantum effects, etc., *error* and *instability* are unavoidable characteristics of nanostructures. Therefore they must be taken as givens in nanocomputational devices and in their interrelationships (e.g., interconnections), and also in the structures constructed by nanocomputational processes (e.g., in algorithmic self-assembly: Winfree, 1998). Therefore, a "perfect" structure is an over-idealized assumption in the context of nanocomputation; defects are unavoidable. In many cases structures are not fixed, but are stationary states occurring in a system in constant flux. Similarly, unlike in the CT model, nanocomputational operations cannot be assumed to proceed correctly, for the probability of error is always non-negligible. Error cannot be considered a second-order detail added to an assumed perfect computational system, but should be built into a model of nanocomputation from the beginning. Indeed, operation cannot even be assumed to proceed uniformly forward. For example, chemical reactions always have a non-zero probability of moving backwards, and therefore molecular computation systems must be designed so that they accomplish their purposes in spite of such reversals. This is a fundamental characteristic of molecular computation and should be an essential part of any model of it.

C.2.e Summary of issues

In summary, the notion of super-Turing computation, *stricto sensu*, exists only in the frame of relevance of the Church-Turing model of computation, for the notion of being able to compute "more" than a Turing machine presupposes a particular notion of "power." Although it is interesting and important to investigate where unconventional models of computation fall in this computational hierarchy, it is also important to explore *non*-Turing computation, that is, models of computation with different frames of relevance from the CT model. Several issues arise in the investigation of non-Turing computation: (1) What is computation in the broad sense? (2) What frames of relevance are appropriate to unconventional conceptions of computation (such as natural computation and nanocomputation), and what sorts of models do we need for them? (3) How can we fundamentally incorporate error, uncertainty, imperfection, and reversibility into computational models? (4) How can we systematically exploit new physical processes (quantum, molecular, biological, optical) for computation? The remainder of this chapter addresses issues (1) and (4).

C.3 Computation in general

C.3.a Kinds of computation

Historically, there have been many kinds of computation, and the existence of alternative frames of relevance shows us the importance of non-Turing models of computation. How, then, can we define "computation" in sufficiently broad terms? Prior to the twentieth century computation involved operations on mathematical objects by means of physical manipulation. The familiar examples are arithmetic operations on numbers, but we are also familiar with the geometric operations on spatial objects of Euclidean geometry, and with logical operations on formal propositions. Modern computers operate on a much wider variety of objects, including character strings, images, sounds, and much else. Therefore, the observation that computation uses physical processes to accomplish mathematical operations on mathematical objects must be understood in the broadest sense, that is, as abstract operations on abstract objects. In terms of the traditional distinction between *form* and *matter*, we may say that computation uses material states and processes to *realize* (implement) formal operations on abstract forms. But what sorts of physical processes?

C.3.b Effectiveness and mechanism

The concepts of *effectiveness* and *mechanism*, familiar from CT computation, are also relevant to computation in a broader sense, but they must be similarly broadened. To do this, we may consider the two primary uses to which models of computation are put: understanding computation in nature and designing computing devices. In both cases the model relates information representation and processing to underlying physical processes that are considered unproblematic within the frame of relevance of the model.

For example, the CT model sought to understand effective calculability and formal derivability in terms of simple processes of symbol recognition and manipulation, such as are routinely performed by mathematicians. Although these are complex processes from a cognitive science standpoint, they were considered unproblematic in the context of metamathematics. Similarly, in the context of natural computation, we may expect a model of computation to explain intelligent information processing in the brain in terms of electrochemical processes in neurons (considered unproblematic in the context of neural network models). Or we may expect a different model to explain the efficient organization of an ant colony in term of pheromone emission and detection, simple stimulus-response rules, etc. In all these cases the explanation is *mechanistic*, in the sense that it refers to *primary qualities*, which can be objectively measured or positively determined, as opposed to *secondary qualities*, which are subjective or depend on human judgment, feeling, etc. (all, of course, in the context to the intended purpose of the model); measurements and determinations of primary qualities are *effective* in that their outcomes are reliable and dependable.

A mechanistic physical realization is also essential if a model of computation is to be applied to the design of computing devices. We want to use physical processes that are *effective* in the broad sense that they result reliably in the intended computations. In this regard, electronic binary logic has proved to be an extraordinarily effective mechanism for computation. (In

Sec. C.4.b I will discuss some general effectiveness criteria.)

C.3.c Multiple realizability

Although the forms operated upon by a computation must be materially realized in some way, a characteristic of computation that distinguishes it from other physical processes is that it is independent of *specific* material realization. That is, although a computation must be materially realized in *some* way, it can be realized in *any* physical system having the required formal structure. (Of course, there will be practical differences between different physical realizations, but I will defer consideration of them until later.) Therefore, when we consider computation *qua computation*, we must, on the one hand, restrict our attention to formal structures that are mechanistically realizable, but, on the other, consider the processes independently of any particular mechanistic realization.

These observations provide a basis for determining whether or not a particular physical system (in the brain, for example) is computational (MacLennan, 1994c, 2004). If the system could, in principle at least, be replaced by another physical system having the same formal properties and still accomplish its purpose, then it is reasonable to consider the system computational (because its formal structure is sufficient to fulfill its purpose). On the other hand, if a system can fulfill its purpose only by control of particular substances or particular forms of energy (i.e., it is not independent of a specific material realization), then it cannot be purely computational. (Nevertheless, a computational system will not be able to accomplish its purpose unless it can interface properly with its physical environment; this is a topic I will consider in Sec. C.3.f.)

C.3.d Defining computation

Based on the foregoing considerations, we have the following definition of computation (MacLennan, 1994c, 2004, 2009b):

Definition 1 Computation *is a mechanistic process, the purpose of which is to perform abstract operations on abstract objects.*

Alternately, we may say that computation accomplishes the formal transformation of formal objects by means of mechanistic processes operating on the objects' material embodiment. The next definition specifies the relation between the physical and abstract processes:

Definition 2 *A mechanistic physical system* realizes *a computation if, at the level of abstraction appropriate to its purpose, the abstract transformation of the abstract objects is a sufficiently accurate model of the physical process. Such a physical system is called a* realization *of the computation.*

That is, the physical system realizes the computation if we can see the material process as a sufficiently accurate embodiment of the formal structure, where the sufficiency of the accuracy must be evaluated in the context of the system's purpose. Mathematically, we may say that there is a homomorphism from the physical system to the abstract system, because the abstract system has some, but not all, of the formal properties of the physical system (MacLennan, 1994a, 2004). The next definition classifies various systems, both natural and artificial, as computational:

Definition 3 *A physical system is* computational *if its function (purpose) is to realize a computation.*

Definition 4 *A* computer *is an artificial computational system.*

Thus the term "computer" is restricted to intentionally manufactured computational devices; to call the brain a computer is a metaphor. These definitions raise a number of issues, which I will discuss briefly; no doubt the definitions can be improved.

C.3.e Purpose

First, these definitions make reference to the *function* or *purpose* of a system, but philosophers and scientists are justifiably wary of appeals to purpose, especially in a biological context. However, the use of purpose in the definition of computation is unproblematic, for in most cases of practical interest, purpose is easy to establish. (There are, of course, borderline cases, but that fact does not invalidate the definition.) On the one hand, in a technological context, we can look to the stated purpose for which an artificial system was designed. On the other, in a biological context, scientists routinely investigate the purposes (functions) of biological systems, such as the digestive system and immune system, and make empirically testable hypotheses about their purposes. Ultimately such claims of biological purpose may be reduced to a system's selective advantage to a particular species in that species' environment of evolutionary adaptedness, but in most cases we can appeal to more immediate ideas of purpose.

On this basis we may identify many natural computational systems. For example, the function of the brain is primarily computational (in the sense used here), which is easiest to see in sensory areas. For example, there is considerable evidence that an important function of primary visual cortex is to perform a Gabor wavelet transform on visual data (Daugman, 1993); this is an abstract operation that could, in principal, be realized by a non-neural physical system (such as a computer chip). Also, pheromone-mediated interactions among insects in colonies often realize computational ends such as allocation of workers to tasks and optimization of trails to food sources. Likewise DNA transcription, translation, replication, repair, etc., are primarily computational processes.

However, there is a complication that arises in biology and can be expected to arise in our biologically-inspired robots and more generally in post-Moore's Law computing. That is, while the distinction between computational and non-computational systems is significant to *us*, it does not seem to be especially significant to *biology*. The reason may be that we are concerned with the *multiple realizability* of computations, that is, with the fact that they have alternative realizations, for this property allows us to consider the implementation of a computation in a different technology, for example in electronics rather than in neurons. In nature, typically, the realization is given, since natural life is built upon a limited range of substances and processes. On the other hand, there is often selective pressure in favor of exploiting a biological system for as many purposes as possible. Therefore, in a biological context, we expect physical systems to serve multiple functions, and therefore many such systems will not be *purely* computational; they will fulfill other functions besides computation. From this perspective, it is remarkable how free nervous systems are of non-computational functions.

C.3.f Transduction

The purpose of computation is the abstract transformation of abstract objects, but obviously these formal operations will be pointless unless the computational system interfaces with its environment in some way. Certainly our computers need input and output interfaces in order to be useful. So also computational systems in the brain must interface with sensory receptors, muscles, and many other noncomputational systems to fulfill their functions. In addition to these practical issues, the computational interface to the physical world is relevant to the *symbol grounding problem*, the philosophical question of how abstract symbols can have real-world content (Harnad, 1990, 1993; MacLennan, 1993). Therefore we need to consider the interface between a computational system and its environment, which comprises *input* and *output transducers*.

The relation of transduction to computation is easiest to see in the case of analog computers. The inputs and outputs of the computational system have some physical dimensions (light intensity, air pressure, mechanical force, etc.), because they must have a specific physical realization for the system to accomplish its purpose. On the other hand, the computation itself is essentially dimensionless, since it manipulates pure numbers. Of course, these internal numbers must be represented by *some* physical quantities, but they can be represented in any appropriate physical medium. In other words, computation is *generically* realized, that is, realized by any physical system with an appropriate formal structure, whereas the inputs and outputs are *specifically* realized, that is, constrained by the environment with which they interface to accomplish the computational system's purpose.

Therefore we can think of *(pure) transduction* as changing matter (or energy) while leaving form unchanged, and of *computation* as transforming form independently of matter (or energy). In fact, most transduction is not pure, for it modifies the form as well as the material substrate, for example, by filtering. Likewise, transductions between digital and analog representations transform the signal between discrete and continuous spaces.

C.3.g Classification of computational dynamics

The preceding definition of computation has been framed quite broadly, to make it *topology-neutral*, so that it encompasses all the forms of computation found in natural and artificial systems. It includes, of course, the familiar computational processes operating in discrete steps and on discrete state spaces, such as in ordinary digital computers. It also includes continuous-time processes operating on continuous state spaces, such as found in conventional analog computers and field computers (Adamatzky, 2001; Adamatzky et al., 2005; MacLennan, 1987, 1999). However, it also includes hybrid processes, incorporating both discrete and continuous computation, so long as they are mathematically consistent (MacLennan, 2010). As we expand our computational technologies outside of the binary electronic realm, we will have to consider these other topologies of computation. This is not so much a problem as an opportunity, for many important applications, especially in natural computation, are better matched to these alternative topologies.

In connection with the classification of computational processes in terms of their topologies, it is necessary to say a few words about the relation between computations and their realizations. A little thought will show that a computation and its realizations do not have to have the same topology, for example, discrete or continuous. For instance, the discrete computations performed on our digital computers are in fact realized by continuous physical systems obeying Maxwell's equations. The realization is approximate, but exact enough for practical purposes. Conversely a discrete system can approximately realize a continuous system, analogously to numerical integration on a digital computer. In comparing the topologies of the computation and its realization, we must describe the physical process at the relevant level of analysis, for a physical system that is discrete on one level may be continuous on another. (The classification of computations and realizations is discussed in MacLennan, 2004.)

C.4 Expanding the range of computing technologies

C.4.a A vicious cycle

A powerful feedback loop has amplified the success of digital VLSI technology to the exclusion of all other computational technologies. The success of digital VLSI encourages and finances investment in improved tools, technologies, and manufacturing methods, which further promote the success of digital VLSI. Unfortunately this feedback loop threatens to become a vicious cycle. We know that there are limits to digital VLSI technology, and, although estimates differ, we will reach them soon (see Ch. II). We have assumed there will always be more bits and more MIPS, but that assumption is false. Unfortunately, alternative technologies and models of computation remain undeveloped and largely uninvestigated, because the rapid advance of digital VLSI has surpassed them before they could be adequately refined. Investigation of alternative computational technologies is further constrained by the assumption that they must support binary logic, because that is the only way we know how to compute, or because our investment in this model of computation is so large. Nevertheless, we must break out of this vicious cycle or we will be technologically unprepared when digital VLSI finally, and inevitably, reaches its limits.

C.4.b General guidelines

Therefore, as a means of breaking out of this vicious cycle, let us step back and look at computation and computational technologies in the broadest sense. What sorts of physical processes can we reasonably expect to use for computation? Based on the preceding discussion, we can see that any mathematical process, that is, any abstract transformation of abstract objects, is a potential computation. Therefore, in principle, *any reasonably controllable, mathematically described, physical process can be used for computation.* Of course, there are practical limitations on the physical processes usable for computation, but the range of possible technologies is much broader than might be suggested by a narrow conception of computation. Considering some of the requirements for computational technologies will reveal some of the possibilities as well as the limitations.

One obvious issue is speed. The rate of the physical process may be either too slow or too fast for a particular computational application. That it might be too slow is obvious, for the development of conventional computing technology has been driven by speed. Nevertheless, there are many applications that have limited speed requirements, for example, if they are interacting with an environment with its own limited rates. Conversely, these applications may benefit from other characteristics of a slower technology, such as energy efficiency; insensitivity to uncertainty, error, and damage; and the ability to be reconfigured or to adapt or repair itself. Sometimes we simply want to slow a simulated process down so we can observe it. Another consideration that may supersede speed is whether the computational medium is suited to the application: Is it organic or inorganic? Living or nonliving? Chemical, optical, or electrical?

A second requirement is the ability to implement the transducers required for the application. Although computation is theoretically independent of its physical embodiment, its inputs and outputs are not, and some conversions to and from a computational medium may be easier than others. For example, if the inputs and outputs to a computation are chemical, then chemical or molecular computation may permit simpler transducers than electronic computation. Also, if the system to be controlled is biological, then some form of biological computation may suit it best.

Finally, a physical realization should have the accuracy, stability, controllability, etc. required for the application. Fortunately, natural computation provides many examples of useful computations that are accomplished by realizations that are not very accurate, for example, neuronal signals have at most about one digit of precision. Also, nature shows us how systems that are subject to many sources of noise and error may be stabilized and thereby accomplish their purposes.

C.4.c Learning to use new technologies

A key component of the vicious cycle is our extensive knowledge about designing and programming digital computers. We are naturally reluctant to abandon this investment, which pays off so well, but as long as we restrict our attention to existing methods, we will be blind to the opportunities of new technologies. On the other hand, no one is going to invest much time or money in technologies that we don't know how to use. How can we break the cycle?

In many respects natural computation provides the best opportunity, for nature offers many examples of useful computations based on different models from digital logic. When we understand these processes in computational terms, that is, as abstractions independent of their physical realizations in nature, we can begin to see how to apply them to our own computational needs and how to realize them in alternative physical processes. As examples we may take information processing and control in the brain, and emergent self-organization in animal societies, both of which have been applied already to a variety of computational problems (e.g., artificial neural networks, genetic algorithms, ant colony optimization, etc.). But there is much more that we can learn from these and other natural computation systems, and we have not made much progress in developing computers better suited to them. More generally we need to increase our understanding of computation in nature and keep our eyes open for physical processes with useful mathematical structure (Calude et al., 1998; Calude & Paun, 2001). Therefore, one important step toward a more broadly based computer technology will be a knowledge-base of well-matched computational methods and physical realizations.

Computation in nature gives us many examples of the matching of physical processes to the needs of natural computation, and so we may learn valuable lessons from nature. First, we may apply the actual natural processes as realizations of our artificial systems, for example using biological neurons or populations of microorganisms for computation. Second, by understanding the formal structure of these computational systems in nature, we may realize them in alternative physical systems with the same abstract structure. For example, neural computation or insect colony-like self-organization might be realized in an optical system.

C.4.d General-purpose computation

An important lesson learned from digital computer technology is the value of programmable general-purpose computers, both for prototyping special-purpose computers as well as for use in production systems. Therefore to make better use of an expanded range of computational methodologies and technologies, it will useful to have general-purpose computers in which the computational process is controlled by easily modifiable parameters. That is, we will want generic computers capable of a wide range of specific computations under the control of an easily modifiable representation. As has been the case for digital computers, the availability of such general-purpose computers will accelerate the development and application of new computational models and technologies.

We must be careful, however, lest we fall into the "Turing Trap," which is to assume that the notion of universal computation found in Turing machine theory is the appropriate notion in all frames of relevance. The criteria of universal computation defined by Turing and his contemporaries was appropriate for their purposes, that is, studying effective calculability and derivability in formal mathematics. For them, all that mattered was whether a result was obtainable in a finite number of atomic operations and using a finite number of discrete units of space. Two machines, for example a particular Turing machine and a programmed universal Turing machine, were considered to be of the same power if they computed the same function by these criteria. Notions of equivalence and reducibility in contemporary complexity theory are not much different.

It is obvious that there are many important uses of computers, such as real-time control applications, for which this notion of universality is irrelevant. In some of these applications, one computer can be said to emulate another only if it does so at the same speed. In other cases, a general-purpose computer may be required to emulate a particular computer with at most a fixed extra amount of a computational resource, such as storage space. The point is that in the full range of computer applications, in particular in natural computation, there may be considerably different criteria of equivalence than computing the same mathematical function. Therefore, in any particular application area, we must consider in what respects the programmed general-purpose computer must behave the same as the computer it is emulating, and in what respects it may behave differently, and by how much. That is, each notion of universality comes with a frame of relevance, and we must uncover and explicate the frame of relevance appropriate to our application area.

There has been limited work on general-purpose computers in the non-Turing context. For example, theoretical analysis of general-purpose analog computation goes back to Claude Shannon (1941), with more recent work by Pour-El, Lipshitz, and Rubel (Pour-El, 1974b; Lipshitz & Rubel, 1987; Rubel, 1993; Shannon, 1941, 1993). In the area of neural networks we have several theorems based on Sprecher's improvement of the Kolmogorov superposition theorem (Sprecher, 1965), which defines one notion of universality for feed-forward neural networks, although perhaps not a very useful one, and there are several "universal approximation theorems" for neural networks and related computational models (Haykin, 2008, pp. 166–168, 219–220, 236–239, 323–326). Also, there are some CT-relative universality results for molecular computation (Calude & Paun, 2001) and for computation in nonlinear media (Adamatzky, 2001). Finally, we have done some work on general-purpose field computers (MacLennan, 1987, 1990, 1999, 2009a) and on general-purpose computation over second-countable metric spaces (which includes both analog and digital computation) (MacLennan, 2010). In any case, much more work needs to be done, especially towards articulating the relation between notions of universality and their frames of relevance.

It is worth remarking that these new types of general-purpose computers might not be programmed with anything that looks like an ordinary program, that is, a textual description of rules of operation. For example, a *guiding image*, such as a potential surface, might be used to govern a gradient descent process or even a nondeterministic continuous process (MacLennan, 1995, 2004). We are, indeed, quite far from universal Turing machines and the associated notions of programs and computation, but non-Turing models are often more relevant in natural computation and other kinds of unconventional computation.

C.5 Conclusions

The historical roots of Church-Turing computation remind us that the theory exists in a *frame of relevance*, which is not well suited to post-Moore's

Law unconventional computation. Therefore we need to supplement it with new models based on different assumptions and suited to answering different questions. Central issues include real-time response, generality, flexibility, adaptability, and robustness in the presence of noise, uncertainty, error, and damage. Once we understand computation in a broader sense than the Church-Turing model, we begin to see new possibilities for using physical processes to achieve our computational goals. These possibilities will increase in importance as we approach the limits of electronic binary logic as a basis for computation, and they will also help us to understand computational processes in nature.

Chapter II

Physics of Computation

A Energy dissipation

As an introduction to the physics of computation, and further motivation for unconventional computation, we will discuss Michael P. Frank's analysis of energy dissipation in conventional computing technologies (Frank, 2005b). The performance R of a computer system can measured by the number of computational operations executed per unit time. This ratio is the product of the number operations per unit of dissipated energy times the energy dissipation per unit time:

$$R = \frac{N_{\text{ops}}}{t} = \frac{N_{\text{ops}}}{E_{\text{diss}}} \times \frac{E_{\text{diss}}}{t} = F_{\text{E}} \times P_{\text{diss}}. \tag{II.1}$$

Here we have defined P_{diss} to be the power dissipated by the computation and the *energy efficiency* F_{E} to be to be the number of low-level bit operations performed per unit of energy. The key parameter is F_{E}, which is the reciprocal of the energy dissipated per bit operation.

This energy can be estimated as follows. Contemporary digital electronics uses CMOS technology, which represents a bit as the charge on a capacitor. The energy to set or reset the bit is (approximately) the energy to charge the capacitor or the energy dissipated when it discharges. Voltage is energy per unit charge, so the work to move an infinitesimal charge $\mathrm{d}q$ from one plate to the other is $V\mathrm{d}q$, where V is the voltage between the plates. But V is proportional to the charge already on the capacitor, $V = q/C$. So the change in energy is $\mathrm{d}E = V\mathrm{d}q = \frac{q}{C}\mathrm{d}q$. Hence the energy to reach a charge Q is

$$E = \int_0^Q \frac{q}{C}\mathrm{d}q = \frac{1}{2}\frac{Q^2}{C}.$$

Therefore, $E = \frac{1}{2}(CV)^2/C = \frac{1}{2}CV^2$ and $F_\mathrm{E} \approx (1 \text{ op})/(\frac{1}{2}CV^2)$.

Frank observes that Moore's law in the 1985–2005 period was a result of an exponential decrease in C resulting from decreasing feature sizes (since capacitance is proportional to area) and a decrease in logic voltage V from 5V to about 1V (further improving E by a factor of 25). The clock rate also went up with smaller feature sizes. (See Fig. II.1.)

Dennard scaling has been another important factor supporting Moore's law (Dennard et al., 1974). This refers to the fact that voltage and current both scale downward with feature size, and therefore power scales with transistor area. As a consequence, *power density* (the power dissipated per unit area) remains constant as the size of transistors decreases. Furthermore, with smaller transistors, computers could operate faster. We have seen that the energy to change state is $CV^2/2$. Therefore, if the clock frequency is f and transistors switch a fraction α of the time, the energy dissipated will be $\alpha f CV^2/2$. But the capacitance is proportional to the square of the linear dimensions, and decreasing them allows a corresponding increase in switching frequency, and so circuits have been able to operate faster with the same power density.

Unfortunately, Dennard scaling began to break down around 2005. The reason is that there are two sources of power dissipation in integrated circuits. A chip's *dynamic power density* results from its transistors changing state and it benefits from Dennard scaling. However, *static power density* is a result of leakage currents through the transistors when they are not switching. This is a result of quantum mechanical tunneling through the transistor's gate and increases with decreasing gate widths. It does not scale down with feature size; on the contrary it scales up rapidly: a 100-fold decrease in size has resulted in a 10^8 increase in static power density. Previously, with larger transistors, static power density was negligible, but it is now comparable to dynamic power density. By 2006–7 these conditions had created a "power wall" and ended Dennard scaling.

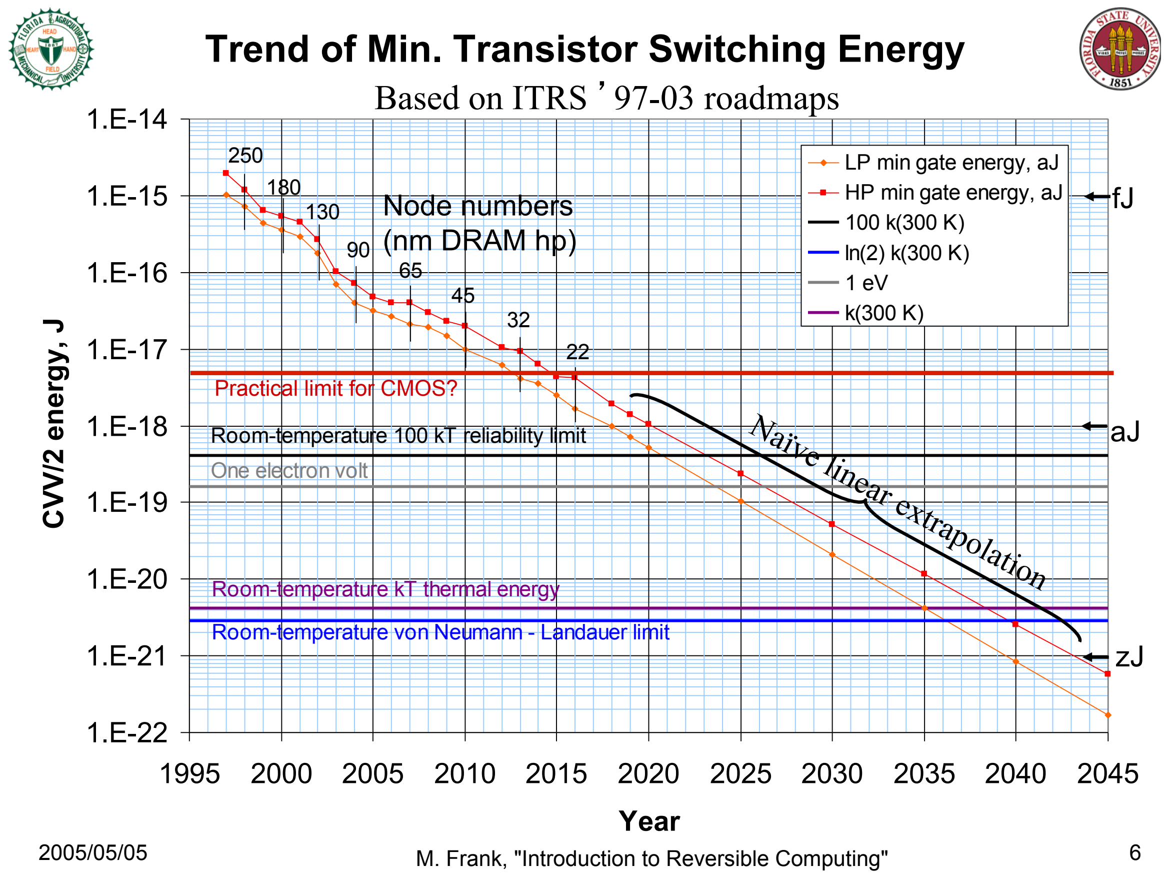


Figure II.1: Historical and extrapolated switching energy. Figure from Frank (2005b, slide 9).

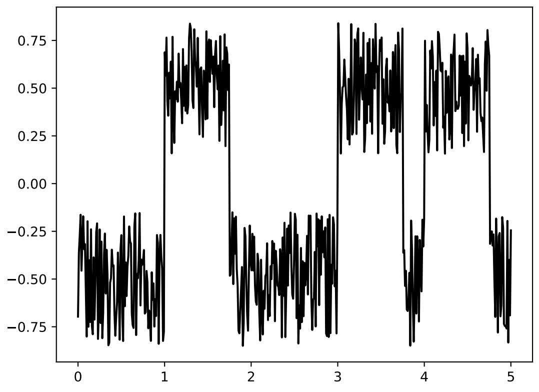


Figure II.2: Depiction of 0-1-0-1-1 pulses in the presence of high thermal noise.

In addition to the preceding issues, there are other limitations on the energy efficiency of computation, for if the signal is too small in comparison with thermal energy, then thermal noise will lead to unreliable operation, because the thermal fluctuations will be of the same order as the signals (Fig. II.2). The thermal energy is $E_T = k_\text{B}T$, where k_B is Boltzmann's constant and T is the absolute temperature. Since $k_\text{B} \approx 8.6 \times 10^{-5}$ eV/K $= 1.38 \times 10^{-23}$J/K, and room temperature $T \approx 300$K, room-temperature thermal energy is

$$E_T = k_\text{B}T \approx 26 \text{ meV} \approx 4.14 \times 10^{-21}\text{J} \approx 4 \text{ zJ}.$$

(Fig. II.1 shows E_T.)

We have seen that $E_\text{sig} = CV^2/2$, but for reliable operation, how big should it be in comparison to E_T? Frank estimates $E_\text{sig} \geq k_\text{B}T \ln R$, where the reliability $R = 1/p_\text{err}$, for a desired probability of error p_err.[1] For example, for a reasonable reliability $R = 2 \times 10^{17}$, $E_\text{sig} \geq 40 k_\text{B}T \approx 1$ eV, which is the energy to move one electron with 1V logic levels. This implies a maximum energy efficiency of

$$F_\text{E} = 1 \text{ op/eV} \approx \frac{1 \text{ op}}{1.6 \times 10^{-19}\text{J}} = 6.25 \times 10^{18} \text{op/J}. \tag{II.2}$$

[1]Frank (2005b, slide 7).

A round $100k_\mathrm{B}T$ corresponds to an error probability of $p_\mathrm{err} = e^{-100} = 3.72 \times 10^{-44}$ (at room temperature). Therefore, a reasonable target for reliable operation is

$$E_\mathrm{sig} \gtrsim 100k_\mathrm{B}T \approx 2.6\ \mathrm{eV} = 414\ \mathrm{zJ}.$$

This, therefore, is an estimate of the minimum energy dissipation per operation for reliable operation using conventional technology. Nevertheless, these conclusions are independent of technology (electronic, optical, carbon nanotube, etc.), since they depend only on relative energy levels for reliable operation.[2]

One apparent solution is to operate at a lower temperature T, but it does not help much, since the effective T has to reflect the environment into which the energy is eventually dissipated (i.e., the energy dissipation has to include the refrigeration to operate below ambient temperature). Another possible solution, operating closer to $k_\mathrm{B}T$ and compensating for low reliability with error-correcting codes, does not help, because we need to consider the *total energy* for encoding a bit. That is, we have to include the additional bits required for error detection and correction.

Frank observed in 2005 that the smallest logic signals were about $10^4 k_\mathrm{B}T$, and therefore that there were only about two orders of magnitude improvement in reliable operation. "A factor of 100 means only around 10 years remain of further performance improvements, given the historical performance doubling period of about 1.5 years. Thus, by about 2015, the performance of conventional computing will stop improving, at least at the device level" (Frank, 2005b).

In fact, these limitations are becoming apparent. By 2011 computer engineers were worrying about "the 3 GHz wall," since computer clock speeds had been stalled at about that rate for five years.[3] Recent processors have gone a little beyond the barrier, but a "power wall" remains, for although individual transistors can be operated at higher speeds, the millions or billions of transistors on a chip dissipate excessive amounts of energy. This presents an obstacle for future supercomputers.

As of June 2018 the fastest supercomputer was Summit (OLCF-4).[4] It is

[2]Frank presentation, "Reversible Computing: A Cross-Disciplinary Introduction" (Beyond Moore), Mar. 10, 2014. $\boxed{\text{put in bib}}$

[3]*Spectrum* (Feb. 2011) `spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers/0` (accessed Aug. 20, 2012).

[4]https://en.wikipedia.org/wiki/Summit_(supercomputer) (accessed Aug. 23, 2018);

rated at 200 petaflops and performed at 122.3 petaflops on LINPACK benchmark. It is also the first supercomputer to reach exascale speed, performing at 1.88 exaops during a genomic analysis. It is expected to reach 3.3 exaops using mixed-precision calculations. Summit has 2,282,544 CPU and GPU processing cores in 4608 nodes, each with two IBM Power9 CPUs and six Nvidia V100 GPUs. The 9216 POWER9 CPUs each have 22 cores (202,752 total), and the 27,648 V100 GPUs each have 80 streaming multiprocessors (SMs), each with 32 FP64 (double-precision) cores, 64 FP32 (single-precision) cores, 64 INT32 cores, and 8 tensor cores.[5] Each node has over 500GB of coherent memory (high-bandwidth memory plus DDR4 SDRAM) addressable by all CPUs and GPUs, plus 800GB of non-volatile RAM. Summit has 250 PB total file storage, occupies 5,600 sq. ft. of floor space (approximately two tennis courts), consumes 13 MW total power and has an energy efficiency F_{E} = 13.889 GFlops/watt (the fifth most energy efficient supercomputer), which is about 72 pJ/flop. Its power consumption is comparable to a town of 3436 households and requires 4,000 gallons of water per minute for cooling.

To convert floating-point operations to basic logic operations, including all the overhead etc., one conversion estimate is 10^7 to 10^8 ops/flop.[6] Therefore, we can compare the theoretical best energy efficiency (Eq. II.2), $F_{\mathrm{E}}^{-1} = 1.6 \times 10^{-7}$pJ/op $\approx$ 1.6 to 16 pJ/flop, with the 72 pJ/flop of Summit. The gap is only about one order of magnitude. Indeed, it has been estimated that scaling up current technology to 1 exaflops would consume 1.5 GW, more than 0.1% of US power grid.[7] This is impractical.

It might be possible to get energy consumption down to 5 to 10 pJ/flop, but "the energy to perform an arithmetic operation is trivial in comparison with the energy needed to shuffle the data around, from one chip to another, from one board to another, and even from rack to rack."[8] Indeed, due to the difficulty of programming parallel computers, and due to delays in internal data transmission, it is difficult to use more than 5% to 10% of a supercom-

https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/ (accessed Aug. 23, 2018)]

[5]https://www.olcf.ornl.gov/for-users/system-user-guides/summit/nvidia-v100-gpus/

[6]And so this is one estimate of the difference in time scale between computational abstractions and the logic that implements them, which was discussed in Ch. I (p. 5).

[7]*Spectrum* (Feb. 2011) `spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers/0` (accessed Aug. 20, 2012).

[8]*Spectrum* (Feb. 2011) `spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers/0` (accessed Aug. 20, 2012).

puter's capacity for any extended period; most of the processors are idling.[9] So with those 2,282,544 cores, most of the time about *two million* of them are idle! There has to be a better way.

[9] *Spectrum* (Feb. 2011) `spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers/0` (accessed 2012-08-20).

B Thermodynamics of computation

"Computers may be thought of as engines for transforming free energy into waste heat and mathematical work." — Bennett (1982)

B.1 Von Neumann-Landauer Principle

B.1.a Information and entropy

We will begin with a quick introduction or review of the entropy concept; we will look at it in more detail soon (Sec. B.2). The *information content* of a signal (message) measures our "surprise," that is, how unlikely it is. $I(s) = -\log_b \mathcal{P}\{s\}$, where $\mathcal{P}\{s\}$ is the probability of signal s. We take logs so that the information content of independent signals is additive. We can use any base, and for $b = 2, e$, and 10, the corresponding units are *bits* (or *shannons*), *nats*, and *dits* (also, *hartleys, bans*). Therefore, if a signal has a 50% probability, then it conveys one bit of information. The *entropy* of a distribution of signals is their average information content:

$$H(S) = \mathcal{E}\{I(s) \mid s \in S\} = \sum_{s \in S} \mathcal{P}\{s\} I(s) = -\sum_{s \in S} \mathcal{P}\{s\} \log \mathcal{P}\{s\}.$$

Or more briefly, $H = -\sum_k p_k \log p_k$. For example, if $\mathcal{P}\{1\} = 1/16$ and $\mathcal{P}\{0\} = 15/16$, we will receive, on the average, $H = 0.3$ bits of information.

According to a well-known story, Shannon was trying to decide what to call this quantity and had considered both "information" and "uncertainty." Because it has the same mathematical form as statistical entropy in physics, von Neumann suggested he call it "entropy," because "nobody knows what entropy really is, so in a debate you will always have the advantage."[10] (This is one version of the quote.) I will call it *information entropy* when I need to distinguish it from the thermodynamical concept.

An important special case is the entropy of a uniform distribution. If there are N signals that are all equally likely, then $H = \log N$. Therefore, if we have eight equally likely possibilities, the entropy is $H = \lg 8 = 3$ bits.[11]

[10]`https://en.wikipedia.org/wiki/History_of_entropy` (accessed 2012-08-24). Ralph Hartley laid the foundations of information theory in 1928, on which Claude Shannon built his information theory in 1948.

[11]I use the notations $\lg x = \log_2 x$ and $\ln x = \log_e x$.

A uniform distribution maximizes the entropy (and minimizes the ability to guess).

In computing, we are often concerned with the *state* of the computation, which is realized by the state of a physical system. Consider a physical system with three *degrees of freedom* (*DoF*), each with 1024 possible values. There are $N = 1024^3 = 2^{30}$ possible states, each describable by three 10-bit integers. If we don't care about the distance between states (i.e., distance on each axis), then states can be specified equally well by six 5-bit numbers or one 30-bit number, etc. (or ten digits, since $30 \log_{10} 2 \approx 9.03$ digits). Any scheme that allows us to identify all 2^{30} states will do. We can say that there are 30 *binary degrees of freedom*.

In computing we often have to deal with things that grow exponentially or are exponentially large (due to combinatorial explosion), such as solution spaces. (For example, NP problems are characterized by the apparent necessity to search a space that grows exponentially with problem size.) In such cases, we are often most concerned with the *exponents* and how they relate. Therefore it is convenient to deal with their logarithms (i.e., with logarithmic quantities). The logarithm represents, in a scale-independent way, the degrees of freedom generating the space.

Different logarithm bases amount to different units of measurement for logarithmic quantities (such as information and entropy). As with other quantities, we can leave the units unspecified, so long as we do so consistently. I will use the notation "$\log x$" for an *unspecific logarithm*, that is, a logarithm with an unspecified base.[12] When I mean a specific base, I will write $\ln x$, $\lg x$, $\log_{10} x$, etc. Logarithms in specific bases can be defined in terms of unspecific logarithms as follows: $\lg x = \log x / \log 2$, $\ln x = \log x / \log e$, etc. (The units can be defined $\text{bit} = \log 2$, $\text{nat} = \log e$, $\text{dit} = \log 10$, etc.)

B.1.b The von Neumann-Landauer bound

Thermodynamic entropy is unknown information residing in the physical state. The macroscopic thermodynamic entropy S is related to microscopic information entropy H by Boltzmann's constant, which expresses the entropy in thermodynamical units (energy over temperature). If H is measured in nats, then $S = k_\text{B} H = k_\text{B} \ln N$, for N equally likely states. When using

[12]Frank (2005a) provides a formal definition for the *indefinite logarithm*. I am using the idea less formally, an "unspecific logarithm," whose base is not mentioned. This is a compromise between Frank's concept and familiar notation; we'll see how well it works!

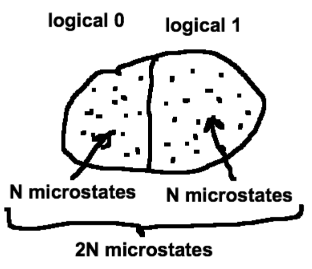


Figure II.3: Physical microstates representing logical states. Setting the binary device decreases the entropy: $\Delta H = \lg N - \lg(2N) = -1$ bit. That is, we have one bit of information about its microstate.

unspecific logarithms, I will drop the "B" subscript: $S = kH = k \log N$. The physical dimensions of entropy are usually expressed as energy over temperature (e.g., joules per kelvin), but the dimensions of temperature are energy per degree of freedom (measured logarithmically), so the fundamental dimension of entropy is degrees of freedom, as we would expect. (There are technical details that I am skipping.)

Consider a macroscopic system composed of many microscopic parts (e.g., a fluid composed of many molecules). In general a very large number of *microstates* (or *microconfigurations*) — such as positions and momentums of molecules — will correspond to a given *macrostate* (or *macroconfiguration*) — such as a combination of pressure and termperature. For example, with $m = 10^{20}$ particles we have $6m$ degrees of freedom, and a $6m$-dimensional *phase space* of its possible microstates.

Now suppose we partition the microstates of a system into two macroscopically distinguishable macrostates, one representing 0 and the other representing 1. For example, whether the electrons are on one plate of a capacitor or the other could determine whether a 0 or 1 bit is stored on it. Next suppose N microconfigurations correspond to each macroconfiguration (Fig. II.3). This could be all the positions, velocities, and spins of the many electrons, which we don't care about and cannot control individually. If we confine the system to one half of its microstate space in order to represent a 0 or a 1, then the entropy (average uncertainty in identifying the microstate) will decrease by one bit. We don't know the exact microstate, but at least

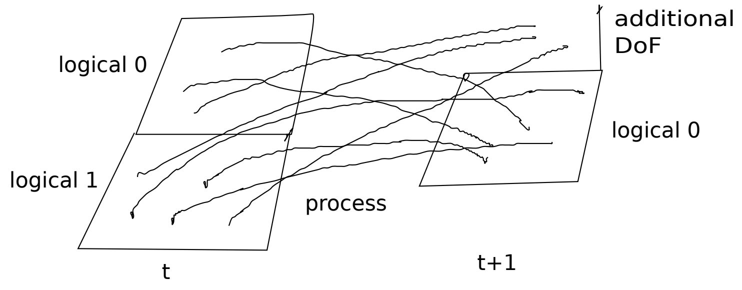


Figure II.4: Thermodynamics of erasing a bit. On the left is the initial state (time t), which may be logical 0 or logical 1; on the right (time $t+1$) the binary device has been set to logical 0. In each case there are N microstates representing each prior state, so a total of $2N$ microstates. However, at time $t+1$ the system must be in one of N posterior microstates. Therefore N of the microstate trajectories must exit the defined region of phase space by expanding into additional, uncontrolled degrees of freedom. Therefore entropy of the environment must increase by at least $\Delta S = k \log(2N) - k \log N = k \log 2$. We lose track of this information because it passes into uncontrolled degrees of freedom.

---

we know which half of the state-space it is in.

In general, in physically realizing a computation we distinguish *information-bearing degrees of freedom* (IBDF), which we control and use for computation, from *non-information-bearing degrees of freedom* (NIBDF), which we do not control and are irrelevant to the computation (Bennett, 2003).

Consider the process of erasing or clearing a bit (i.e., setting it to 0, no matter what its previous state): we are losing one bit of physical information. The physical information still exists, but we have lost track of it; it has been *thermalized*.

Suppose we have N physical microstates per logical macrostate (logical 0 or logical 1). Before the bit is erased it can be in one of $2N$ possible microstates, but there are only N microstates representing its final state. The laws of physics are reversible,[13] so they cannot lose any information. Since physical information can't be destroyed, it must go into NIBDF (e.g., the environment or thermal motion of the atoms) (Fig. II.4). The trajectories

[13]This is true in both classical and quantum physics. In the latter case, we cannot have $2N$ quantum states mapping reversibly into only N quantum states.

have to expand into other degrees of freedom (NIBDF) to maintain the phase space volume.

The information lost, or dissipated into NIBDF (typically as heat), is $\Delta S = k\log(2N) - k\log N = k\log 2$. (In physical units this is $\Delta S = k_{\text{B}} \ln 2 \approx 10^{-23}$J/K.) Therefore the increase of energy in the device's environment is $\Delta Q = \Delta S \times T_{\text{env}} = k_{\text{B}} T_{\text{env}} \ln 2 \approx 0.7kT_{\text{env}}$. At $T_{\text{env}} = 300$K, $k_{\text{B}} T_{\text{env}} \ln 2 \approx 18$ meV $\approx 3 \times 10^{-9}$ pJ $= 3$ zJ. We will see that this is the minimum energy dissipation for any irreversible operation (such as erasing a bit); it is called the *von Neumann–Landauer (VNL) bound* (or sometimes simply the *Landauer bound*). Von Neumann suggested the idea in 1949, but it was published first by Rolf Landauer (IBM) in 1961.[14] Recall that for reliable operation we need minimum logic levels around $40k_{\text{B}}T_{\text{env}}$ to $100k_{\text{B}}T_{\text{env}}$, which is two orders of magnitude above the von Neumann–Landauer limit of $0.7k_{\text{B}}T_{\text{env}}$. "From a technological perspective, energy dissipation per logic operation in present-day silicon-based digital circuits is about a factor of 1,000 greater than the ultimate Landauer limit, but is predicted to quickly attain it within the next couple of decades" (Berut et al., 2012). That is, current circuits have signal levels of about 18 eV, which we may compare to the VNL, 18 meV.

In research reported in 2012 Berut et al. (2012) confirmed experimentally the Landauer Principle and showed that it is the erasure that dissipates energy. They trapped a 2μ silica ball in either of two laser traps, representing logical 0 and logical 1. For storage, the potential barrier was greater than $8k_{\text{B}}T$, and for erasure, the barrier was lowered to $2.2k_{\text{B}}T$ by decreasing the power of the lasers and by tilting the device to put it into the logical 0 state (see Fig. II.5). At these small sizes, heat is a stochastic property, so the dissipated heat was computed by averaging the trajectory of the particle over multiple trials:

$$\langle Q \rangle = \left\langle -\int_0^\tau \dot{x}(t) \frac{\partial U(x,t)}{\partial x} \text{d}t \right\rangle_x .$$

(The angle brackets means "average value.") Complete erasure results in the ball being in the logical 0 state; incomplete erasure results in it being in the logical 0 state with probability p. They established a generalized Landauer bound in which the dissipated heat depends on the completeness of erasure:

$$\langle Q \rangle^p_{\text{Landauer}} = kT[\log 2 + p\log p + (1-p)\log(1-p)] = kT[\log 2 - H(p, 1-p)].$$

[14]See Landauer (1961), reprinted in Leff & Rex (1990) and Leff & Rex (2003), which include a number of other papers analyzing the VNL principle.

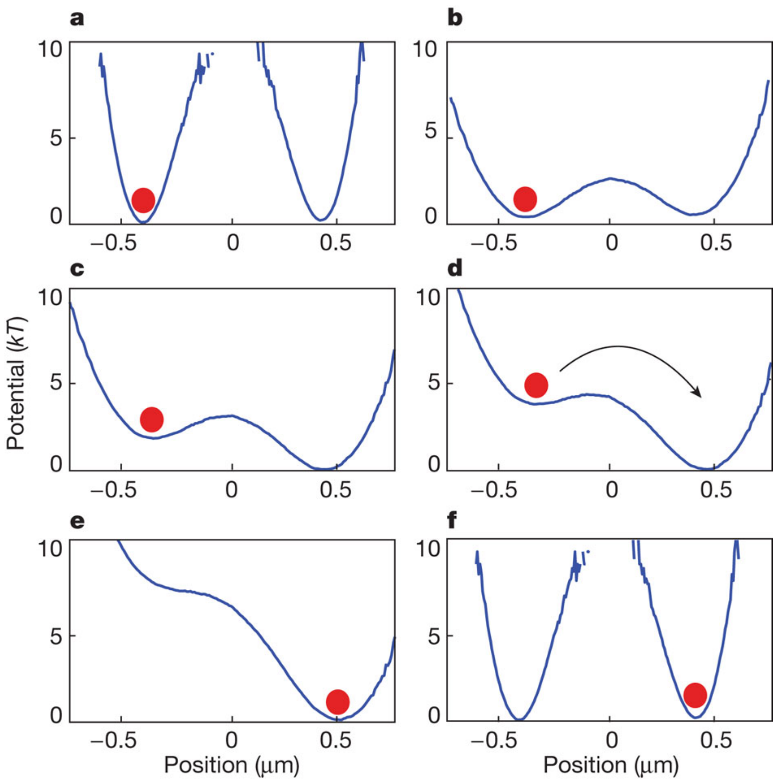


Figure II.5: Erasing a bit by changing potential barrier. (Figure from Berut et al. (2012).)

Therefore, for $p = 1$, heat dissipation is $kT \log 2$, but for $p = 1/2$ (ineffective erasure) no heat needs to be dissipated. Notice how the dissipated energy depends on the entropy $H(p, 1-p)$ of the final macrostate.

B.1.c Irreversible operations

Suppose the phase space is divided into M macrostates of size $N_1, N_2, \ldots, N_M$, where $N = N_1 + N_2 + \cdots + N_M$. Let p_{ij} be the probability the device is in microstate i of macrostate j. The total entropy is

$$S = -k \sum_{ij} p_{ij} \log p_{ij}. \tag{II.3}$$

We can separate this into the *macroscopic entropy* associated with the macrostates (IBDF) and the *microscopic entropy* associated with the microstates (NIBDF). Now let $P_j = \sum_{i=1}^{N_j} p_{ij}$ be the probability of being in macrostate j. Then Eq. II.3 can be rearranged (Exer. II.1):

$$S = -k \sum_j P_j \log P_j - k \sum_j P_j \sum_{i=1}^{N_j} \frac{p_{ij}}{P_j} \log \frac{p_{ij}}{P_j} = S_{\mathrm{i}} + S_{\mathrm{h}}. \tag{II.4}$$

The first term is the macrostate entropy (IBDF):

$$S_{\mathrm{i}} = -k \sum_j P_j \log P_j.$$

The second is the microstate entropy (NIBDF):

$$S_{\mathrm{h}} = -k \sum_j P_j \sum_{i=1}^{N_j} \frac{p_{ij}}{P_j} \log \frac{p_{ij}}{P_j}.$$

Note that the ratios in the inner summation are essentially conditional probabilities, and that the inner summation is the conditional entropy given that you are in macrostate j.

When we erase a bit, we go from a maximum S_{i} of 1 bit (if 0 and 1 are equally likely), to 0 bits (since there is no uncertainty). Thus we lose one bit of information, and the macrostate entropy decreases $\Delta S_{\mathrm{i}} = -k \log 2$. (The actual entropy decrease can be less than 1 bit if the 0 and 1 are not equally likely initial states.) Since according to the Second Law of Thermodynamics

$\Delta S \geq 0$, we have a corresponding minimum increase in microstate entropy, $\Delta S_{\mathrm{h}} \geq k \log 2$. Typically this is dissipated as heat, $\Delta Q \geq kT \log 2$. The information becomes inaccessible and unusable.

The standard logic gates (AND, OR, XOR, NAND, etc.) have two input bits and one output bit. Therefore the output will have lower entropy than the input, and so these gates must dissipate at least 1 bit of entropy, $kT \log 2$ energy. Consider, for example, AND. If the four inputs 00, 01, 10, 11, are equally likely, then the input entropy is $H_{\mathrm{i}} = 2$ bits. However the output entropy will be $H_{\mathrm{o}} = -(1/4)\lg(1/4) - (3/4)\lg(3/4) = 0.811$, so the entropy lost is 1.189 bits. To compute the dissipated energy in Joules, multiply by $\ln 2$ to convert bits to nats (or shannons to hartleys):

$$\Delta Q = T\Delta S \geq -Tk_{\mathrm{B}}(H_{\mathrm{o}} - H_{\mathrm{i}}) \ln 2 \approx 1.189 k_{\mathrm{B}} T \ln 2 \approx 0.83 k_{\mathrm{B}} T.$$

For each gate, we can express H_{o} in terms of the probabilities of the inputs and compute the decrease from H_{i} (exercise). If the inputs are not equally likely, then the input entropy will be less than 2 bits, but we will still have $H_{\mathrm{i}} > H_{\mathrm{o}}$ and energy will be dissipated. (Except in a trivial, uninteresting case. What is it?)

More generally, any irreversible operation (non-invertible function) will lose information, which has to be dissipated into the environment. That is, it is irreversible because it loses information, and every time we lose information, we lose energy. If the function is not one-to-one (injective), then at least two inputs map to the same output, and so information about the inputs is lost. For example, changing a bit, that is, overwriting a bit with another bit, is a fundamental irreversible operation, subject to the VNL limit. Therefore, the assignment operation is bound by it. Also, when two control paths join, we forget where we came from, and so again we must dissipate at least a bit's worth of entropy (Bennett, 2003). These considerations suggest that reversible operations might not be subject to the VNL limit, and this is in fact the case, as we will see.

The preceding observations have important connections with the problem of Maxwell's Demon and its resolution. Briefly, the demon has to reset its mechanism after each measurement in preparation for the next measurement, and this dissipates at least $kT \log 2$ energy into the heat bath for each decision that it makes. That is, the demon must "pay" for any information that it acquires. Therefore, the demon cannot do useful work. Further discussion is outside the scope of this book, so if you are interested, please see Leff &

Rex (2003) and Leff & Rex (1990) (which have a large intersection), in which many of the papers on the topic are collected.

B.2 Mechanical and thermal modes

We need to understand in more detail the reason for the increase of entropy and its relation to reversibility and irreversibility.[15] We can classify systems according to their size and completeness of specification:

specification:	complete	incomplete	
size:	~ 1	~ 100	$\sim 10^{23}$
laws:	dynamical	statistical	thermodynamical
reversible:	yes	no	no

Dynamical systems have a relatively small number of particles or degrees of freedom and can be completely specified. For example, we may have 6 degrees of freedom for each particle (x, y, z, p_x, p_y, p_z). We can prepare an *individual* dynamical system in an initial state and expect that it will behave according to the dynamical laws that describe it. Think of billiard balls or pucks on a frictionless surface, or electrons moving through an electric or magnetic field. So far as we know, the laws of physics at this level (either classical or quantum) are reversible.

If there are a large number of particles with many degrees of freedom (several orders of magnitude), then it is impractical to specify the system completely. Moreover, small errors in the initial state will have a larger effect, due to complex interaction of the particles. Also, there are small effects from interaction with the environment. Therefore we must resort to *statistical laws* and we have a *statistical system*. If we can't *manage* all the degrees of freedom, then we average over those that we can't manage. Statistical laws don't tell us how an individual system will behave (there are too many sources of variability), but they tell us how *ensembles* of similar systems (or preparations) behave. We can talk about the average behavior of such systems, but we also have to consider the variance, because unlikely outcomes are not impossible. For example, tossing 10 coins has a probability of 1/1024 of turning up all heads; this is small, but not negligible. Statistical laws are in general irreversible (because there are many ways to get to the same state).

[15]This section is based primarily on Edward Fredkin and Tommaso Toffoli's "Conservative logic" (Fredkin & Toffoli, 1982).

Finally we consider *thermodynamical systems*. Macroscopic systems have a very large number of particles ($\sim 10^{23}$) and a correspondingly large number of degrees of freedom. We call these "Avogadro scale" numbers. It is important to grasp how truly enormous these numbers are; in comparison (Tong, 2012, p. 37): the number of grains of sand on all beaches $\approx 10^{18}$, the number of stars in our galaxy $\approx 10^{11}$, and the number of stars in the visible universe $\approx 10^{22}$, but the number of water molecules in a cup of tea $\approx 10^{23}$. Obviously such systems cannot be completely specified (we cannot describe the initial state and trajectory of every atom). Indeed, because information is physical, it is *physically impossible* to "know" (i.e., to represent physically) the physical state of a macroscopic system (i.e., to use the macrostates of one system to represent the microstates of another macroscopic system).

We can derive statistical laws for thermodynamical systems, but in these cases most macrostates become so improbable that they are virtually impossible (for example, the cream unmixing from your coffee). The central limit theorem shows that the variance decreases with n: By the law of large numbers (specifically, Bernoulli's Theorem), variance in the relative number of successes is $\sigma^2 = p(1-p)/n$, that is, $\sigma^2 = 1/4n$ for $p = 1/2$. Therefore the standard deviation is $\sigma = 1/(2n^{1/2})$. For example, for $n = 10^{22}$, $\sigma = 5 \times 10^{-10}$. Also, 99.99% of the probability density is within $4\sigma = 2 \times 10^{-9}$ (see Thomas, *Intro. Appl. Prob. & Rand. Proc.*, p. 111). The probability of deviating more than ϵ from the mean decreases exponentially with n: $\frac{1}{6}\exp(-\epsilon^2 n/2) + \frac{1}{2}\exp(-2\epsilon^2 n/3)$.

In the thermodynamic limit, the *likely* is *inevitable*, and the *unlikely* is *impossible*. In these cases, *thermodynamical laws* describe the virtually deterministic (but *irreversible*) dynamics of the system.

Sometimes in a macroscopic system we can separate a small number of *mechanical modes* (DoF) from the *thermal modes*. "Mechanical" here includes "electric, magnetic, chemical, etc. degrees of freedom." The mechanical modes are strongly coupled to each other but weakly coupled to the thermal modes. For example, in a rigid body (e.g., a bullet, a billiard ball) the mechanical modes are the positions and momentums of the particles in the body. Thus the mechanical modes can be treated exactly or approximately independently of the thermal modes. In the ideal case the mechanical modes are completely decoupled from the thermal modes, and so the mechanical modes can be treated as an isolated (and reversible) dynamical system. The energy of the mechanical modes (once initialized) is independent of the energy ($\sim kT$) of the thermal modes. The mechanical modes are *conservative*;

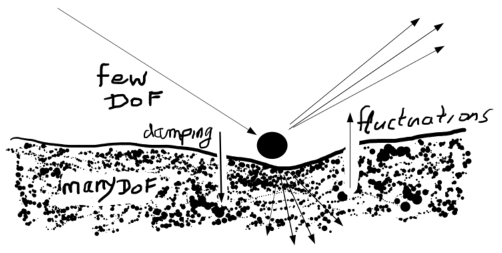


Figure II.6: Complementary relation of damping and fluctuations. The macroscopic ball has few mechanical degrees of (e.g., 6), but during the collision there is an interaction of the enormously many degrees of freedom of the microstates of the ball and those of the wall.

they don't dissipate any energy. (This is what we have with *elastic* collisions.) This is the approach of reversible computing.

Suppose we *want* irreversible mechanical modes, e.g., for implementing irreversible logic. The underlying physics is reversible, and so the information lost by the mechanical modes cannot simply disappear; it must be transferred to the thermal modes. This is *damping*: Information in the mechanical modes, where it is accessible and usable, is transferred to the thermal modes, where it is inaccessible and unusable. This is the *thermalization* of information, the transfer of physical information from accessible DoF to inaccessible DoF. But the interaction is bidirectional, so *noise* (uncontrolled DoF) will flow from the thermal modes back to the mechanical modes, making the system nondeterministic.

As Feynman said, "If we know where the *damping* comes from, it turns out that that is also the source of the *fluctuations*" [Feynman, 1963]. Think of a bullet ricocheting off a flexible wall filled with sand. It dissipates energy into the sand and also acquires noise in its trajectory (see Fig. II.6). To avoid nondeterminacy, the information may be encoded redundantly so that the noise can be filtered out. For example, the signal can be encoded in multiple mechanical modes, over which we take a majority vote or an average. Or the signal can be encoded with energy much greater than any one of the thermal modes, $E \gg kT$, to bias the energy flow from mechanical to thermal (preferring dissipation over noise). Free energy must be used to refresh the mechanical modes, and heat must be flushed from the thermal modes. "[I]mperfect knowledge of the dynamical laws leads to uncertainties in the behavior of a system comparable to those arising from imperfect knowledge of its initial conditions... Thus, the same regenerative processes which help overcome thermal noise also permit reliable operation in spite of substantial fabrication tolerances." (Fredkin & Toffoli, 1982)

Damped mechanisms have proved to be very successful, but they are inherently inefficient.

> In a damped circuit, the rate of heat generation is proportional to the number of computing elements, and thus approximately to the useful volume; on the other hand, the rate of heat removal is only proportional to the free *surface* of the circuit. As a consequence, computing circuits using damped mechanisms can grow arbitrarily large in two dimensions only, thus precluding the much tighter packing that would be possible in three dimensions. (Fredkin & Toffoli, 1982)

In an extreme case (force of impact $>$ binding forces), a signal's interacting with the environment might cause it to lose its coherence (the correlation of its constituent DoFs, such as the correlation between the positions and momenta of its particles). The information implicit in the mechanical modes is lost into the thermal modes.

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.[16] 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. If we do things differently, 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. Moreover, 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: 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 see, Fredkin and Toffoli described computation with idealized, perfectly

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

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 the 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 (Ch. IV).

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 accommodate a small degree of irreversibility (see Fig. II.7). In these cases,

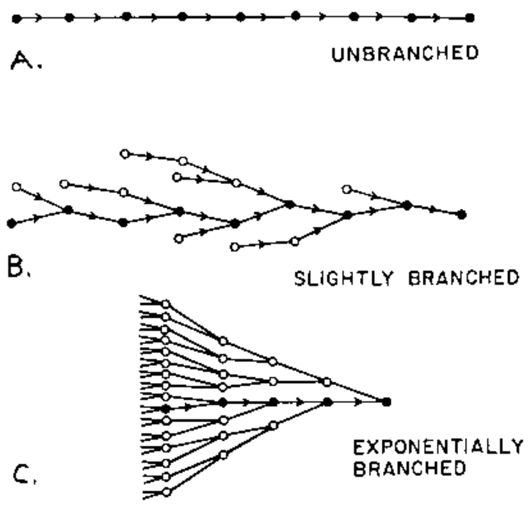


Figure II.7: Different degrees of logical reversibility. [from Bennett (1982)]

there are a few backward detours (e.g., it might be equally likely to go in one forward 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 predecessors of the statement following 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, $c_{\text{E}} \approx (40kT)/(1\text{kHz}) = 40\times 26\text{ meV/kHz} \approx 1\text{ eV/kHz}$ (since at room temperature, $k_{\text{B}}T \approx 26$ meV: see Sec. A, p. 32). 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 times 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.[17] 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.

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

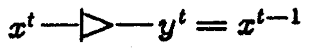


Figure II.8: Symbol for unit wire. (Fredkin & Toffoli, 1982)

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*.

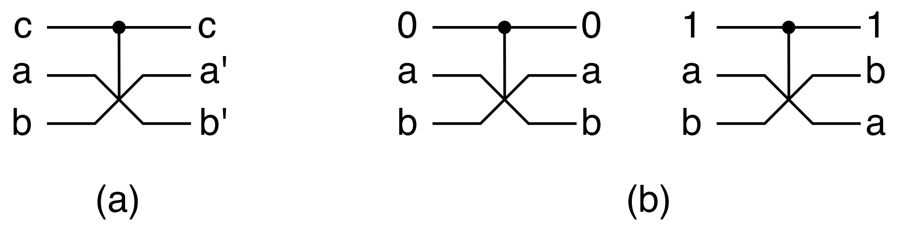


Figure II.9: Fredkin gate or CSWAP (conditional swap): (a) symbol and (b) operation.

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:

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

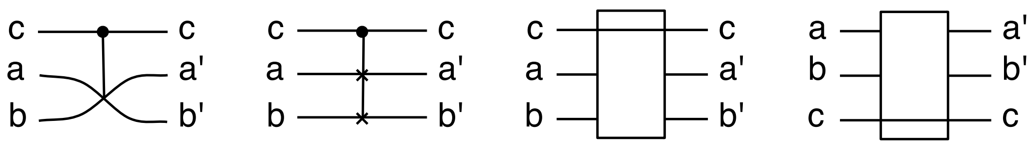


Figure II.10: Alternative notations for Fredkin gate.

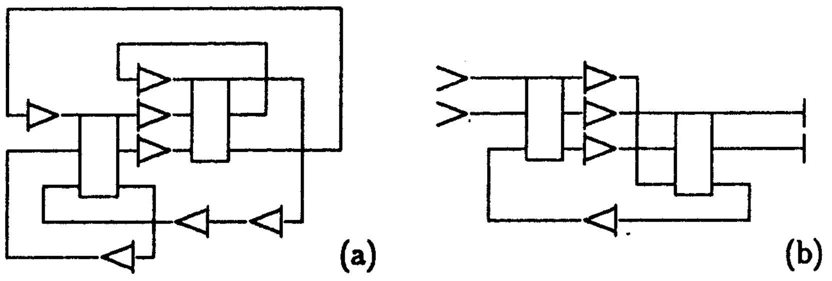


Figure II.11: "(a) closed and (b) open conservative-logic circuits." (Fredkin & Toffoli, 1982)

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

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

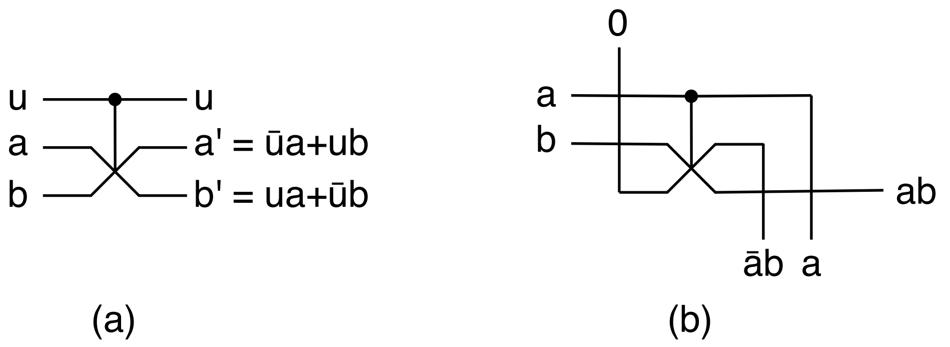


Figure II.12: (a) Logical behavior of Fredkin gate. (b) Implementation of AND gate by Fredkin gate by constraining one input to 0 and discarding two "garbage" outputs.

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.13 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.

With these constructions one 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 extra constant inputs and garbage outputs. It's better to design the conservative circuit

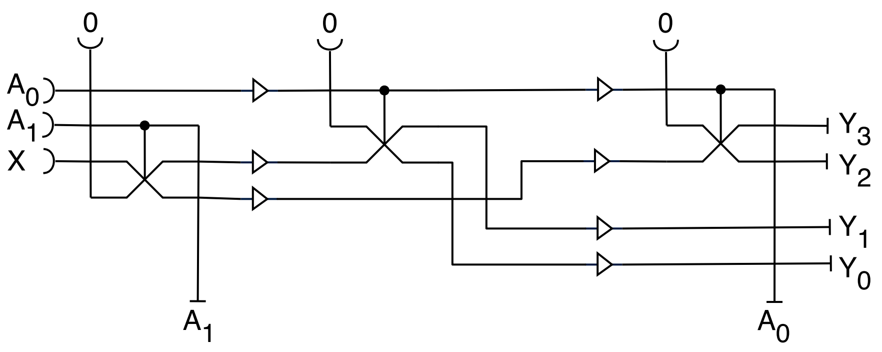


Figure II.13: 1-line-to 4-line demultiplexer. The address bits $A_1A_0 = 00, 01, 10, 11$ direct the data bit X into Y_0, Y_1, Y_2 or Y_3, respectively. Note that each Fredkin gate uses an address bit to route X into either of two wires. (Adapted from circuit in Fredkin & Toffoli (1982).)

from scratch. Nevertheless, 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 function by providing appropriate constants from a *source* and ignoring some of the outputs, the *sink*, which are considered *garbage* (Fig. II.14). 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.

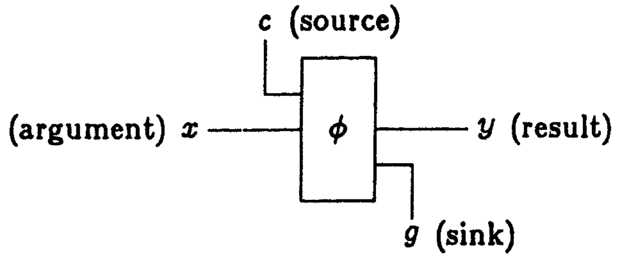


Figure II.14: "Realization of f by ϕ using source and sink. The function $\phi : (c, x) \mapsto (y, g)$ is chosen so that, for a particular value of $c, y = f(x)$." (Fredkin & Toffoli, 1982)

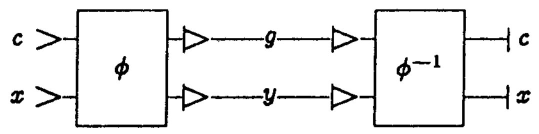


Figure II.15: Composition of combinational conservative-logic network with its inverse to consume the garbage. (Fredkin & Toffoli, 1982)

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)

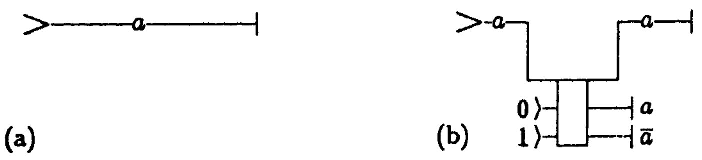


Figure II.16: The "spy circuit" for tapping into the output. (Fredkin & Toffoli, 1982)

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 $(a, 0, 1) \mapsto (a, a, \bar{a})$. 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 $00 \cdots 0011 \cdots 11$ 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 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 $00 \cdots 0011 \cdots 11$ in the lower register has been rearranged to yield the result and its complement, $y\bar{y}$. Restoring the $0 \cdots 01 \cdots 1$ inputs for another computation dissipates energy, but the amount of energy depends

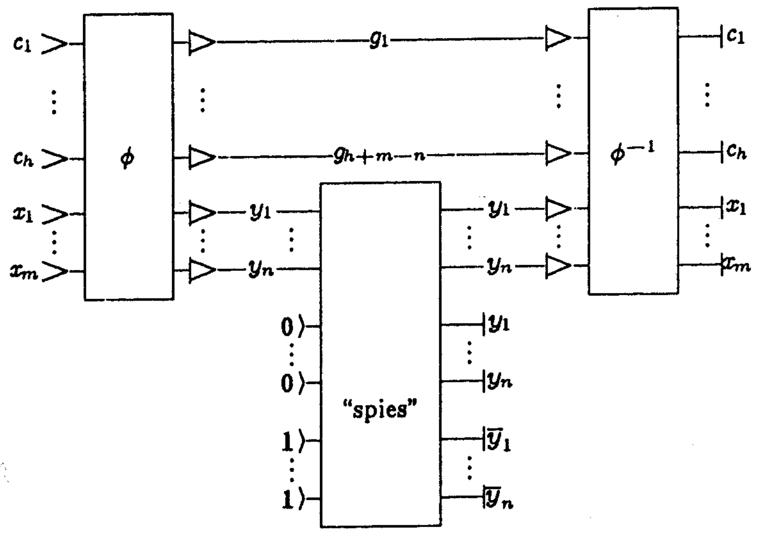


Figure II.17: Garbageless circuit. (Fredkin & Toffoli, 1982)

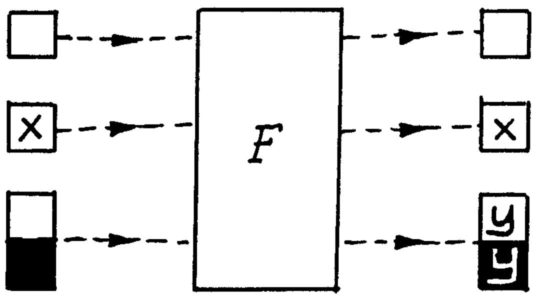


Figure II.18: "The conservative-logic scheme for garbageless computation. Three data registers are 'shot' through a conservative-logic black-box F. The register with the argument, x, is returned unchanged; the clean register on top of the figure, representing an appropriate supply of input constants, 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)

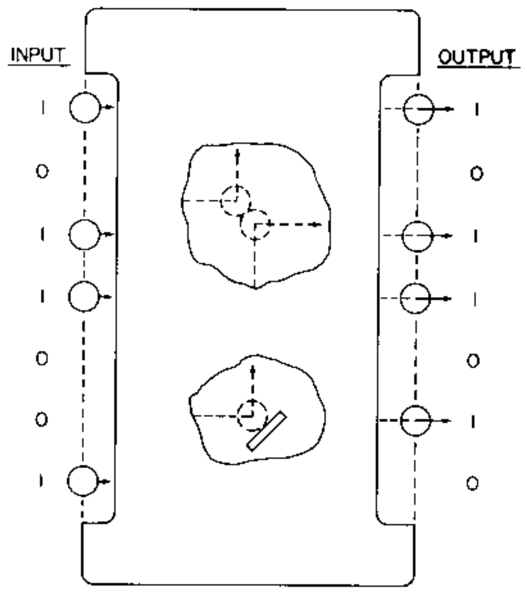


Figure II.19: Overall structure of ballistic computer. (Bennett, 1982)

on the size of the output (number of bits), not the amount of computation.[18]

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 of each other

[18]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.)

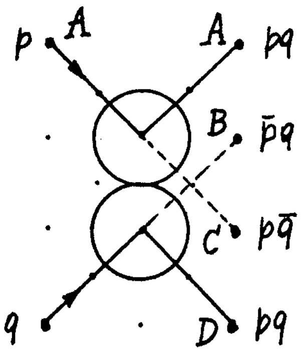


Figure II.20: "Billiard ball model realization of the interaction gate." (Fredkin & Toffoli, 1982)

and off of fixed reflectors, and this interaction 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 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

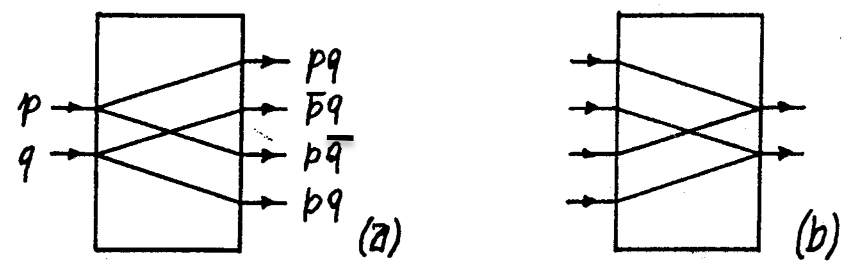


Figure II.21: "(a) The interaction gate and (b) its inverse." (Fredkin & Toffoli, 1982)

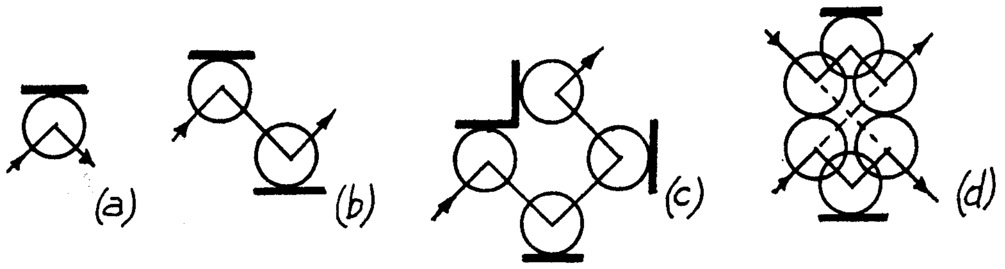


Figure II.22: "The mirror (indicated by a solid dash) can be used to deflect a ball's path (a), introduce a sideways shift (b), introduce a delay (c), and realize nontrivial crossover (d)." (Fredkin & Toffoli, 1982)

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 are 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.

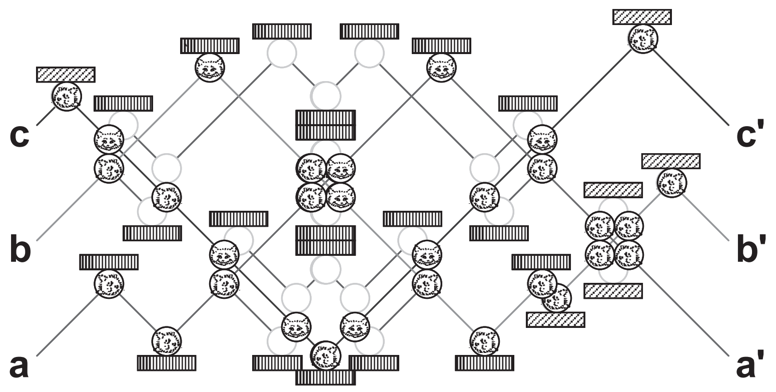


Figure II.23: Realization of the Fredkin gate in terms of multiple interaction gates. [NC]

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 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.

D Exercises

Exercise II.1 Show that Eq. II.3 (p. 42) can be rearranged to Eq. II.4 (p. 42).

Exercise II.2 Show that the Fredkin gate is reversible.

Exercise II.3 Show that the Fredkin gate implements $(a, 0, 1) \mapsto (a, a, \bar{a})$ (the "spy circuit").

Exercise II.4 Show how to use a single Fredkin gate to implement each of the NOT, OR, and FAN-OUT gates. (FAN-OUT copies its input value to two output wires.) Note! You cannot use FAN-IN or FAN-OUT in your implementations, only Fredkin gates.

Exercise II.5 Use the Fredkin gate to implement XOR. Minimize the number of Fredkin gates you use.

Exercise II.6 Give a truth table for a reversible gate that is not conservative (does not preserve the total number of 1s and 0s). It should have the same number of outputs as inputs.

Exercise II.7 Give a truth table for a 2-input / 2-output gate that is conservative but not reversible.

Exercise II.8 Show for the eight possible inputs that Fig. II.13 is a correct implementation of a 1-line to 4-line demultiplexer. That is, show in each of the four cases $A_1A_0 = 00, 01, 10, 11$ the bit $X = 0$ or 1 gets routed to Y_0, Y_1, Y_2, Y_3, respectively. You can use a Boolean algebra proof, if you prefer.

Exercise II.9 Show that implementation of a J-$\bar{\text{K}}$ flip-flop with Fredkin gates in Fig. II.24 is correct. A J-$\bar{\text{K}}$ flip-flop has the following behavior:

J	$\bar{\text{K}}$	behavior
0	0	reset, Q $\to$ 0
0	1	hold, Q doesn't change
1	0	toggle, Q $\to$ $\bar{\text{Q}}$
1	1	set, Q $\to$ 1

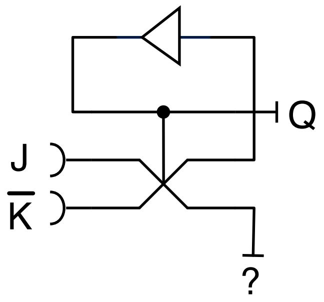


Figure II.24: Implementation of J-$\bar{\text{K}}$ flip-flop. [adapted from Fredkin & Toffoli (1982)]

Exercise II.10 Show that the inverse of the interaction gate (Fig. II.20) works correctly. Hint: It only needs to work correctly for outputs that actually occur. Therefore, to invert a pq output, balls must be shot into outputs A and D simultaneously.

Exercise II.11 Use interaction gates (and constant inputs and garbage outputs, as necessary) to implement NOT, OR (inclusive or), and XOR (exclusive or).

Exercise II.12 Show that the realization of the Fredkin gate in terms of interaction gates (Fig. II.23) is correct, by labeling the inputs and outputs of the interaction gates with Boolean expressions of a, b, and c.

Chapter III

Quantum Computation

A Mathematical preliminaries

"[I]nformation is physical, and surprising physical theories such as quantum mechanics may predict surprising information processing abilities." (Nielsen & Chuang, 2010, p. 98)

A.1 Complex numbers

If you go to the course webpage, and look under Quantum Computation in the Topics section, you will see a link to "complex number review [FFC-ch4]." Depending on how familiar you are with complex numbers, read or skim it through section 4.4.2.1 (pp. 41–53). This should tell you all you need to know (and a little more).

A.2 Linear algebra review

A.2.a Dirac bracket notation

Much of the math of quantum computation is just elementary linear algebra, but the notation is different (and of course there is a physical interpretation). The Dirac bracket notation will seem peculiar if you are not used to it, but it is elegant and powerful, as are all good notations. Think of it like a new programming language.

First, the notation $|\psi\rangle$ represents an n-dimensional vector, which we can write in a particular basis as a $n \times 1$ complex column vector, $|\psi\rangle = (v_1, \ldots, v_n)^{\mathrm{T}}$. We pronounce $|\psi\rangle$ "ket psi" or "psi ket." Normally the vectors are finite-dimensional, but they can be infinite-dimensional if the vectors have a finite magnitude (their components are square-summable), $\sum_k |v_k|^2 < \infty$.

The Dirac notation has the advantage that we can use arbitrary names for vectors, for example:

$$|\text{excited}\rangle,\ |\text{zero}\rangle,\ |\text{one}\rangle,\ |\uparrow\rangle,\ |\nearrow\rangle,\ |1\rangle,\ |101\rangle,\ |5\rangle,\ |f(\mathbf{x})\rangle,\ |1 \otimes g(1)\rangle.$$

It may be easier to remember if you notice that it looks kind of like an arrow; compare $|v\rangle$ and $\vec{v}$.

The notation $\langle\phi|$ represents a $1 \times n$ complex *row* vector, $\langle\phi| = (u_1, \ldots, u_n)$ in a particular basis. We pronounce $\langle\psi|$ "bra psi" or "psi bra." If $|\psi\rangle = (v_1, \ldots, v_n)^{\mathrm{T}}$, then $\langle\psi| = (\overline{v_1}, \ldots, \overline{v_n})$, where $\overline{v_k}$ is the complex conjugate of v_k. Recall that

$$\overline{x + iy} = x - iy \text{ and } \overline{re^{i\phi}} = re^{-i\phi}.$$

We define the *adjoint* (*conjugate transpose, Hermitian transpose*) $M^\dagger$ of a matrix M by

$$(M^\dagger)_{jk} = \overline{M_{kj}}.$$

We pronounce it "M dagger," "M adjoint," etc. Note that corresponding bras and kets are adjoints: $\langle\psi| = |\psi\rangle^\dagger$ and $|\psi\rangle = \langle\psi|^\dagger$.

A.2.b INNER PRODUCT

Suppose $|\phi\rangle = (u_1, \ldots, u_n)^{\mathrm{T}}$ and $|\psi\rangle = (v_1, \ldots, v_n)^{\mathrm{T}}$ in the same basis. Then the *complex inner product* of the vectors is defined $\sum_k \overline{u_k} v_k = \langle\phi|\,|\psi\rangle$. Thus, the inner product of two vectors is the conjugate transpose of the first times the second. This is the convention in physics, which we will follow; mathematicians usually put the complex conjugate on the second argument. It doesn't make any difference so long as you are consistent. Since the inner product multiplies a $1 \times n$ matrix by an $n \times 1$ matrix, the result is a 1×1 matrix, or scalar. This product of a bra and a ket is usually abbreviated $\langle\phi \mid \psi\rangle = \langle\phi|\,|\psi\rangle$, which can be pronounced "ϕ-bra ket-ψ" or "ϕ bra-ket ψ."

The complex inner product satisfies several important properties:

1. It is *positive definite*:

$$\begin{aligned}\langle\psi \mid \psi\rangle &> 0, \quad \text{if } |\psi\rangle \neq \mathbf{0},\\ \langle\psi \mid \psi\rangle &= 0, \quad \text{if } |\psi\rangle = \mathbf{0}.\end{aligned}$$

2. It has *conjugate symmetry*: $\langle\phi \mid \psi\rangle = \overline{\langle\psi \mid \phi\rangle}$.

3. It is *linear in its second argument* (the one that's not conjugated):

$$\begin{aligned}\langle\phi \mid c\psi\rangle &= c\langle\phi \mid \psi\rangle, \quad \text{for } c \in \mathbb{C},\\ \langle\phi \mid \psi + \chi\rangle &= \langle\phi \mid \psi\rangle + \langle\phi \mid \chi\rangle.\end{aligned}$$

Note that conjugate symmetry and linearity in the second argument together imply that $\langle c\phi \mid \psi\rangle = \overline{c}\langle\phi \mid \psi\rangle$ (antilinearity in the first argument). The complex inner product is called *sesquilinear*, which means "one-and-a-half linear" (in contrast to the inner product of real vectors, which is linear in both arguments, i.e., *bilinear*).

The *norm* or *magnitude* of a vector is defined $\||\psi\rangle\| = \sqrt{\langle\psi \mid \psi\rangle}$. That is, $\||\psi\rangle\|^2 = |v_1|^2 + \cdots |v_n|^2$. A vector is *normalized* if $\||\psi\rangle\| = 1$. Note that normalized vectors fall on the surface of an n-dimensional hypersphere.

A.2.c Bases and Generalized Fourier Series

Vectors $|\phi\rangle$ and $|\psi\rangle$ are *orthogonal* if $\langle\phi \mid \psi\rangle = 0$. A set of vectors is *orthogonal* if each vector is orthogonal to all the others. An *orthonormal (ON)* set of vectors is an orthogonal set of normalized vectors. A set of vectors $|\phi_1\rangle, |\phi_2\rangle, \ldots$ *spans* a vector space if for every vector $|\psi\rangle$ in the space there are complex coefficients $c_1, c_2, \ldots$ such that $|\psi\rangle = \sum_k c_k|\phi_k\rangle$. A *basis* for a vector space is a linearly independent set of vectors that spans the space. Equivalently, a basis is a *minimal generating set* for the space; that is, all of the vectors in the space can be generated by linear combinations of the basis vectors. An *(orthonormal) basis* for a vector space is an (orthonormal) set of vectors that spans the space. In general, when I write "basis" I mean "orthonormal basis." Any vector in the space has a unique representation as a linear combination of the basis vectors.

A *Hilbert space* is a complete inner-product space. "Complete" means that all Cauchy sequences of vectors (or functions) have a limit in the space. (In a Cauchy sequence, $\|x_m - x_n\| \to 0$ as $m, n \to \infty$.) Hilbert spaces may be finite- or infinite-dimensional. Suppose $|1\rangle$, $|2\rangle$, ... is an ON basis for a Hilbert space $\mathcal{H}$. Note, that these are just the names of the basis vectors (we could have used $|e_1\rangle$, $|e_2\rangle$, ... or something similar), and have nothing to do with the integers 1, 2, etc. Given such a basis, any $|\psi\rangle$ in $\mathcal{H}$ can be expanded in a *generalized Fourier series*:

$$|\psi\rangle = \sum_k c_k |k\rangle.$$

The *generalized Fourier coefficients* c_k can be determined by taking inner products with the corresponding basis vectors:

$$\langle k \mid \psi\rangle = \langle k| \sum_j c_j |j\rangle = \sum_j c_j \langle k \mid j\rangle = c_k.$$

Therefore, $c_k = \langle k \mid \psi\rangle$. Hence,

$$|\psi\rangle = \sum_k c_k|k\rangle = \sum_k \langle k \mid \psi\rangle\, |k\rangle = \sum_k |k\rangle\langle k \mid \psi\rangle.$$

This is just the vector's representation in a particular basis. (Note that this equation implies the identity matrix $I = \sum_k |k\rangle\langle k|$.)

A *linear operator* $L : \mathcal{H} \to \hat{\mathcal{H}}$ satisfies $L(c|\phi\rangle + d|\psi\rangle) = cL(|\phi\rangle) + dL(|\psi\rangle)$ for all $|\phi\rangle, |\psi\rangle \in \mathcal{H}$ and $c, d \in \mathbb{C}$. For example, differentiation is a linear operator.

A linear operator $L : \mathcal{H} \to \hat{\mathcal{H}}$ can be represented by a (possibly infinite-dimensional) matrix relative to bases for $\mathcal{H}$ and $\hat{\mathcal{H}}$. To see this, suppose $|1\rangle$, $|2\rangle$, ... is a basis for $\mathcal{H}$ and $|\hat{1}\rangle$, $|\hat{2}\rangle$, ... is a basis for $\hat{\mathcal{H}}$. Consider $|\phi\rangle = L|\psi\rangle$ and represent the vectors in these bases by their Fourier coefficients: $b_j = \langle \hat{\jmath} \mid \phi\rangle$ and $c_k = \langle k \mid \psi\rangle$. Hence $|\phi\rangle$ is represented by the vector $\mathbf{b} = (b_1, b_2, \ldots)^{\mathrm{T}}$ and $|\psi\rangle$ by the vector $\mathbf{c} = (c_1, c_2, \ldots)^{\mathrm{T}}$. Apply the linearity of L:

$$\begin{aligned}
b_j &= \langle \hat{\jmath} \mid \phi\rangle \\
&= \langle \hat{\jmath} \mid L \mid \psi\rangle \\
&= \langle \hat{\jmath}|L\left(\sum_k c_k|k\rangle\right) \\
&= \langle \hat{\jmath}|\left(\sum_k c_k L|k\rangle\right) \\
&= \sum_k \langle \hat{\jmath} \mid L \mid k\rangle c_k.
\end{aligned}$$

Therefore, define the matrix $M_{jk} \overset{\text{def}}{=} \langle \hat{\jmath} \mid L \mid k \rangle$ and we see $\mathbf{b} = \mathbf{Mc}$. For this reason, an expression of the form $\langle \hat{\jmath} \mid L \mid k \rangle$ is sometimes called a *matrix element* of the operator L. Note that the matrix depends on the bases we choose.

A.2.d Outer product or dyad

We can form the product of a ket and a bra, which is called a *dyad* or *outer product*. Consider first the finite dimensional case. If $|\phi\rangle$ is an $m \times 1$ column vector, and $|\psi\rangle$ is an $n \times 1$ column vector (so that $\langle\psi|$ is a $1 \times n$ row vector), then the outer product $|\phi\rangle\langle\psi|$ is an $m \times n$ matrix. In most cases of interest $m = n$. Since matrix multiplication is associative, $(|\phi\rangle\langle\psi|)\,|\chi\rangle = |\phi\rangle\,\langle\psi \mid \chi\rangle$. More generally, we can form outer products of infinite-dimensional vectors in Hilbert spaces. If $|\phi\rangle \in \mathcal{H}'$ and $|\psi\rangle \in \mathcal{H}$, then $|\phi\rangle\langle\psi|$ is the linear operator $L : \mathcal{H} \to \mathcal{H}'$ defined, for any $|\chi\rangle \in \mathcal{H}$:

$$L|\chi\rangle = (|\phi\rangle\langle\psi|)|\chi\rangle = |\phi\rangle\,\langle\psi \mid \chi\rangle.$$

That is, $|\phi\rangle\langle\psi|$ is the operator that returns $|\phi\rangle$ scaled by the inner product of $|\psi\rangle$ and its argument. To the extent that the inner product $\langle\psi \mid \chi\rangle$ measures the similarity of $|\psi\rangle$ and $|\chi\rangle$, the result $|\phi\rangle$ is weighted by this similarity. The product of a ket and a bra $|\phi\rangle\langle\psi|$ can be pronounced "ϕ-ket bra-ψ" or "ϕ ket-bra ψ," and abbreviated $|\phi\rangle\!\langle\psi|$. This product is also called a *dyad*.

The special case of $|\phi\rangle\!\langle\phi|$ in which $|\phi\rangle$ is normalized is called a *projector* onto $|\phi\rangle$. This is because $|\phi\rangle\!\langle\phi|\;|\psi\rangle = |\phi\rangle\,\langle\phi \mid \psi\rangle$, that is, $|\phi\rangle$ scaled by the projection of $|\psi\rangle$ on $|\phi\rangle$. More generally, if $|\eta_1\rangle, \ldots, |\eta_m\rangle$ are orthonormal, then $\sum_{k=1}^{m} |\eta_k\rangle\!\langle\eta_k|$ projects into the m-dimensional subspace spanned by these vectors.

Any linear operator can be represented as a weighted sum of outer products. To see this, suppose $L : \mathcal{H} \to \hat{\mathcal{H}}$, $|\hat{\jmath}\rangle$ is a basis for $\hat{\mathcal{H}}$, and $|k\rangle$ is a basis for $\mathcal{H}$. Consider $|\phi\rangle = L|\psi\rangle$. We know from Sec. A.2.c that

$$\langle \hat{\jmath} \mid \phi \rangle = \sum_k M_{jk} c_k, \text{ where } M_{jk} = \langle \hat{\jmath} \mid L \mid k \rangle, \text{ and } c_k = \langle k \mid \psi \rangle.$$

Hence,

$$|\phi\rangle \;=\; \sum_j |\hat{\jmath}\rangle\,\langle \hat{\jmath} \mid \phi \rangle$$

$$
\begin{aligned}
&= \sum_j |\hat{\jmath}\rangle \left(\sum_k M_{jk} \langle k \mid \psi \rangle \right) \\
&= \left(\sum_j |\hat{\jmath}\rangle \sum_k M_{jk} \langle k| \right) |\psi\rangle \\
&= \left(\sum_{jk} M_{jk} |\hat{\jmath}\rangle \langle k| \right) |\psi\rangle.
\end{aligned}
$$

Hence, we have a sum-of-outer-products representation of the operator in terms of its matrix elements:

$$L = \sum_{jk} M_{jk} |\hat{\jmath}\rangle\langle k|, \text{ where } M_{jk} = \langle \hat{\jmath} \mid L \mid k \rangle.$$

A.2.e Tensor products

In this section we define the *tensor product*, which is of central importance in quantum computation. To see where we are headed, suppose $|\phi\rangle$ and $|\psi\rangle$ are the quantum states of two objects (e.g., the states of two *qubits*, or quantum bits). Then the state of the composite system (e.g., the pair of qubits) will be represented by the tensor product $|\phi\rangle \otimes |\psi\rangle$, which we can think of as a sort of concatenation or structure formed of $|\phi\rangle$ and $|\psi\rangle$. If $\mathcal{H}$ and $\mathcal{H}'$ are the Hilbert spaces from which these states are drawn, then the *tensor product space* $\mathcal{H} \otimes \mathcal{H}'$ is the space of all possible states of the two-qubit system (i.e., all such pairs or structures). We can apply the tensor product to operators as well: $L \otimes M$ is the operator that applies, in parallel, L to the first qubit of the pair and M to the second qubit:

$$(L \otimes M)(|\phi\rangle \otimes |\psi\rangle) = (L|\phi\rangle) \otimes (M|\psi\rangle).$$

For vectors, operators, and spaces, we pronounce $L \otimes M$ as "L tensor M." As we will see, the tensor product is essential to much of the power of quantum computation. Next we develop these ideas more formally.

Suppose that $|\eta_j\rangle$ is an ON basis for $\mathcal{H}$ and $|\eta'_k\rangle$ is an ON basis for $\mathcal{H}'$. For every pair of basis vectors, define the *tensor product* $|\eta_j\rangle \otimes |\eta'_k\rangle$ as a sort of couple or pair of the two basis vectors; that is, there is a one-to-one correspondence between the $|\eta_j\rangle \otimes |\eta'_k\rangle$ and the pairs in $\{|\eta_0\rangle, |\eta_1\rangle, \ldots\} \times \{|\eta'_0\rangle, |\eta'_1\rangle, \ldots\}$. Define the *tensor product space* $\mathcal{H} \otimes \mathcal{H}'$ as the space spanned by all linear combinations of the basis vectors $|\eta_j\rangle \otimes |\eta'_k\rangle$. Therefore each element of $\mathcal{H} \otimes \mathcal{H}'$ is represented by a unique sum $\sum_{jk} c_{jk} |\eta_j\rangle \otimes |\eta'_k\rangle$. In order to make $\mathcal{H} \otimes \mathcal{H}'$ a Hilbert space, we need an inner product:

$$\langle \phi_1 \otimes \phi_2 \mid \psi_1 \otimes \psi_2 \rangle = \langle \phi_1 \mid \psi_1 \rangle \, \langle \phi_2 \mid \psi_2 \rangle.$$

That is, we multiply the inner products of the corresponding elements of the tensor product pairs.

Usually, we are dealing with finite-dimensional spaces, in which case the tensor products can be defined less abstractly in terms of matrices. Suppose in a given basis $|\phi\rangle = (u_1, \ldots, u_m)^{\mathrm{T}}$ and $|\psi\rangle = (v_1, \ldots, v_n)^{\mathrm{T}}$, then their tensor product in that basis can be defined by the *Kronecker product*:

$$\begin{aligned} |\phi\rangle \otimes |\psi\rangle &= \begin{pmatrix} u_1|\psi\rangle \\ \vdots \\ u_m|\psi\rangle \end{pmatrix} \\ &= \left(u_1|\psi\rangle^{\mathrm{T}}, \ldots, u_m|\psi\rangle^{\mathrm{T}}\right)^{\mathrm{T}} \\ &= (u_1v_1, \ldots, u_1v_n, \ldots, u_mv_1 \ldots, u_mv_n)^{\mathrm{T}}. \end{aligned}$$

Note that this is an $mn \times 1$ column vector and that

$$(|\phi\rangle \otimes |\psi\rangle)_{(j-1)n+k} = u_j v_k.$$

This combinatorial explosion of dimension is what gives quantum computation its power.

The following abbreviations are frequent: $|\phi\psi\rangle = |\phi, \psi\rangle = |\phi\rangle|\psi\rangle = |\phi\rangle \otimes |\psi\rangle$. Note that $|\phi\rangle|\psi\rangle$ can only be a tensor product because it would not be a legal matrix product. These are some useful properties of the tensor product (which is bilinear):

$$\begin{aligned} (c|\phi\rangle) \otimes |\psi\rangle &= c(|\phi\rangle \otimes |\psi\rangle) = |\phi\rangle \otimes (c|\psi\rangle), \\ (|\phi\rangle + |\psi\rangle) \otimes |\chi\rangle &= (|\phi\rangle|\chi\rangle) + (|\psi\rangle|\chi\rangle), \\ |\phi\rangle \otimes (|\psi\rangle + |\chi\rangle) &= (|\phi\rangle \otimes |\psi\rangle) + (|\phi\rangle \otimes |\chi\rangle). \end{aligned}$$

The tensor product of linear operators is defined

$$(L \otimes M)\,(|\phi\rangle \otimes |\psi\rangle) = L|\phi\rangle \otimes M|\psi\rangle.$$

Using the fact that $|\psi\rangle = \sum_{jk} c_{jk}|\eta_j\rangle \otimes |\eta'_k\rangle$ you can compute $(L \otimes M)|\psi\rangle$ for an arbitrary $|\psi\rangle \in \mathcal{H} \otimes \mathcal{H}'$ (exercise). If $\mathbf{M}$ is a $k \times m$ matrix and $\mathbf{N}$ is a $l \times n$ matrix, then their Kronecker product is a $kl \times mn$ matrix:

$$\mathbf{M} \otimes \mathbf{N} = \begin{pmatrix} M_{11}\mathbf{N} & M_{12}\mathbf{N} & \cdots & M_{1m}\mathbf{N} \\ M_{21}\mathbf{N} & M_{22}\mathbf{N} & \cdots & M_{2m}\mathbf{N} \\ \vdots & \vdots & \ddots & \vdots \\ M_{k1}\mathbf{N} & M_{k2}\mathbf{N} & \cdots & M_{km}\mathbf{N} \end{pmatrix}.$$

Matrices, of course, are linear operators and satisfy $(\mathbf{M} \otimes \mathbf{N})\,(|\phi\rangle \otimes |\psi\rangle) = \mathbf{M}|\phi\rangle \otimes \mathbf{N}|\psi\rangle$.

For a vector, operator, or space M, we define the *tensor power* $M^{\otimes n}$ to be M tensored with itself n times:

$$M^{\otimes n} = \overbrace{M \otimes M \otimes \cdots \otimes M}^{n}.$$

A.2.f Properies of operators and matrices

Several properties of operators and matrices are important in quantum computation. An operator $L : \mathcal{H} \to \mathcal{H}$ is *normal* if $L^\dagger L = LL^\dagger$. The same applies to square matrices. That is, normal operators commute with their adjoints.

An operator $L : \mathcal{H} \to \mathcal{H}$ is *Hermitian* or *self-adjoint* if $L^\dagger = L$. The same applies to square matrices. (Hermitian matrices are the complex analogues of symmetric matrices.) Note that Hermitian operators are normal. It is easy to see that L is Hermitian if and only if $\langle \phi \mid L \mid \psi\rangle = \langle \psi \mid L \mid \phi\rangle$ for all $|\phi\rangle, |\psi\rangle$ (since $\langle \psi \mid L \mid \phi\rangle = \langle \phi \mid L^\dagger \mid \psi\rangle = \langle \phi \mid L \mid \psi\rangle$). A normal matrix is Hermitian if and only if it has real eigenvalues (exercise). This is important in quantum mechanics, since measurement results are usually assumed to be real.

An operator U is *unitary* if $U^\dagger U = UU^\dagger = I$. That is, a unitary operator is invertible and its inverse is its adjoint: if U is unitary, then $U^{-1} = U^\dagger$. Obviously every unitary operator is also normal. (A normal matrix is unitary if and only if its *spectrum* — the multiset of its eigenvalues — is contained in the unit circle in the complex plane.)

Unitary operators are like rotations of a complex vector space (analogous to orthogonal operators, which are rotations of a real vector space). Just as orthogonal transformations preserve the angles between vectors, unitary operators preserve their inner products (which are analogous to angles).

Consider the inner product between $U|\phi\rangle$ and $U|\psi\rangle$:

$$(U|\phi\rangle)^\dagger U|\psi\rangle = \langle\phi \mid U^\dagger U \mid \psi\rangle = \langle\phi \mid \psi\rangle.$$

Hence, the inner product of $U|\phi\rangle$ and $U|\psi\rangle$ is the same as the inner product of $|\phi\rangle$ and $|\psi\rangle$. Therefore, unitary operators are *isometric*, that is, they preserve norms:

$$\|U|\psi\rangle\|^2 = \langle\psi \mid U^\dagger U \mid \psi\rangle = \langle\psi \mid \psi\rangle = \||\psi\rangle\|^2.$$

Unitary operators are important in quantum computation because the evolution of quantum systems is unitary.

A.2.g Spectral decomposition and operator functions (supplementary)

For any normal operator on a finite-dimensional Hilbert space, there is an ON basis that diagonalizes the operator, and conversely, any diagonalizable operator is normal. This is called a *spectral decomposition* of the operator. The ON basis comprises its set of eigenvectors, which we can write $|0\rangle, |1\rangle, \ldots, |n\rangle$ and the corresponding eigenvalues λ_k are the diagonal elements (cf. Sec. A.2.d, p. 72): $L = \sum_{k=1}^n \lambda_k |k\rangle\langle k|$. Therefore, a matrix is normal if and only if it can be diagonalized by a unitary transform (see A.2.f, above). That is, it is normal if and only if there is a unitary U such that $L = U\Lambda U^\dagger$, where $\Lambda = \operatorname{diag}(\lambda_1, \ldots \lambda_n)$. If $|0\rangle, |1\rangle, \ldots, |n\rangle$ is the basis, then $U = (|0\rangle, |1\rangle, \ldots, |n\rangle)$ and

$$U^\dagger = \begin{pmatrix} \langle 0| \\ \langle 1| \\ \vdots \\ \langle n| \end{pmatrix}.$$

(More generally, this property holds for compact normal operators.)

It is often convenient to extend various complex functions (e.g., ln, exp, $\sqrt{}$) to normal matrices and operators. If $f : \mathbb{C} \rightarrow \mathbb{C}$ and $L : \mathcal{H} \rightarrow \mathcal{H}$, then we define:

$$f(L) \stackrel{\text{def}}{=} \sum_{k=1}^n f(\lambda_k)|k\rangle\langle k|,$$

where $L = \sum_{k=1}^{n} \lambda_k |k\rangle\langle k|$ is the spectral decomposition of L. Therefore, for a normal linear operator or matrix L we can write $\sqrt{L}$, $\ln L$, e^L, etc. This is not an arbitrary extension, but follows from the power series expansion for f.

B Basic concepts from quantum theory

B.1 Introduction

B.1.a Bases

In quantum mechanics certain physical quantities are quantized, such as the energy of an electron in an atom. Therefore an atom might be in certain distinct energy states $|\text{ground}\rangle$, $|\text{first excited}\rangle$, $|\text{second excited}\rangle$, $\ldots$. Other particles might have distinct states such as spin-up $|\uparrow\rangle$ and spin-down $|\downarrow\rangle$. In each case these alternative states correspond to orthonormal vectors:

$$\begin{aligned}&\langle\uparrow\mid\downarrow\rangle = 0,\\ &\langle\text{ground} \mid \text{first excited}\rangle = 0,\\ &\langle\text{ground} \mid \text{second excited}\rangle = 0,\\ &\langle\text{first excited} \mid \text{second excited}\rangle = 0.\end{aligned}$$

In general we may express the same state with respect to different bases, such as vertical or horizontal polarization $|\rightarrow\rangle$, $|\uparrow\rangle$; or orthogonal diagonal polarizations $|\nearrow\rangle$, $|\searrow\rangle$.

B.1.b Superpositions of Basis States

One of the unique characteristics of quantum mechanics is that a physical system can be in a superposition of basis states, for example,

$$|\psi\rangle = c_0|\text{ground}\rangle + c_1|\text{first excited}\rangle + c_2|\text{second excited}\rangle,$$

where the c_j are complex numbers, called *(probability) amplitudes.* With respect to a given basis, a state $|\psi\rangle$ is interchangeable with its vector of coefficients, $\mathbf{c} = (c_0, c_1, \ldots, c_n)^{\mathrm{T}}$. When the basis is understood, we can use $|\psi\rangle$ as a name for this vector. This ability of a quantum system to be in many states simultaneously is the foundation of *quantum parallelism.*

As we will see, when we measure the quantum state

$$c_0|E_0\rangle + c_1|E_1\rangle + \ldots + c_n|E_n\rangle$$

with respect to the $|E_0\rangle, \ldots, |E_n\rangle$ basis, we will get the result $|E_j\rangle$ with probability $|c_j|^2$ and the state will "collapse" into state $|E_j\rangle$. Since the probabilities must add to 1, $|c_0|^2 + |c_1|^2 + \cdots + |c_n|^2 = 1$, we know $\| |\psi\rangle \| = 1$, that is, the vector is normalized.

For the purposes of quantum computation, we usually pick two basis states and use them to represent the bits 1 and 0, for example, $|1\rangle = |\text{ground}\rangle$ and $|0\rangle = |\text{excited}\rangle$. We call this the *computational basis*. I've picked the opposite of the "obvious" assignment ($|0\rangle = |\text{ground}\rangle$) just to show that the assignment is arbitrary (just as for classical bits). Note that $|0\rangle \neq \mathbf{0}$, the zero element of the vector space, since $\| |0\rangle \| = 1$ but $\|\mathbf{0}\| = 0$. (Thus $\mathbf{0}$ does not represent a physical state, since it is not normalized.)[1]

B.2 Postulates of QM

In this section you will learn the four fundamental postulates of quantum mechanics.[2]

B.2.a Postulate 1: state space

Postulate 1: Associated with any isolated physical system is a *state space*, which is a Hilbert space. The state of the system "is completely defined by its *state vector*, which is a unit vector in the system's state space" (Nielsen & Chuang, 2010). The state vector has to be normalized so that the total probability is 1; it is equivalent to the probability axiom that states that the maximum probability (probability of the whole sample space) = 1.

In previous examples, the state vectors have been finite dimensional, but Hilbert spaces can be infinite dimensional as well. For example, a quantum system might have an unlimited number of energy levels, $|0\rangle, |1\rangle, |2\rangle, \ldots$. If the state of the system is a superposition, $|\psi\rangle = \sum_{k=0}^{\infty} c_k |k\rangle$, then the squared amplitudes must sum to 1, $\sum_{k=0}^{\infty} |c_k|^2 = 1$.

A quantum state $|\psi\rangle$ is often a *wavefunction*, which defines the *probability amplitude* distribution (actually, the probability density function) of some continuous quantity. For example, $|\psi\rangle$ may define the complex amplitude $\psi(\mathbf{r})$ associated with each location $\mathbf{r}$ in space, and $|\Psi\rangle$ may define the complex amplitude of $\Psi(\mathbf{p})$ associated with each momentum $\mathbf{p}$ (see Fig. III.1). Infinite dimensional Hilbert spaces also include spaces of wavefunctions such as these.

[1]Physical systems with at least three distinct quantum states (e.g., $|0\rangle$, $|1\rangle$, $|2\rangle$) can be used to implement three-dimensional quantum data (called *qutrits*), which have some advantages. Most quantum computing to date, however, has used two-dimensional *qubits* rather than these higher-dimensional representations.

[2]Quotes are from Nielsen & Chuang (2010) unless otherwise specified.

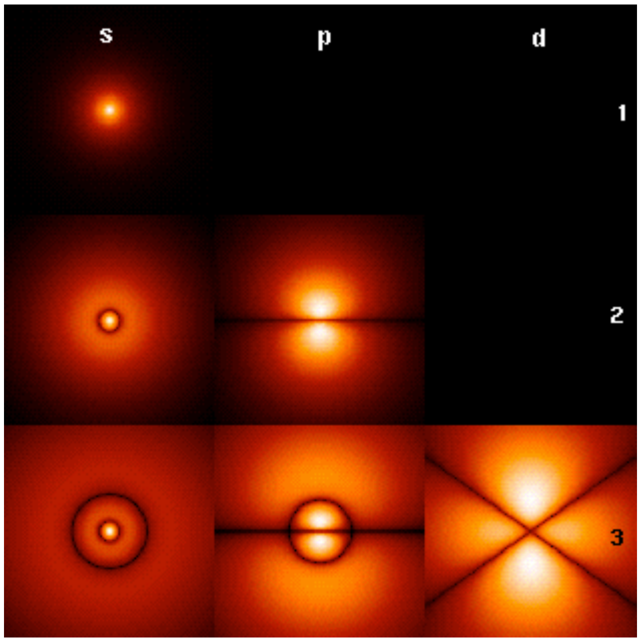


Figure III.1: Probability density of first six hydrogen orbitals. The main quantum number ($n = 1, 2, 3$) and the angular momentum quantum number ($\ell = 0, 1, 2 =$ s, p, d) are shown. (The magnetic quantum number $m = 0$ in these plots.) [fig. from wikipedia commons]

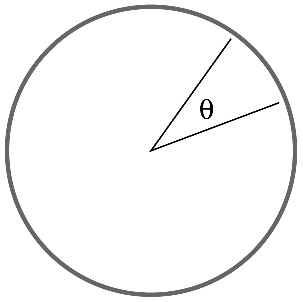


Figure III.2: Relative phase vs. global phase. What matters in quantum mechanics is the relative phase between state vectors (e.g., θ in the figure). Global phase "has no physical meaning"; i.e., we can choose to put the 0° point anywhere we like.

The inner product of wavefunctions is defined:

$$\langle \phi \mid \psi \rangle = \int_{\mathbb{R}^3} \overline{\phi(\mathbf{r})} \psi(\mathbf{r}) \mathrm{d}\mathbf{r}.$$

(For this example we are assuming the domain is 3D space.) Wavefunctions are also normalized, $1 = \| |\psi\rangle \|^2 = \int_{\mathbb{R}^3} |\psi(\mathbf{r})|^2 \mathrm{d}\mathbf{r}$. For our purposes, finite dimensional spaces are usually adequate.

In quantum mechanics, global phase has no physical meaning; all that matters is relative phase. In other words, if you consider all the angles around the circle, there is no distinguished 0° (see Fig. III.2). Likewise, in a continuous wave (such as a sine wave), there is no distinguished starting point (see Fig. III.3).

To say all quantum states are normalized is equivalent to saying that their absolute length has no physical meaning. That is, only their *form* (shape) matters, not their absolute size. This is a characteristic of *information.*

Another way of looking at quantum states is as *rays* in a *projective Hilbert space.* A *ray* is an equivalence class of nonzero vectors under the relation, $\phi \cong \psi$ iff $\exists z \neq 0 \in \mathbb{C} : \phi = z\psi$, where $\phi, \psi \neq \mathbf{0}$. That is, global magnitude and phase (r and ϕ in $z = re^{i\phi}$) are irrelevant (i.e., have no physical meaning). This is another way of expressing the fact that the *form* is significant, but not the *size.* However, it is more convenient to use normalized vectors in ordinary Hilbert spaces and to ignore global phase.

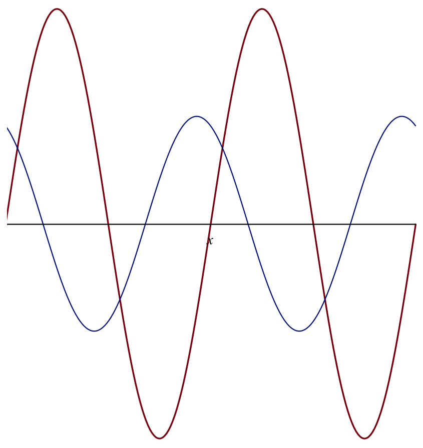


Figure III.3: Relative phase vs. global phase of sine waves. There is no privileged point from which to start measuring absolute phase, but there is a definite relative phase between the two waves.

B.2.b Postulate 2: evolution

Postulate 2: "The evolution of a closed quantum system is described by a unitary transformation" (Nielsen & Chuang, 2010). Therefore a closed quantum system evolves by "complex rotation" of a Hilbert space. More precisely, the state $|\psi\rangle$ of the system at time t is related to the state $|\psi'\rangle$ of the system at time t' by a unitary operator U which depends only on the times t and t',

$$|\psi'\rangle = U(t,t')|\psi\rangle = U|\psi\rangle.$$

This postulate describes the evolution of systems that don't interact with the rest of the world. That is, the quantum system is a dynamical system of relatively low dimension, whereas the environment, including any measurement apparatus, is a thermodynamical system (recall Ch. II, Sec. B).

Dynamics (supplementary) The laws of quantum mechanics, like the laws of classical mechanics, are expressed in differential equations. However, in quantum computation we usually deal with quantum gates operating in discrete time, so it is worth mentioning their relation.

The continuous-time evolution of a closed quantum mechanical system is given by the Schrödinger equation:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t} |\psi(t)\rangle = H|\psi(t)\rangle,$$

or more compactly, $i\hbar|\dot{\psi}\rangle = H|\psi\rangle$. H is the Hamiltonian of the system (a fixed Hermitian operator), and $\hbar$ is the reduced Planck constant (often absorbed into H).

Since H is Hermitian, it has a spectral decomposition, $H = \sum_E E|E\rangle\langle E|$, where the normalized $|E\rangle$ are *energy eigenstates* (or *stationary states*) with corresponding energies E. The lowest energy is the *ground state energy.*

In quantum computing, we are generally interested in the discrete-time dynamics of quantum systems. Stone's theorem shows that the solution to the Schrödinger equation is:

$$|\psi(t+s)\rangle = e^{-iHt/\hbar}|\psi(s)\rangle.$$

Therefore define $U(t) \overset{\text{def}}{=} \exp(-iHt/\hbar)$; then $|\psi(t+s)\rangle = U(t)|\psi(s)\rangle$. It turns out that U is unitary (Exer. III.6). Hence the evolution of a closed quantum mechanical system from a state $|\psi\rangle$ at time t to a state $|\psi'\rangle$ at time t' can be described by a unitary operator, $|\psi'\rangle = U|\psi\rangle$. Conversely, for any unitary operator U there is a Hermitian K such that $U = \exp(iK)$ (Exer. III.7).

B.2.c Postulate 3: quantum measurement

What happens if the system is no longer closed, that is, if it interacts with the larger environment? In particular, what happens if a quantum system interacts with a much larger measurement apparatus, the purpose of which is to translate a microscopic state into a macroscopic, observable effect? For example, suppose we have a quantum system that can be in two distinct states, for example, an atom that can be in a ground state $|0\rangle$ and an excited state $|1\rangle$. Since they are distinct states, they correspond to orthogonal vectors, $\langle 0 \mid 1\rangle = 0$. Suppose further that we have a measurement apparatus that turns on one light if it measures state $|0\rangle$ and a different light if it measures state $|1\rangle$.

Now consider an atom in a quantum state $|\psi\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$, a superposition of the states $|0\rangle$ and $|1\rangle$. When we measure $|\psi\rangle$ in the computational basis, we will measure $|0\rangle$ with probability $\left|\frac{1}{2}\right|^2 = \frac{1}{4}$, and we will measure $|1\rangle$ with probability $\left|\frac{\sqrt{3}}{2}\right|^2 = \frac{3}{4}$. After measurement, the system is in the state we measured ($|0\rangle$ or $|1\rangle$, respectively); this is the "collapse" of the wavefunction. We depict the possibilities as follows:

$$\begin{aligned} |\psi\rangle &\xrightarrow{1/4} |0\rangle, \\ |\psi\rangle &\xrightarrow{3/4} |1\rangle. \end{aligned}$$

Now consider a more complicated example, a quantum system that can be in three distinct states, say an atom that can be in a ground state $|0\rangle$ or two excited states, $|1\rangle$ and $|2\rangle$. Note that $\langle 0 \mid 1\rangle = \langle 1 \mid 2\rangle = \langle 0 \mid 2\rangle = 0$. Suppose the quantum system is in state $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle$. Further, suppose we have a measurement apparatus that turns on a light if it measures state $|0\rangle$ and does not turn it on otherwise. When we measure $|\psi\rangle$, with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$ we will measure $|0\rangle$ and after measurement it will collapse to state $|0\rangle$. With probability $\left|\frac{1}{2}\right|^2 + \left|\frac{1}{2}\right|^2 = \frac{1}{2}$ it will not measure state $|0\rangle$ and the light won't go on. In this case, it will collapse to state $\frac{1}{\sqrt{2}}|1\rangle + \frac{1}{\sqrt{2}}|2\rangle$, which we get by renormalizing the state measured:

$$\frac{\frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle}{\sqrt{\left|\frac{1}{2}\right|^2 + \left|\frac{1}{2}\right|^2}} = \frac{1}{\sqrt{2}}|1\rangle + \frac{1}{\sqrt{2}}|2\rangle.$$

We can depict the possible outcomes as follows:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle \quad \left\{ \begin{array}{ll} \xrightarrow{1/2} & |0\rangle \\ \xrightarrow{1/2} & \frac{1}{\sqrt{2}}|1\rangle + \frac{1}{\sqrt{2}}|2\rangle \end{array} \right. .$$

In other words, we zero out the coefficients of the states we didn't measure and renormalize (because quantum states are always normalized). Now we develop these ideas more formally.

A measurement can be characterized by a set of projectors P_m, for each possible measurement outcome m. In the first example above, the measurement operators are $P_1 = |0\rangle\langle 0|$ and $P_2 = |1\rangle\langle 1|$. In the second example, the operators are $P_1 = |0\rangle\langle 0|$ and $P_2 = |1\rangle\langle 1| + |2\rangle\langle 2|$. In the latter case, P_1 projects the quantum state into the subspace spanned by $\{|0\rangle\}$, and P_2 projects the quantum state into the subspace spanned by $\{|1\rangle, |2\rangle\}$. These are *orthogonal subspaces* of the original space (spanned by $\{|0\rangle, |1\rangle, |2\rangle\}$).

Since a measurement must measure some definite state, a projective measurement is a set of projectors $P_1, \ldots, P_N$ satisfying: (1) They project into orthogonal subspaces, so for $m \neq n$ we have $P_m P_n = \mathbf{0}$, the identically zero operator. (2) They are complete, that is, $I = \sum_{m=1}^{N} P_m$, so measurement always produces a result. Projectors are also idempotent, $P_m P_m = P_m$, since if a vector is already projected into the m subspace, projecting it again has no effect. Finally, projectors are Hermitian (self-adjoint), as we can see:

$$P_m^\dagger = \left(\sum_j |\eta_j\rangle\langle\eta_j| \right)^\dagger = \sum_j (|\eta_j\rangle\langle\eta_j|)^\dagger = \sum_j |\eta_j\rangle\langle\eta_j| = P_m.$$

Now we can state Postulate 3.

Postulate 3: Quantum measurements are described by a complete set of orthogonal *projectors*, P_m, for each possible measurement outcome m.

Measurement projects the state into a subspace with a probability given by the squared magnitude of the projection. Therefore, the probability of measurement m of state $|\psi\rangle$ is given by:

$$p(m) = \| P_m |\psi\rangle \|^2 = \langle \psi \mid P_m^\dagger P_m \mid \psi \rangle = \langle \psi \mid P_m P_m \mid \psi \rangle = \langle \psi \mid P_m \mid \psi \rangle. \tag{III.1}$$

This is *Born's Rule*, which gives the probability of a measurement outcome. The measurement probabilities must sum to 1, which we can check:

$$\sum_m p(m) = \sum_m \langle \psi \mid P_m \mid \psi \rangle = \langle \psi | \left(\sum_m P_m \right) |\psi\rangle = \langle \psi \mid I \mid \psi \rangle = \langle \psi \mid \psi \rangle = 1.$$

This follows from the completeness if the projectors, $\sum_m P_m = I$.

For an example, suppose $P_m = |m\rangle\langle m|$, and write the quantum state in the measurement basis: $|\psi\rangle = \sum_m c_m |m\rangle$. Then the probability $p(m)$ of measuring m is:

$$\begin{aligned} p(m) &= \langle \psi \mid P_m \mid \psi \rangle \\ &= \langle \psi | (|m\rangle\langle m|) | \psi \rangle \\ &= \langle \psi \mid m \rangle \langle m \mid \psi \rangle \\ &= \overline{\langle m \mid \psi \rangle} \langle m \mid \psi \rangle \\ &= |\langle m \mid \psi \rangle|^2 \\ &= |c_m|^2. \end{aligned}$$

More generally, the same holds if P_m projects into a subspace, $P_m = \sum_k |k\rangle\langle k|$; the probability is $p(m) = \sum_k |c_k|^2$. Alternatively, we can "zero out" the c_j for the orthogonal subspace, that is, for the $|j\rangle\langle j|$ omitted by P_m. To maintain a total probability of 1, the normalized state vector after measurement is

$$\frac{P_m|\psi\rangle}{\sqrt{p(m)}} = \frac{P_m|\psi\rangle}{\|P_m|\psi\rangle\|}.$$

B.2.d Postulate 4: composite systems

Postulate 4: "The state space of a composite physical system is the tensor product of the state spaces of the component physical systems" (Nielsen & Chuang, 2010). If there are n subsystems, and subsystem j is prepared in state $|\psi_j\rangle$, then the composite system is in state

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle = \bigotimes_{j=1}^{n} |\psi_j\rangle.$$

B.3 Wave-particle duality (supplementary)

Some of the capabilities of quantum computation depend on the fact that microscopic objects behave as both particles and waves. To see why, imagine performing the double-slit experiment with three different kinds of objects.

Imagine a stream of classical particles impinging on the two slits and consider the probability of their arriving on a screen. Define $P_j(x)$ to be the probability of a particle arriving at x with just slit j open, and $P_{12}(x)$ to be the probability of a particle arriving at x with both open. We observe $P_{12} = P_1 + P_2$, as expected.

Now consider classical waves, such as water waves, passing through the two slits. The energy I of a water wave depends on the square of its height H, which may be positive or negative. Hence,

$$I_{12} = H_{12}^2 = (H_1 + H_2)^2 = H_1^2 + 2H_1H_2 + H_2^2 = I_1 + 2H_1H_2 + I_2.$$

The $2H_1H_2$ term may be positive or negative, which leads to constructive and destructive interference.

Finally, consider quantum particles. The probability of observing a particle is given by the rule for waves. In particular, the probability P is given by the square of a complex amplitude A:

$$\begin{aligned} P_{12} &= |A_1 + A_2|^2 = \overline{A_1}A_1 + \overline{A_1}A_2 + \overline{A_2}A_1 + \overline{A_2}A_2, \\ &= P_1 + \overline{A_1}A_2 + A_1\overline{A_2} + P_2. \end{aligned}$$

Again, the interference terms $\overline{A_1}A_2 + A_1\overline{A_2}$ can be positive or negative leading to constructive and destructive interference. How does a particle going through one slit "know" whether or not the other slit is open?

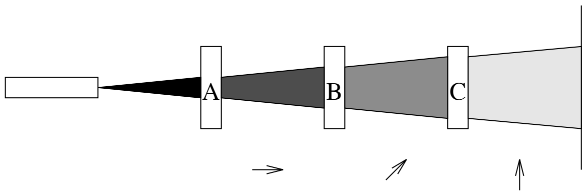


Figure III.4: Fig. from Rieffel & Polak (2000).

B.4 Superposition

A simple experiment demonstrates quantum effects that can not be explained by classical physics (see Fig. III.4). Suppose we have three polarizing filters, A, B, and C, polarized horizontally, 45°, and vertically, respectively. Place the horizontal filter A between a strong light source, such as a laser, and a screen. The light intensity is reduced by one half and the light is horizontally polarized. (Note: Since the light source is unpolarized, i.e., it has all polarizations, the resulting intensity would be much less than one half if the filter allowed only exactly horizontally polarized light through, as would be implied by a sieve model of polarization.) Next insert filter C, polarized vertically, and the intensity drops to zero. This is not surprising, since the filters are cross-polarized. Finally, insert filter B, polarized diagonally, between A and C, and surprisingly some light (about 1/8 intensity) will return! This can't be explained by the sieve model. How can putting in more filters increase the light intensity?

Quantum mechanics provides a simple explanation of the this effect; in fact, it's exactly what we should expect. A photon's polarization state can be represented by a unit vector pointing in appropriate direction. Therefore, arbitrary polarization can be expressed by $a|0\rangle + b|1\rangle$ for any two basis vectors $|0\rangle$ and $|1\rangle$, where $|a|^2 + |b|^2 = 1$.

A polarizing filter measures a state with respect to a basis that includes a vector parallel to its polarization and one orthogonal to it. The effect of filter A is the projector $P_{\text{A}} = |{\rightarrow}\rangle\langle{\rightarrow}|$. To get the probability amplitude, apply it to $|\psi\rangle \overset{\text{def}}{=} a|{\rightarrow}\rangle + b|\uparrow\rangle$:

$$p(\text{A}) = |\,\langle{\rightarrow}\,|\psi\rangle\,|^2 = |\langle{\rightarrow}\,|(a|{\rightarrow}\rangle + b|{\uparrow}\rangle)|^2 = |a\langle{\rightarrow}|{\rightarrow}\rangle + b\langle{\rightarrow}|{\uparrow}\rangle|^2 = |a|^2.$$

So with probability $|a|^2$ we get $|{\rightarrow}\rangle$ (recall Eqn. III.1, p. 84). So if the polarizations are randomly distributed from the source, half will get through

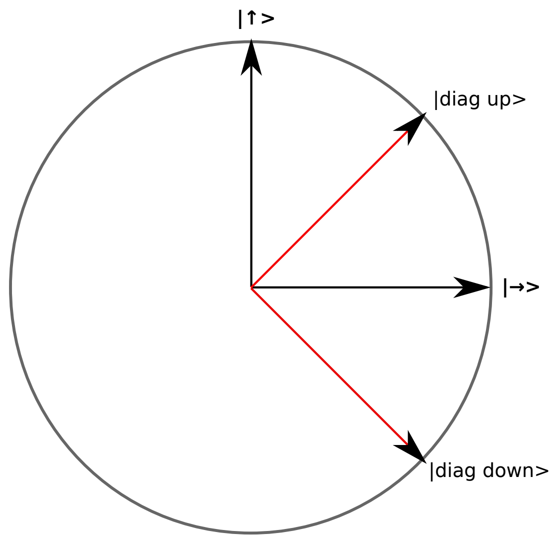


Figure III.5: Alternative polarization bases for measuring photons (black = rectilinear basis, red = diagonal basis). Note $|\nearrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\rightarrow\rangle)$ and $|\rightarrow\rangle = \frac{1}{\sqrt{2}}(|\nearrow\rangle + |\searrow\rangle)$.

---

and all of them will be in state $|\rightarrow\rangle$. Why one half? Note that $a = \cos\theta$, where θ is the angle between $|\psi\rangle$ and $|\rightarrow\rangle$, and that

$$\langle a^2\rangle = \frac{1}{2\pi}\int_0^{2\pi} \cos^2\theta \, \mathrm{d}\theta = \frac{1}{2}.$$

When we insert filter C we are measuring with the projector $P_{\mathrm{C}} = |\uparrow\rangle\langle\uparrow|$ and the result is 0, as expected:

$$p(\mathrm{AC}) = |\langle\uparrow|\rightarrow\rangle|^2 = 0.$$

Now insert the diagonal filter B between the horizontal and vertical filters A and C. Filter B measures with respect to the projector $\{|\nearrow\rangle, |\searrow\rangle\}$ basis (see Fig. III.5). Transmitted light is given by the projector $P_{\mathrm{B}} = |\nearrow\rangle\langle\nearrow|$. To find the result of applying filter B to the horizontally polarized light emerging from filter A, we must express $|\rightarrow\rangle$ in the diagonal basis:

$$|\rightarrow\rangle = \frac{1}{\sqrt{2}}(|\nearrow\rangle + |\searrow\rangle).$$

So if filter B is $|\nearrow\rangle\langle\nearrow|$ we get $|\nearrow\rangle$ photons passing through filter B with probability 1/2:

$$p(\mathrm{B}) = |\langle\nearrow|\rightarrow\rangle|^2 = \left|\langle\nearrow|\left[\frac{1}{\sqrt{2}}(|\nearrow\rangle + |\searrow\rangle)\right]\right|^2 = \left|\frac{1}{\sqrt{2}}\langle\nearrow|\nearrow\rangle + \frac{1}{\sqrt{2}}\langle\nearrow|\searrow\rangle\right|^2 = \frac{1}{2}.$$

Hence, the probability of source photons passing though filters A and B is $p(\mathrm{AB}) = p(\mathrm{A})p(\mathrm{B}) = 1/4$.

The effect of filter C, then, is to measure $|\nearrow\rangle$ by projecting against $|\uparrow\rangle$. Note that

$$|\nearrow\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\rightarrow\rangle).$$

The probability of these photons getting through filter C is

$$|\langle\rightarrow|\nearrow\rangle|^2 = \left|\langle\rightarrow|\left[\frac{1}{\sqrt{2}}(|\uparrow\rangle + |\rightarrow\rangle)\right]\right|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

Therefore we get $|\rightarrow\rangle$ with another 1/2 decrease in intensity (so 1/8 overall).

B.5 No-cloning theorem

Copying and erasing are two of the fundamental (blackboard-inspired) operations of conventional computing. However, the *No-cloning Theorem* of quantum mechanics states that it is impossible to copy the state of a qubit. To see this, assume on the contrary that we have a unitary transformation U that does the copying, so that $U(|\psi\rangle \otimes |c\rangle) = |\psi\rangle \otimes |\psi\rangle$, where $|c\rangle$ is an arbitrary constant qubit (actually, $|c\rangle$ can be any quantum state; see Exer. III.11). That is, $U|\psi c\rangle = |\psi\psi\rangle$. Next suppose that $|\psi\rangle = a|0\rangle + b|1\rangle$. By the linearity of U:

$$\begin{array}{rcll} U\,|\psi\rangle|c\rangle &=& U\,(a|0\rangle + b|1\rangle)|c\rangle & \\ &=& U(a|0\rangle|c\rangle + b|1\rangle|c\rangle) & \text{distrib. of tensor prod.} \\ &=& U(a|0c\rangle + b|1c\rangle) & \\ &=& a(U|0c\rangle) + b(U|1c\rangle) & \text{linearity} \\ &=& a|00\rangle + b|11\rangle & \text{copying property.} \end{array}$$

On the other hand, by expanding $|\psi\psi\rangle$ we have:

$$\begin{array}{rcl} U\,|\psi c\rangle &=& |\psi\psi\rangle \\ &=& (a|0\rangle + b|1\rangle) \otimes (a|0\rangle + b|1\rangle) \\ &=& a^2|00\rangle + ba|10\rangle + ab|01\rangle + b^2|11\rangle. \end{array}$$

Note that these two expansions cannot be made equal in general, so no such unitary transformation exists. Cloning is possible only in the special cases $a = 0, b = 1$ or $a = 1, b = 0$, that is, only where we know that we are cloning a determinate (classical) basis state. The inability to simply copy a quantum state is one of the characteristics of quantum computation that makes it significantly different from classical computation.

B.6 Entanglement

B.6.a Entangled and decomposable states

The possibility of *entangled quantum states* is one of the most remarkable characteristics distinguishing quantum from classical systems. Suppose that $\mathcal{H}'$ and $\mathcal{H}''$ are the state spaces of two quantum systems. Then $\mathcal{H} = \mathcal{H}' \otimes \mathcal{H}''$ is the state space of the *composite system* (Postulate 4). For simplicity, suppose that both spaces have the basis $\{|0\rangle, |1\rangle\}$. Then $\mathcal{H}' \otimes \mathcal{H}''$ has the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. (Recall that $|01\rangle = |0\rangle \otimes |1\rangle$, etc.) Arbitrary elements of $\mathcal{H}' \otimes \mathcal{H}''$ can be written in the form

$$\sum_{j,k=0,1} c_{jk}|jk\rangle = \sum_{j,k=0,1} c_{jk}\ |j'\rangle \otimes |k''\rangle.$$

Sometimes the state of the composite systems can be written as the tensor product of the states of the subsystems, $|\psi\rangle = |\psi'\rangle \otimes |\psi''\rangle$. Such a state is called a *separable*, *decomposable* or *product state*. In other cases the state cannot be decomposed, in which case it is called an *entangled state*

For an example of an entangled state, consider the *Bell state* $|\beta_{01}\rangle$, which might arise from a process that produced two particles with opposite spin (but without determining which is which):

$$|\beta_{01}\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \stackrel{\text{def}}{=} |\Phi^+\rangle. \tag{III.2}$$

(The notations $|\beta_{01}\rangle$ and $|\Phi^+\rangle$ are both used.) Note that the states $|01\rangle$ and $|10\rangle$ both have probability 1/2. Such a state might arise, for example, from a process that emits two particles with opposite spin angular momentum in order to preserve conservation of spin angular momentum.

To show that $|\beta_{01}\rangle$ is entangled, we need to show that it cannot be decomposed, that is, that we cannot write $|\beta_{01}\rangle = |\psi'\rangle \otimes |\psi''\rangle$, for two state vectors $|\psi'\rangle = a_0|0\rangle + a_1|1\rangle$ and $|\psi''\rangle = b_0|0\rangle + b_1|1\rangle$. Let's try a separation or decomposition:

$$|\beta_{01}\rangle \stackrel{?}{=} (a_0|0\rangle + a_1|1\rangle) \otimes (b_0|0\rangle + b_1|1\rangle).$$

Multiplying out the RHS yields:

$$a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle.$$

Therefore we must have $a_0b_0 = 0$ and $a_1b_1 = 0$. But this implies that either $a_0b_1 = 0$ or $a_1b_0 = 0$ (as opposed to $1/\sqrt{2}$), so the decomposition is impossible.

For an example of a decomposable state, consider $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. Writing out the product $(a_0|0\rangle + a_1|1\rangle) \otimes (b_0|0\rangle + b_1|1\rangle)$ as before, we require $a_0b_0 = a_0b_1 = a_1b_0 = a_1b_1 = \frac{1}{2}$. This is satisfied by $a_0 = a_1 = b_0 = b_1 = \frac{1}{\sqrt{2}}$, therefore the state is decomposable.

In addition to Eq. III.2, the other three Bell states are defined:

$$|\beta_{00}\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \stackrel{\text{def}}{=} |\Psi^+\rangle, \tag{III.3}$$

$$|\beta_{10}\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \stackrel{\text{def}}{=} |\Psi^-\rangle, \tag{III.4}$$

$$|\beta_{11}\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \stackrel{\text{def}}{=} |\Phi^-\rangle. \tag{III.5}$$

The Ψ states have two identical qubits, the Φ states have opposite qubits. The $+$ superscript indicates they are added, the $-$ that they are subtracted. The general definition is:

$$|\beta_{xy}\rangle = \frac{1}{\sqrt{2}}(|0, y\rangle + (-1)^x|1, \neg y\rangle).$$

Remember this useful formula! The Bell states are orthogonal and in fact constitute a basis for $\mathcal{H}' \otimes \mathcal{H}''$ (exercise).

B.6.b EPR paradox

The EPR Paradox was proposed by Einstein, Podolsky, and Rosen in 1935 to show problems in quantum mechanics. Our discussion here will be informal.

Suppose a source produces an entangled *EPR pair* (or *Bell state*) $|\Psi^+\rangle = |\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, and the entangled particles are sent to Alice and Bob. If Alice measures her particle and gets $|0\rangle$, then that collapses the state to $|00\rangle$, and so Bob will have to get $|0\rangle$ if he measures his particle. Likewise, if Alice happens to get $|1\rangle$, Bob is also required to get $|1\rangle$ if he measures. This happens instantaneously (but it does not permit faster-than-light communication, as explained below).

One explanation is that there is some internal state in the particles that will determine the result of the measurement. Both particles have the same internal state. Such *hidden-variable theories* of quantum mechanics assume that particles are "really" in some definite state and that superposition reflects our ignorance of its state. However, they cannot explain the results of measurements in different bases. In 1964 John Bell showed that any local hidden variable theory would lead to measurements satisfying a certain inequality (Bell's inequality). Actual experiments, which have been conducted over tens of kilometers, violate Bell's inequality. Thus local hidden variable theories cannot be correct.

Another explanation is that Alice's measurement affects Bob's (or vice versa, if Bob measures first). These are called *causal theories*. According to relativity theory, however, in some frames of reference Alice's measurement comes first, and in other frames, Bob's comes first. Therefore there is no consistent cause-effect relation. This is why Alice and Bob cannot use entangled pairs to communicate.

B.7 Uncertainty principle (supplementary)

You might be surprised that the famous Heisenberg uncertainty principle is not among the postulates of quantum mechanics. That is because it is not a postulate, but a theorem, which can be proved from the postulates. This section is optional, since the uncertainty principle is not required for quantum computation.

B.7.a Informally

The uncertainty principle states a lower bound on the precision with which certain pairs of variables, called *conjugate variables*, can be measured. These are such pairs as position and momentum, and energy and time. For example, the same state can be represented by the wave function $\psi(x)$ as a function of space and by $\phi(p)$ as a function of momentum. The most familiar version of the Heisenberg principle, limits the precision with which location and momentum can be measured simultaneously: $\Delta x \, \Delta p \geq \hbar/2$, where the reduced Plank constant $\hbar = h/2\pi$, where h is Planck's constant.

It is often supposed that the uncertainty principle is a manifestation of the *observer effect*, the inevitable effect that measuring a system has on it, but this is not the case. "While it is true that measurements in quantum mechanics cause disturbance to the system being measured, this is most emphatically *not* the content of the uncertainty principle."(Nielsen & Chuang, 2010, p. 89)

Often the uncertainty principle is a result of the variables representing measurements in two bases that are Fourier transforms of each other. Consider an audio signal $\psi(t)$ and its Fourier transform $\Psi(\omega)$ (its spectrum). Note that ψ is a function of time, with dimension t, and its spectrum Ψ is a function of frequency, with dimension t^{-1}. They are reciprocals of each other, and that is always the case with Fourier transforms. Simultaneous measurement in the time and frequency domains obeys the uncertainty relation $\Delta t \Delta \omega \geq 1/2$. (For more details on this, including an intuitive explanation, see **?**, ch. 6.)

Time and energy are also conjugate, as a result of the de Broglie relation, according to which energy is proportional to frequency: $E = h\nu$ (ν in Hertz, or cycles per second) or $E = \hbar\omega$ (ω in radians per second). Therefore simultaneous measurement in the time and energy domains obeys the uncertainty principle $\Delta t \Delta E \geq \hbar/2$.

More generally, the observables are represented by Hermitian operators P, Q that do not commute. That is, to the extent they do not commute, to that extent you cannot measure them both (because you would have to do either PQ or QP, but they do not give the same result). The best interpretation of the uncertainty principle is that if you set up the experiment multiple times, and measure the outcomes, you will find

$$2\,\Delta P\,\Delta Q \geq |\langle[P,Q]\rangle|,$$

where P and Q are conjugate observables. (The commutator $[P,Q]$ is defined below, Def. B.2, p. 97.)

Note that this is a *purely mathematical* result (proved in Sec. B.7.b). Any system obeying the QM postulates will have uncertainty principles for every pair of non-commuting observables.

B.7.b Formally

In this section we'll derive the uncertainty principle more formally. Since it deals with the variances of measurements, we begin with their definition. To understand the motivation for these definitions, suppose we have a quantum system (such as an atom) that can be in three distinct states $|\text{ground}\rangle$, $|\text{first excited}\rangle$, $|\text{second excited}\rangle$ with energies e_0, e_1, e_2, respectively. Then the *energy observable* is the operator

$$\begin{aligned} E &= e_0|\text{ground}\rangle\langle\text{ground}| + e_1|\text{first excited}\rangle\langle\text{first excited}| \\ &\quad + e_2|\text{second excited}\rangle\langle\text{second excited}|, \end{aligned}$$

or more briefly, $\sum_{m=0}^{2} e_m|m\rangle\langle m|$.

Definition B.1 (observable) *An* observable M *is a Hermitian operator on the state space.*

An observable M has a spectral decomposition (Sec. A.2.g):

$$M = \sum_{m=1}^{N} e_m P_m,$$

where the P_m are *projectors* onto the eigenspaces of M, and the eigenvalues e_m are the corresponding measurement results. The projector P_m projects into the eigenspace corresponding to eigenvalue e_m. (For projectors, see Sec. A.2.d.) Since an observable is described by a Hermitian operator M, it has a spectral decomposition with real eigenvalues, $M = \sum_{m=1}^{N} e_m |m\rangle\langle m|$, where $|m\rangle$ is the measurement basis. Therefore we can write $M = UEU^\dagger$, where $E = \text{diag}(e_1, e_2, \ldots, e_N)$, $U = (|1\rangle, |2\rangle, \ldots, |N\rangle)$, and

$$U^\dagger = (|1\rangle, |2\rangle, \ldots, |N\rangle)^\dagger = \begin{pmatrix} \langle 1| \\ \langle 2| \\ \vdots \\ \langle N| \end{pmatrix}.$$

$U^\dagger$ expresses the state in the measurement basis and U translates back. In the measurement basis, the matrix for an observable is a diagonal matrix: $E = \text{diag}(e_1, \ldots, e_N)$. The probability of measuring e_m is

$$p(m) = \langle \psi \mid P_m^\dagger P_m \mid \psi \rangle = \langle \psi \mid P_m P_m \mid \psi \rangle = \langle \psi \mid P_m \mid \psi \rangle.$$

We can derive the mean or expectation value of an energy measurement for a given quantum state $|\psi\rangle$:

$$\begin{aligned}
\langle E \rangle &\overset{\text{def}}{=} \mu_E \overset{\text{def}}{=} \mathcal{E}\{E\} \\
&= \sum_m e_m p(m) \\
&= \sum_m e_m \langle \psi \mid m \rangle \langle m \mid \psi \rangle \\
&= \sum_m \langle \psi|\, e_m |m\rangle\langle m|\, |\psi\rangle \\
&= \langle \psi| \left(\sum_m e_m |m\rangle\langle m| \right) |\psi\rangle \\
&= \langle \psi \mid E \mid \psi \rangle.
\end{aligned}$$

This formula can be used to derive the standard deviation σ_E and variance σ_E^2, which are important in the uncertainty principle:

$$\begin{aligned}
\sigma_E^2 &\overset{\text{def}}{=} (\Delta E)^2 \overset{\text{def}}{=} \text{Var}\{E\} \\
&= \mathcal{E}\{(E - \langle E \rangle)^2\} \\
&= \langle E^2 \rangle - \langle E \rangle^2 \\
&= \langle \psi \mid E^2 \mid \psi \rangle - (\langle \psi \mid E \mid \psi \rangle)^2.
\end{aligned}$$

Note that E^2, the matrix E multiplied by itself, is also the operator that measures the square of the energy, $E^2 = \sum_j e_m^2 |m\rangle\langle m|$. (This is because E is diagonal in this basis; alternately, E^2 can be interpreted as an operator function.)

We now proceed to the derivation of the uncertainty principle.[3]

Definition B.2 (commutator) *If $L, M : \mathcal{H} \rightarrow \mathcal{H}$ are linear operators, then their* commutator *is defined:*

$$[L, M] = LM - ML. \tag{III.6}$$

Remark B.1 *In effect, $[L, M]$ distills out the non-commutative part of the product of L and M. If the operators commute, then $[L, M] = \mathbf{0}$, the identically zero operator. Constant-valued operators always commute ($cL = Lc$), and so $[c, L] = \mathbf{0}$.*

Definition B.3 (anti-commutator) *If $L, M : \mathcal{H} \rightarrow \mathcal{H}$ are linear operators, then their* anti-commutator *is defined:*

$$\{L, M\} = LM + ML. \tag{III.7}$$

If $\{L, M\} = \mathbf{0}$, we say that L and M anti-commute*, $LM = -ML$.*

See B.2.c (p. 82) for the justification of the following definitions.

Definition B.4 (mean of measurement) *If M is a Hermitian operator representing an observable, then the mean value of the measurement of a state $|\psi\rangle$ is*

$$\langle M \rangle = \langle \psi \mid M \mid \psi \rangle.$$

Definition B.5 (variance and standard deviation of measurement) *If M is a Hermitian operator representing an observable, then the variance in the measurement of a state $|\psi\rangle$ is*

$$\text{Var}\{M\} = \langle (M - \langle M \rangle^2) \rangle = \langle M^2 \rangle - \langle M \rangle^2.$$

As usual, the standard deviation ΔM *of the measurement is defined*

$$\Delta M = \sqrt{\text{Var}\{M\}}.$$

[3]The following derivation is from **?**, ch. 5.

Proposition B.1 *If L and M are Hermitian operators on $\mathcal{H}$ and $|\psi\rangle \in \mathcal{H}$, then*

$$4\langle\psi \mid L^2 \mid \psi\rangle \, \langle\psi \mid M^2 \mid \psi\rangle \geq |\langle\psi \mid [L, M] \mid \psi\rangle|^2 + |\langle\psi \mid \{L, M\} \mid \psi\rangle|^2.$$

More briefly, in terms of average measurements,

$$4\langle L^2\rangle\langle M^2\rangle \geq |\langle[L, M]\rangle|^2 + |\langle\{L, M\}\rangle|^2.$$

Proof: Let $x + iy = \langle\psi \mid LM \mid \psi\rangle$. Then,

$$\begin{aligned} 2x &= \langle\psi \mid LM \mid \psi\rangle + (\langle\psi \mid LM \mid \psi\rangle)^* \\ &= \langle\psi \mid LM \mid \psi\rangle + \langle\psi \mid M^\dagger L^\dagger \mid \psi\rangle \\ &= \langle\psi \mid LM \mid \psi\rangle + \langle\psi \mid ML \mid \psi\rangle \quad \text{since } L, M \text{ are Hermitian} \\ &= \langle\psi \mid \{L, M\} \mid \psi\rangle. \end{aligned}$$

Likewise,

$$\begin{aligned} 2iy &= \langle\psi \mid LM \mid \psi\rangle - (\langle\psi \mid LM \mid \psi\rangle)^* \\ &= \langle\psi \mid LM \mid \psi\rangle - \langle\psi \mid ML \mid \psi\rangle \\ &= \langle\psi \mid [L, M] \mid \psi\rangle. \end{aligned}$$

Hence,

$$\begin{aligned} |\langle\psi \mid LM \mid \psi\rangle|^2 &= 4(x^2 + y^2) \\ &= |\langle\psi \mid [L, M] \mid \psi\rangle|^2 + |\langle\psi \mid \{L, M\} \mid \psi\rangle|^2. \end{aligned}$$

Let $|\lambda\rangle = L|\psi\rangle$ and $|\mu\rangle = M|\psi\rangle$. By the Cauchy-Schwarz inequality, $\| |\lambda\rangle \| \, \| |\mu\rangle \| \geq |\langle\lambda \mid \mu\rangle|$ and so $\langle\lambda \mid \lambda\rangle \, \langle\mu \mid \mu\rangle \geq |\langle\lambda \mid \mu\rangle|^2$. Hence,

$$\langle\psi \mid L^2 \mid \psi\rangle \, \langle\psi \mid M^2 \mid \psi\rangle \geq |\langle\psi \mid LM \mid \psi\rangle|^2.$$

The result follows.

□

Proposition B.2 *Prop. B.1 can be weakened into a more useful form:*

$$4\langle\psi \mid L^2 \mid \psi\rangle \, \langle\psi \mid M^2 \mid \psi\rangle \geq |\langle\psi \mid [L, M] \mid \psi\rangle|^2,$$

or $4\langle L^2\rangle\langle M^2\rangle \geq |\langle[L, M]\rangle|^2$

Proposition B.3 (uncertainty principle) *If Hermitian operators P and Q are measurements (observables), then*

$$\Delta P \, \Delta Q \geq \frac{1}{2} |\langle \psi \mid [P, Q] \mid \psi \rangle|.$$

That is, $\Delta P \, \Delta Q \geq |\langle [P, Q] \rangle|/2$. So the product of the variances is bounded below by the degree to which the operators do not commute.

Proof: Let $L = P - \langle P \rangle$ and $M = Q - \langle Q \rangle$. By Prop. B.2 we have

$$\begin{array}{rcl} 4 \, \mathrm{Var}\{P\} \, \mathrm{Var}\{Q\} & = & 4 \langle L^2 \rangle \langle M^2 \rangle \\ & \geq & |\langle [L, M] \rangle|^2 \\ & = & | \, \langle [P - \langle P \rangle, Q - \langle Q \rangle] \rangle \, |^2 \\ & = & |\langle [P, Q] \rangle|^2. \end{array}$$

Hence,

$$2 \, \Delta P \Delta Q \geq |\langle [P, Q] \rangle|$$

$\square$

C Quantum information

C.1 Qubits

C.1.a Single qubits

Just as the bits 0 and 1 are represented by distinct physical states in a conventional computer, so the *quantum bits* (or *qubits*) $|0\rangle$ and $|1\rangle$ are represented by distinct quantum states. We call $|0\rangle$ and $|1\rangle$ the *computational* or *standard* measurement basis. What distinguishes qubits from classical bits is that they can be in a superposition of states, $a_0|0\rangle + a_1|1\rangle$, for $a_0, a_1 \in \mathbb{C}$, where $|a_0|^2 + |a_1|^2 = 1$. If we measure this state in the computational basis, we will observe $|0\rangle$ with probability $|a_0|^2$ and likewise for $|1\rangle$; after measurement the qubit is in the observed state. This applies, of course, to measurement in any basis. I will depict the measurement possibilities this way:

$$\begin{aligned} a_0|0\rangle + a_1|1\rangle &\xrightarrow{|a_0|^2} |0\rangle, \\ a_0|0\rangle + a_1|1\rangle &\xrightarrow{|a_1|^2} |1\rangle. \end{aligned}$$

The following *sign basis* is often useful:

$$|+\rangle \overset{\text{def}}{=} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \tag{III.8}$$

$$|-\rangle \overset{\text{def}}{=} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{III.9}$$

Notice that $|+\rangle$ is "halfway" between $|0\rangle$ and $|1\rangle$, and likewise $|-\rangle$ is halfway between $|0\rangle$ and $-|1\rangle$. Draw them to be sure you see this. As a consequence (Exer. III.34):

$$\begin{aligned} |0\rangle &= \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle), \\ |1\rangle &= \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle). \end{aligned}$$

To remember this, think $(+x) + (-x) = 0$ and $(+x) - (-x) = (+2x)$, which is nonzero (this is just a mnemonic).

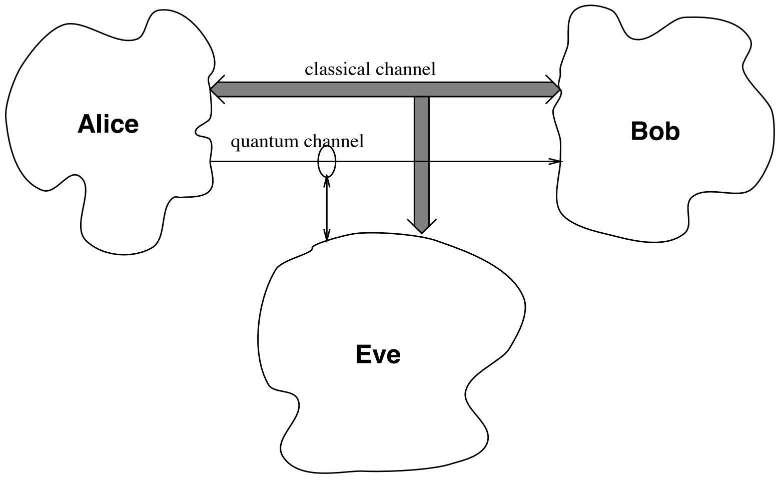
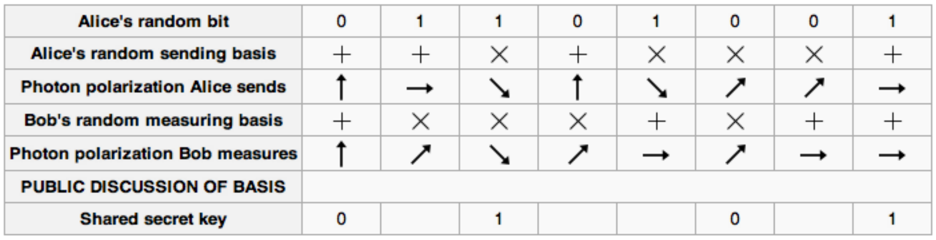
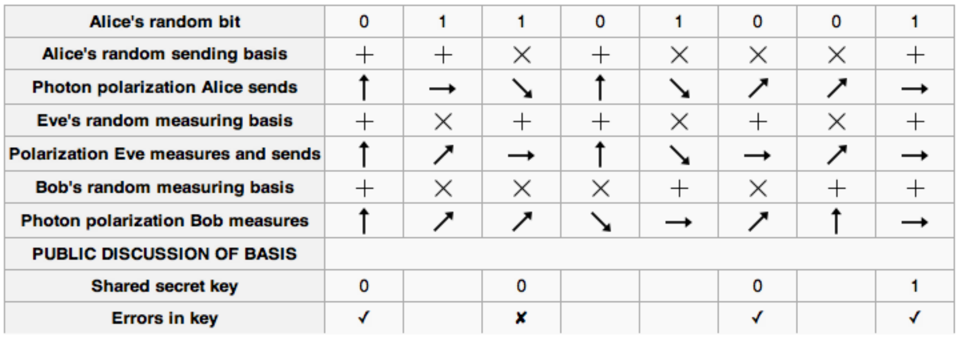


Figure III.6: Quantum key distribution [from Rieffel & Polak (2000)].



Alice's random bit	0	1	1	0	1	0	0	1
Alice's random sending basis	+	+	×	+	×	×	×	+
Photon polarization Alice sends	↑	→	↘	↑	↘	↗	↗	→
Bob's random measuring basis	+	×	×	×	+	×	+	+
Photon polarization Bob measures	↑	↗	↘	↗	→	↗	→	→
PUBLIC DISCUSSION OF BASIS								
Shared secret key	0		1			0		1

Figure III.7: Example if QKD without interference. [fig. from wikipedia]



Alice's random bit	0	1	1	0	1	0	0	1
Alice's random sending basis	$+$	$+$	$\times$	$+$	$\times$	$\times$	$\times$	$+$
Photon polarization Alice sends	↑	→	↘	↑	↘	↗	↗	→
Eve's random measuring basis	$+$	$\times$	$+$	$+$	$\times$	$+$	$\times$	$+$
Polarization Eve measures and sends	↑	↗	→	↑	↘	→	↗	→
Bob's random measuring basis	$+$	$\times$	$\times$	$\times$	$+$	$\times$	$+$	$+$
Photon polarization Bob measures	↑	↗	↗	↘	→	↗	↑	→
PUBLIC DISCUSSION OF BASIS								
Shared secret key	0		0			0		1
Errors in key	✓		✘			✓		✓

Figure III.8: Example if QKD with eavesdropping. [fig. from wikipedia]

C.1.b Quantum key distribution

In 1984 Bennett and Brassard showed how sequences of qubits could be used to distribute an encryption key securely.[4] This is called the "BB84 protocol." Ironically, the idea was proposed initially by Stephen Wiesner in the 1970s, but he couldn't get it published.

We are supposing that Alice is transmitting a key to Bob over two channels, one classical and one quantum. Eve may eavesdrop on both channels and even replace the signals in them. Over the quantum channel Alice will send the photons to Bob that encode the key bits in two different bases, either $\{|\uparrow\rangle, |\rightarrow\rangle\}$, which I'll call the "+ basis," or $\{|\nearrow\rangle, |\searrow\rangle\}$ (the "$\times$ basis") (respectively 0, 1 in each basis). Alice chooses randomly the basis in which to encode her bits (see Fig. III.7). Bob will measure the photons according to these two bases, also chosen randomly and independently of Alice. After the transmission, Alice and Bob will communicate over the classical channel and compare their random choices; where they picked the same basis, they will keep the bit, otherwise they will discard it. (They will have agreed on about 50% of the choices.)

Suppose Eve is eavesdropping on the quantum channel, measuring the qubits and retransmitting them to Bob (see Fig. III.8). About 50% of the time, she will guess the wrong basis, and will also resend it in this same incorrect basis. If this is one of the times Alice and Bob chose the same basis, the bit will nevertheless be incorrect about half of the time (the times

[4]This section is based on Rieffel & Polak (2000), which is also the source for otherwise unattributed quotes.

Eve chose the wrong basis). That is, about 50% of the time Eve picks the same basis as Alice, so she reads the bit correctly and transmits it to Bob correctly. About 50% of the time Eve guesses the wrong basis. She will know this, if she is listening in on the classical channel, but she has already transmitted it to Bob in the wrong basis. If this is a case in which Alice and Bob used the same basis (and so Bob should get it correct), he will get it incorrect 50% of the time, since Eve transmitted it in the other basis. So 25% of the bits that should be correct will be wrong. This high error rate will be apparent to Alice and Bob if they have been using an error-detecting code for the key. (In effect Eve is introducing significant, detectable noise into the channel.) Furthermore, Eve's version of the key will be about 25% incorrect. Therefore Bob knows that the key was not transmitted securely and Eve gets an incorrect key.

This is only the most basic technique, and it has some vulnerabilities, and so other techniques have been proposed, but they are outside the scope of this book. "The highest bit rate system currently demonstrated exchanges secure keys at 1 Mbit/s (over 20 km of optical fibre) and 10 kbit/s (over 100 km of fibre)"[5] "As of March 2007 the longest distance over which quantum key distribution has been demonstrated using optic fibre is 148.7 km, achieved by Los Alamos National Laboratory/NIST using the BB84 protocol." In Aug. 2015 keys were distributed over a 307 km optical cable, with 12.7 kbps key generation rate. "The distance record for free space QCD [quantum key distribution] is 144 km between two of the Canary Islands, achieved by a European collaboration using entangled photons (the Ekert scheme) in 2006,[7] and using BB84 enhanced with decoy states[8] in 2007.[9] The experiments suggest transmission to satellites is possible, due to the lower atmospheric density at higher altitudes." At least three companies offer commercial QKD. "Quantum encryption technology provided by the Swiss company Id Quantique was used in the Swiss canton (state) of Geneva to transmit ballot results to the capitol in the national election occurring on October 21, 2007." Four QKD networks have been in operation since mid-late 2000s. Among them,

> [t]he world's first computer network protected by quantum key distribution was implemented in October 2008, at a scientific conference in Vienna. The name of this network is SECOQC (**Se**cure **Co**mmunication Based on **Q**uantum **C**ryptography) and

[5]`https://en.wikipedia.org/wiki/Quantum_key_distribution` (accessed 12-09-18).

EU funded this project. The network used 200 km of standard fibre optic cable to interconnect six locations across Vienna and the town of St Poelten located 69 km to the west.

C.1.c Multiple qubits

We can combine multiple qubits into a *quantum register*. By Postulate 4, if $\mathcal{H}$ is the state space of one qubit, then the tensor power $\mathcal{H}^{\otimes n}$ will be the state space of an n-qubit quantum register. The computational basis of this space is the set of all vectors $|b_1 b_2 \cdots b_n\rangle$ with $b_k \in \mathbf{2}$. (I define $\mathbf{2} \overset{\text{def}}{=} \{0, 1\}$ to be the set of bits, and in general I use a boldface integer $\mathbf{N}$ for the set integers $\{0, 1, \ldots, N-1\}$.) Therefore the dimension of the space $\mathcal{H}^{\otimes n}$ is 2^n, and the set of states is the set of normalized vectors in $\mathbb{C}^{2^n}$. For 10 qubits we are dealing with 1024-dimensional complex vectors (because each of the 2^{10} basis vectors has its own complex amplitude). This is a huge space, exponentially larger than the 2^n classical n-bit strings. This is part of the origin of *quantum parallelism*, because we can compute on all of these qubit strings in parallel. Consider a quantum computer with 500 qubits; it could be very small (e.g., 500 atoms), but it is computing in a space of 2^{500} complex numbers. Note that 2^{500} is more than the number of particles in the universe times the age of the universe in femtoseconds! That is, a 500-qubit quantum computer is equivalent to a universe-sized computer working at high speed since the Big Bang.

Whereas an ordinary direct product has dimension $\dim(S \times T) = \dim S + \dim T$, a tensor product has dimension $\dim(S \otimes T) = \dim S \times \dim T$. Hence if $\dim S = 2$, $\dim S^{\otimes n} = 2^n$.

Measuring some of the qubits in a register causes partial collapse of the quantum state. Suppose we have a composite state

$$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle,$$

and we measure just the first bit. We will get 0 with probability $|a_{00}|^2 + |a_{01}|^2$ and it will collapse into the state $a_{00}|00\rangle + a_{01}|01\rangle$, but we must renormalize it:

$$|\psi'\rangle = \frac{a_{00}|00\rangle + a_{01}|01\rangle}{\sqrt{|a_{00}|^2 + |a_{01}|^2}}.$$

Do this by striking out all terms in $|\psi\rangle$ that have 1 in the first qubit.

$$|\psi\rangle \overset{|a_{00}|^2+|a_{01}|^2}{\longrightarrow} a_{00}|00\rangle + a_{01}|01\rangle \cong \frac{a_{00}|00\rangle + a_{01}|01\rangle}{\sqrt{|a_{00}|^2 + |a_{01}|^2}}.$$

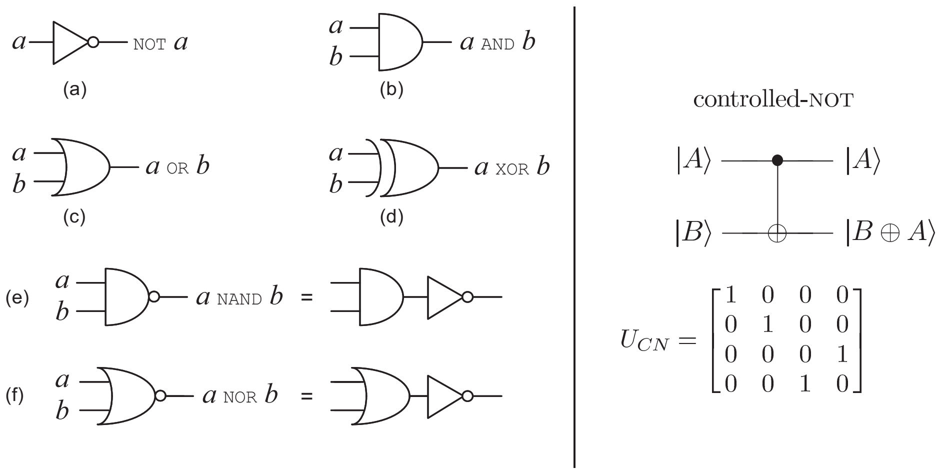

a NOT a
(a)
a b a AND b
(b)
a b a OR b
(c)
a b a XOR b
(d)
(e) a b a NAND b =
(f) a b a NOR b =
controlled-NOT
$|A\rangle$ $|A\rangle$
$|B\rangle$ $|B \oplus A\rangle$
$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Figure III.9: Left: classical gates. Right: controlled-Not gate. [from Nielsen & Chuang (2010, Fig. 1.6)]

C.2 Quantum gates

Quantum gates are analogous to ordinary logic gates (the fundamental building blocks of circuits), but they must be unitary transformations (see Fig. III.9, left, for ordinarty logic gates). Fortunately, Bennett, Fredkin, and Toffoli have already shown how all the usual logic operations can be done reversibly. In this section you will learn the most important quantum gates.

C.2.a Single-qubit gates

The NOT gate is simple because it is reversible: $\text{NOT}|0\rangle = |1\rangle$, $\text{NOT}|1\rangle = |0\rangle$. Its desired behavior can be represented:

$$\begin{aligned} \text{NOT}: \quad |0\rangle &\mapsto |1\rangle \\ |1\rangle &\mapsto |0\rangle. \end{aligned}$$

Note that defining it on a basis defines it on all quantum states. Therefore it can be written as a sum of dyads (outer products):

$$\text{NOT} = |1\rangle\langle 0| + |0\rangle\langle 1|.$$

You can read this, "return $|1\rangle$ if the input is $|0\rangle$, and return $|0\rangle$ if the input is $|1\rangle$." Recall that in the standard basis $|0\rangle = (1\ 0)^{\mathrm{T}}$ and $|1\rangle = (0\ 1)^{\mathrm{T}}$.

Therefore NOT can be represented in the standard basis by computing the outer products:

$$\text{NOT} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} (1\ 0) + \begin{pmatrix} 1 \\ 0 \end{pmatrix} (0\ 1) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The first column represents the result for $|0\rangle$, which is $|1\rangle$, and the second represents the result for $|1\rangle$, which is $|0\rangle$.

Although NOT is defined in terms of the computational basis vectors, it applies to any qubit, in particular to superpositions of $|0\rangle$ and $|1\rangle$:

$$\text{NOT}(a|0\rangle + b|1\rangle) = a\text{NOT}|0\rangle + b\text{NOT}|1\rangle = a|1\rangle + b|0\rangle = b|0\rangle + a|1\rangle.$$

Therefore, NOT exchanges the amplitudes of $|0\rangle$ and $|1\rangle$.

In quantum mechanics, the NOT transformation is usually called X. It is one of four useful unitary operations, called the *Pauli matrices*, which are worth remembering. In the standard basis:

$$I \stackrel{\text{def}}{=} \sigma_0 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{(III.10)}$$

$$X \stackrel{\text{def}}{=} \sigma_x \stackrel{\text{def}}{=} \sigma_1 \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \text{(III.11)}$$

$$Y \stackrel{\text{def}}{=} \sigma_y \stackrel{\text{def}}{=} \sigma_2 \stackrel{\text{def}}{=} \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} \quad \text{(III.12)}$$

$$Z \stackrel{\text{def}}{=} \sigma_z \stackrel{\text{def}}{=} \sigma_3 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad \text{(III.13)}$$

We have seen that X is NOT, and I is obviously the identity gate. Z leaves $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$. It is called the phase-flip operator because it flips the phase of the $|1\rangle$ component by π relative to the $|0\rangle$ component. (Recall that global/absolute phase doesn't matter.) The Pauli matrices span the space of 2×2 complex matrices (Exer. III.21).

Note that $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$. It is thus the analog in the sign basis of X (NOT) in the computational basis. What is the effect of Y on the computational basis vectors? (Exer. III.15)

Note that there is an alternative definition of Y that differs only in global phase:

$$Y \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

This is a $90° = \pi/2$ counterclockwise rotation: $Y(a|0\rangle + b|1\rangle) = b|0\rangle - a|1\rangle$. Draw a diagram to make sure you see this.

Note that the Pauli operations apply to *any* state, not just basis states. The X, Y, and Z operators get their names from the fact that they reflect state vectors along the x, y, z axes of the Bloch-sphere representation of a qubit, which we will not use in this book. Since they are reflections, they are Hermitian (their own inverses).

C.2.b Multiple-qubit gates

We know that any logic circuit can be built up from NAND gates. Can we do the same for quantum logic, that is, is there a universal quantum logic gate? We can't use NAND, because it's not reversible, but we will see that there are universal sets of quantum gates.

The *controlled-NOT* or CNOT gate has two inputs: the first determines what it does to the second (negate it or not).

$$\begin{array}{rrcl} \text{CNOT}: & |00\rangle & \mapsto & |00\rangle \\ & |01\rangle & \mapsto & |01\rangle \\ & |10\rangle & \mapsto & |11\rangle \\ & |11\rangle & \mapsto & |10\rangle. \end{array}$$

Its first argument is called the *control* and its second is called the *target*, *controlled*, or *data* qubit. It is a simple example of conditional quantum computation. CNOT can be translated into a sum-of-dyads representation (Sec. A.2.d), which can be written in matrix form (Ex. III.24, p. 196):

$$\begin{array}{rcl} \text{CNOT} & = & |00\rangle\langle 00| \\ & + & |01\rangle\langle 01| \\ & + & |11\rangle\langle 10| \\ & + & |10\rangle\langle 11| \end{array}$$

We can also define it (for $x, y \in \mathbf{2}$), $\text{CNOT}|xy\rangle = |xz\rangle$, where $z = x \oplus y$, the exclusive OR of x and y. That is, $\text{CNOT}|x, y\rangle = |x, x \oplus y\rangle$ CNOT is the only non-trivial 2-qubit reversible logic gate. Note that CNOT is unitary since obviously $\text{CNOT} = \text{CNOT}^\dagger$ (which you can show using its dyadic representation or its matrix representation, Ex. III.24, p. 196). See the right

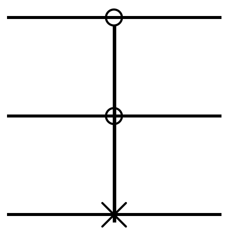


Figure III.10: Diagram for CCNOT or Toffoli gate [fig. from Nielsen & Chuang (2010)]. Sometimes the $\times$ is replaced by $\oplus$ because $\text{CCNOT}|xyz\rangle = |x, y, xy \oplus z\rangle$.

panel of Fig. III.9 (p. 105) for the matrix and note the diagram notation for CNOT.

CNOT can be used to produce an entangled state:

$$\text{CNOT}\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right]|0\rangle = \text{CNOT}\frac{1}{\sqrt{2}}(|00\rangle+|10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle+|11\rangle) = |\beta_{00}\rangle.$$

Note also that $\text{CNOT}|x, 0\rangle = |x, x\rangle$, that is, FAN-OUT, which would seem to violate the No-cloning Theorem, but it works as expected only for $x \in \mathbf{2}$. In general $\text{CNOT}|\psi\rangle|0\rangle \neq |\psi\rangle|\psi\rangle$ (Exer. III.25).

Another useful gate is the three-input/output *Toffoli gate* or *controlled-controlled-NOT*. It negates the third qubit if and only if the first two qubits are both 1. For $x, y, z \in \mathbf{2}$,

$$\begin{aligned}\text{CCNOT}|1, 1, z\rangle &\overset{\text{def}}{=} |1, 1, \neg z\rangle,\\ \text{CCNOT}|x, y, z\rangle &\overset{\text{def}}{=} |x, y, z\rangle, \quad \text{otherwise.}\end{aligned}$$

That is, $\text{CCNOT}|x, y, z\rangle = |x, y, xy \oplus z\rangle$. All the Boolean operations can be implemented (reversibly!) by using Toffoli gates (Exer. III.30). For example, $\text{CCNOT}|x, y, 0\rangle = |x, y, x \wedge y\rangle$. Thus it is a universal gate for quantum logic.

In Jan. 2009 CCNOT was implemented successfully using trapped ions.[6]

[6] Monz, T.; Kim, K.; Hänsel, W.; Riebe, M.; Villar, A. S.; Schindler, P.; Chwalla, M.; Hennrich, M. et al. (Jan 2009). "Realization of the Quantum Toffoli Gate with Trapped Ions." *Phys. Rev. Lett.* **102** (4): 040501. `arXiv:0804.0082`.

C.2.c Walsh-Hadamard transformation

Recall that the sign basis is defined $|+\rangle \overset{\text{def}}{=} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle \overset{\text{def}}{=} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The *Hadamard transformation* or *gate* is defined:

$$\begin{aligned} H|0\rangle &\overset{\text{def}}{=} |+\rangle, &\text{(III.14)}\\ H|1\rangle &\overset{\text{def}}{=} |-\rangle. &\text{(III.15)}\end{aligned}$$

In sum-of-dyads form: $H \overset{\text{def}}{=} |+\rangle\langle 0| + |-\rangle\langle 1|$. In matrix form (with respect to the standard basis):

$$H \overset{\text{def}}{=} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{III.16}$$

Note that H is self-adjoint, $H^2 = I$ (since $H^\dagger = H$). H can be defined also in terms of the Pauli matrices: $H = (X + Z)/\sqrt{2}$ (Exer. III.38).

The H transform can be used to transform the computational basis into the sign basis and back (Exer. III.37):

$$\begin{aligned} H(a|0\rangle + b|1\rangle) &= a|+\rangle + b|-\rangle,\\ H(a|+\rangle + b|-\rangle) &= a|0\rangle + b|1\rangle. \end{aligned}$$

Alice and Bob could use this in quantum key distribution.

When applied to a $|0\rangle$, H generates an (equal-amplitude) superposition of the two bit-values, $H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. This is a useful way of generating a superposition of both possible input bits, and the Walsh transform, a tensor power of H, can be applied to a quantum register to generate a superposition of all possible register values. Consider the $n = 2$ case:

$$\begin{aligned} H^{\otimes 2}|\psi, \phi\rangle &= (H \otimes H)(|\psi\rangle \otimes |\phi\rangle)\\ &= (H|\psi\rangle) \otimes (H|\phi\rangle) \end{aligned}$$

In particular,

$$\begin{aligned} H^{\otimes 2}|00\rangle &= (H|0\rangle) \otimes (H|0\rangle)\\ &= |+\rangle^{\otimes 2}\\ &= \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right]^{\otimes 2}\\ &= \left(\frac{1}{\sqrt{2}}\right)^2 (|0\rangle + |1\rangle)(|0\rangle + |1\rangle)\\ &= \frac{1}{\sqrt{2^2}}(|00\rangle + |01\rangle + |10\rangle + |11\rangle). \end{aligned}$$

Notice that this is an equal superposition of all possible values of the 2-qubit register. (I wrote the amplitude in a complicated way, $1/\sqrt{2^2}$, to help you see the general case.) In general,

$$\begin{aligned}
H^{\otimes n}|0\rangle^{\otimes n} &= \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle)^{\otimes n} \\
&= \frac{1}{\sqrt{2^n}} \overbrace{(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \cdots \otimes (|0\rangle + |1\rangle)}^{n} \\
&= \frac{1}{\sqrt{2^m}} (|00\ldots00\rangle + |00\ldots01\rangle + \cdots + |11\ldots11\rangle) \\
&= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}\in\mathbf{2}^n} |\mathbf{x}\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |\mathbf{x}\rangle.
\end{aligned}$$

Note that "$2^n - 1$" represents a string of n 1-bits, and that $\mathbf{2} = \{0, 1\}$. Hence, $H^{\otimes n}|0\rangle^{\otimes n}$ generates an equal superposition of all the 2^n possible values of the n-qubit register. We often write $W_n = H^{\otimes n}$ for the Walsh transformation.

An linear operation applied to such a superposition state in effect applies the operation simultaneously to all 2^n possible input values. This is *exponential* quantum parallelism and suggests that quantum computation might be able to solve exponential problems much more efficiently than classical computers. To see this, suppose $U|x\rangle = |f(x)\rangle$. Then:

$$U(H^{\otimes n}|0\rangle^{\otimes n}) = U\left[\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\right] = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}U|x\rangle = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|f(x)\rangle$$

This is a superposition of the function values $f(x)$ for all of the 2^n possible values of x; it is computed by one pass through the operator U.

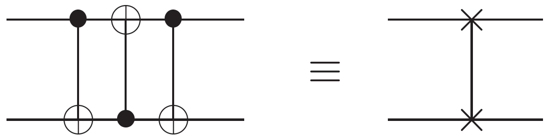


Figure III.11: Diagram for swap [from Nielsen & Chuang (2010)].

C.3 Quantum circuits

A *quantum circuit* is a sequential series of quantum transformations on a quantum register. The inputs are usually computational basis states (all $|0\rangle$ unless stated otherwise). *Quantum circuit diagrams* are drawn with time going from left to right, with the quantum gates crossing one or more "wires" (qubits) as appropriate. The circuit represents a sequence of unitary operations on a quantum register rather than physical wires.

These "circuits" are different in several respects from ordinary sequential logic circuits. First, loops (feedback) are not allowed, but you can apply transforms repeatedly. Second, FAN-IN (equivalent to OR) is not allowed, since it it not reversible or unitary. FAN-OUT is also not allowed, because it would violate the No-cloning Theorem. (N.B.: This does not contradict the universality of the Toffoli or Fredkin gates, which are universal only with respect to logical or classical states.)

Fig. III.9 (right) on page 105 shows the symbol for CNOT and its effect.

The swap operation is defined $|xy\rangle \mapsto |yx\rangle$, or explicitly

$$\text{SWAP} = \sum_{x,y \in \mathbf{2}} |yx\rangle\langle xy|.$$

We can put three CNOTs in series to swap two qubits (Exer. III.40). Swap has a special symbol as shown in Fig. III.11.

In general, any unitary operator U (on any number of qubits) can be conditionally controlled (see Fig. III.12); this is the quantum analogue of an if-then statement. If the control bit is 0, this operation does nothing, otherwise it does U. This is implemented by $|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U$. Effectively, the *operators* are entangled.

Suppose the control bit is in superposition, $|\chi\rangle = a|0\rangle + b|1\rangle$. The effect

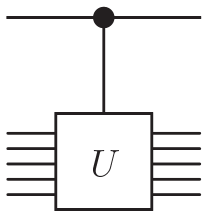


Figure 1.8. Controlled-U gate.

Figure III.12: Diagram for controlled-U [from Nielsen & Chuang (2010)].

of the conditional operation is:

$$\begin{aligned}
&(|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U)|\chi, \psi\rangle \\
&\quad = (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U)(a|0\rangle + b|1\rangle) \otimes |\psi\rangle \\
&\quad = |0\rangle\langle 0|(a|0\rangle + b|1\rangle) \otimes I|\psi\rangle + |1\rangle\langle 1|(a|0\rangle + b|1\rangle) \otimes U|\psi\rangle \\
&\quad = a|0\rangle \otimes |\psi\rangle + b|1\rangle \otimes U|\psi\rangle \\
&\quad = a|0, \psi\rangle + b|1, U\psi\rangle.
\end{aligned}$$

The result is a superposition of entangled outputs. Notice that CNOT is a special case of this construction, a controlled X.

We also have a quantum analogue for an if-then-else construction. If U_0 and U_1 are unitary operators, then we can make the choice between them conditional on a control bit as follows:

$$|0\rangle\langle 0| \otimes U_0 + |1\rangle\langle 1| \otimes U_1.$$

For example,

$$\text{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X. \quad \text{(III.17)}$$

In quantum circuit diagrams, the symbol for the CCNOT gate is show in Fig. III.10, or with $\bullet$ for top two connections and $\oplus$ for bottom, suggesting $\text{CCNOT}|x, y, z\rangle = |x, y, xy \oplus z\rangle$. Alternately, put "CCNOT" in a box. Other operations may be shown by putting a letter or symbol in a box, for example "H" for the Hadamard gate.

The Hadamard gate can be used to generate Bell states (Exer. III.39):

$$\text{CNOT}(H \otimes I)|xy\rangle = |\beta_{xy}\rangle. \quad \text{(III.18)}$$

In	Out
$\|00\rangle$	$(\|00\rangle + \|11\rangle)/\sqrt{2} \equiv \|\beta_{00}\rangle$
$\|01\rangle$	$(\|01\rangle + \|10\rangle)/\sqrt{2} \equiv \|\beta_{01}\rangle$
$\|10\rangle$	$(\|00\rangle - \|11\rangle)/\sqrt{2} \equiv \|\beta_{10}\rangle$
$\|11\rangle$	$(\|01\rangle - \|10\rangle)/\sqrt{2} \equiv \|\beta_{11}\rangle$

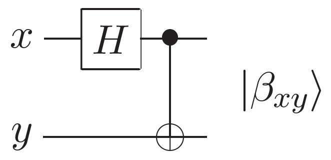


Figure III.13: Quantum circuit for generating Bell states. [from Nielsen & Chuang (2010, fig. 1.12)]

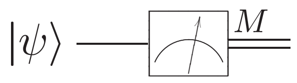


Figure III.14: Symbol for measurement of a quantum state (from Nielsen & Chuang (2010)).

The circuit for generating Bell states (Eq. III.18) is shown in Fig. III.13.

It's also convenient to have a symbol for quantum state measurement, such as Fig. III.14.

C.4 Quantum gate arrays

Fig. III.15 shows a quantum circuit for a 1-bit full adder. As we will see (Sec. C.7), it is possible to construct reversible quantum gates for any classically computable function. In particular the Fredkin and Toffoli gates are universal.

Because quantum computation is a unitary operator, it must be reversible. You know that an irreversible computation $x \mapsto f(x)$ can be embedded in a reversible computation $(x, c) \mapsto (g(x), f(x))$, where c are suitable ancillary constants and $g(x)$ represents the garbage qubits. Note that throwing away the garbage qubits (dumping them into the environment) will collapse the quantum state (equivalent to measurement) by entangling them in the many degrees of freedom of the environment. Typically these garbage qubits will be entangled with other qubits in the computation, collapsing them as well, and interfering with the computation. Therefore the garbage

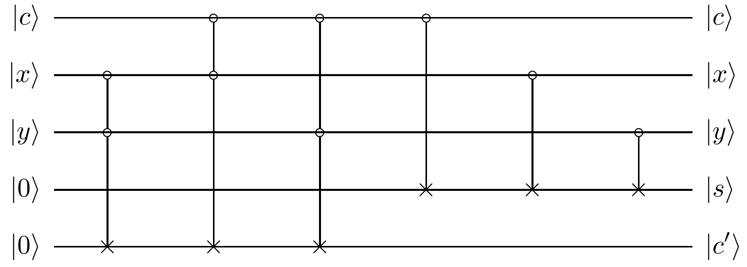


Figure III.15: Quantum circuit for 1-bit full adder [from Rieffel & Polak (2000)]. "x and y are the data bits, s is their sum (modulo 2), c is the incoming carry bit, and c' is the new carry bit."

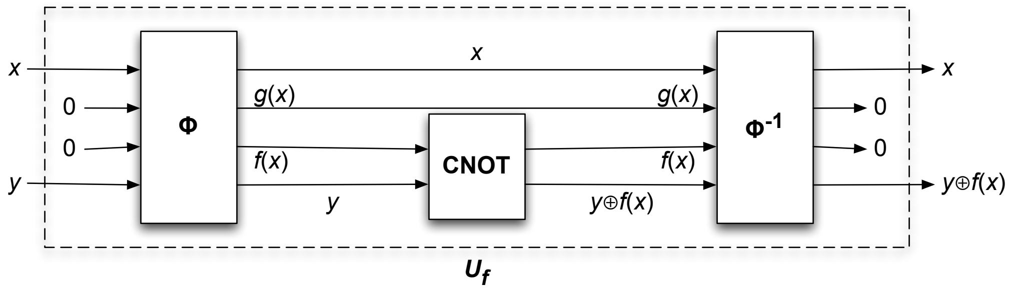


Figure III.16: Quantum gate array for reversible quantum computation.

must be produced in a *standard state* independent of x. This is accomplished by uncomputing, as we did in classical reversible computing (Ch. II, Sec. C.6, p. 58).

Since NOT is reversible, each 1 bit in c can be replaced by a 0 bit followed by a NOT, so we need only consider computations of the form $(x, 0) \mapsto (g(x), f(x))$; that is, all the constant bits can be zero.

Therefore, we begin by embedding our irreversible computation of f in a reversible computation Φ, which we get by providing 0 constants and generating garbage $g(x)$; see Fig. III.16. That is, Φ will perform the following computation on four registers (*data, workspace, result, target*):

$$(x, 0, 0, y) \mapsto (x, g(x), f(x), y).$$

The result $f(x)$ is in the result register and the garbage $g(x)$ is in the workspace register. Notice that x and y (data and target) are passed through. Now use CNOTs between corresponding places in the result and target registers to compute $y \oplus f(x)$, where $\oplus$ represents bitwise exclusive-or, in the target register. Thus we have computed:

$$(x, 0, 0, y) \mapsto (x, g(x), f(x), y \oplus f(x)).$$

Now we uncompute with Φ^{-1}, but since the data and target registers are passed through, we get $(x, 0, 0, y \oplus f(x))$ in the registers. We have restored the data, workspace, and result registers to their initial values and have $y \oplus f(x)$ in the target register. Ignoring the result and workspace registers, we write

$$(x, y) \mapsto (x, y \oplus f(x)).$$

This is the standard approach we will use for embedding a classical computation in a quantum computation.

Therefore, for any computable $f : \mathbf{2}^m \to \mathbf{2}^n$, there is a reversible *quantum gate array* $U_f : \mathcal{H}^{m+n} \to \mathcal{H}^{m+n}$ such that for $x \in \mathbf{2}^m$ and $y \in \mathbf{2}^n$,

$$U_f|x, y\rangle = |x, y \oplus f(x)\rangle,$$

See Fig. III.17. In particular, $U_f|x, \mathbf{0}\rangle = |x, f(x)\rangle$. The first m qubits are called the *data register* and the last n are called the *target register*.

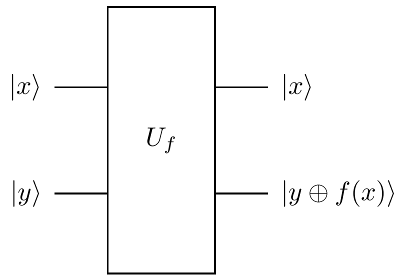


Figure III.17: Computation of function by quantum gate array (Rieffel & Polak, 2000).

C.5 Quantum parallelism

Since U_f is linear, if it is applied to a superposition of bit strings, it will produce a superposition of the results of applying f to them in parallel (i.e., in the same time it takes to compute it on one input):

$$U_f(c_1|\mathbf{x}_1\rangle + c_2|\mathbf{x}_2\rangle + \cdots + c_k|\mathbf{x}_k\rangle) = c_1U_f|\mathbf{x}_1\rangle + c_2U_f|\mathbf{x}_2\rangle + \cdots + c_kU_f|\mathbf{x}_k\rangle.$$

For example, if we have a superposition of the inputs $\mathbf{x}_1$ and $\mathbf{x}_2$,

$$U_f\left(\frac{\sqrt{3}}{2}|\mathbf{x}_1\rangle + \frac{1}{2}|\mathbf{x}_2\rangle\right) \otimes |\mathbf{0}\rangle = \frac{\sqrt{3}}{2}|\mathbf{x}_1, f(\mathbf{x}_1)\rangle + \frac{1}{2}|\mathbf{x}_2, f(\mathbf{x}_2)\rangle.$$

The amplitude of a result y will be the sum of the amplitudes of all x such that $y = f(x)$.

If we apply U_f to a superposition of all possible 2^m inputs, it will compute a superposition of all the corresponding outputs *in parallel* (i.e., in the same time as required for one function evaluation)! The Walsh-Hadamard transformation can be used to produce this superposition of all possible inputs:

$$\begin{aligned} W_m|00\ldots0\rangle &= \frac{1}{\sqrt{2^m}}(|00\ldots0\rangle + |00\ldots1\rangle + \cdots + |11\ldots1\rangle) \\ &= \frac{1}{\sqrt{2^m}}\sum_{\mathbf{x}\in\mathbf{2}^m}|\mathbf{x}\rangle \\ &= \frac{1}{\sqrt{2^m}}\sum_{x=0}^{2^m-1}|x\rangle. \end{aligned}$$

In the last line we are obviously interpreting the bit strings as natural numbers. Hence, for $f : \mathbf{2}^m \rightarrow \mathbf{2}^n$,

$$\begin{aligned} U_f(W_m|0\rangle^{\otimes m})|0\rangle^{\otimes n} &= U_f\left(\frac{1}{\sqrt{2^m}}\sum_{x=0}^{2^m-1}|x\rangle\right)|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^m}}\sum_{x=0}^{2^m-1}U_f|x,0^n\rangle \\ &= \frac{1}{\sqrt{2^m}}\sum_{x=0}^{2^m-1}|x,f(x)\rangle \end{aligned}$$

A single circuit does all 2^m computations simultaneously! "Note that since n qubits enable working simultaneously with 2^n states, quantum parallelism circumvents the time/space trade-off of classical parallelism through its ability to provide an exponential amount of computational space in a linear amount of physical space." (Rieffel & Polak, 2000)

This is amazing, but not immediately useful. If we measure the input bits, we will get a random value, and the state will be projected into a superposition of the outputs for the inputs we measured. If we measure an output bits, we will get a value probabilistically, and a superposition of all the inputs that can produce the measured output. Neither of the above is especially useful, so most quantum algorithms transform the state in such a way that the values of interest have a high probability of being measured. The other thing we can do is to extract common properties of all values of $f(x)$. Both of these require different programming techniques than classical computing.

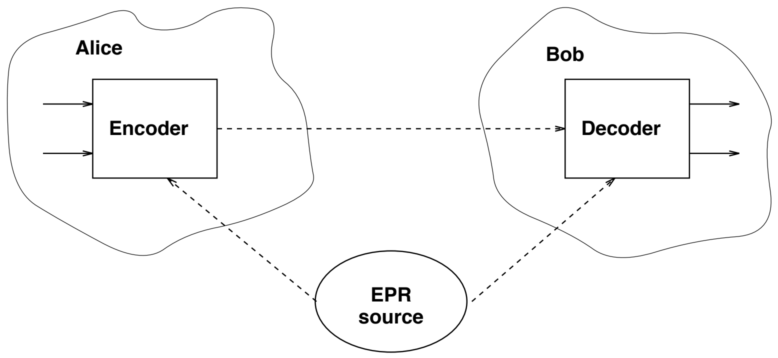


Figure III.18: Superdense coding. (Rieffel & Polak, 2000)

C.6 Applications

C.6.a Superdense coding

We will consider a couple simple applications of these ideas. The first is called *superdense coding* or (more modestly) *dense coding*, since it is a method by which one quantum particle can be used to transmit two classical bits of information. It was described by Bennett and Wiesner in 1992, and was partially validated experimentally by 1998.

Here is the idea. Alice and Bob share an entangled pair of qubits. To transmit two bits of information, Alice applies one of four transformations to her qubit. She then sends her qubit to Bob, who can apply an operation to the entangled pair to determine which of the four transformations she applied, and hence recover the two bits of information.

Now let's work it through more carefully. Suppose Alice and Bob share the entangled pair $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Since the four Bell states are a basis for the quantum state of the pair of qubits, Alice's two bits of information can be encoded as one of the four Bell states. For example, Alice can use the state $|\beta_{zx}\rangle$ to encode the bits z, x (the correspondence is arbitrary so long as we are consistent, but this one is easy to remember). Recall the circuit for generating Bell states (Fig. III.13, p. 113). Its effect is $\text{CNOT}(H \otimes I)|zx\rangle = |\beta_{zx}\rangle$. This cannot be used by Alice for generating the Bell states, because she doesn't have access to Bob's qubit. However, the Bell states differ from each other only in the relative parity and phase of their component qubits (i.e., whether they have the same or opposite bit values and the same or opposite signs). Therefore, Alice can alter the parity and phase of just her qubit to transform the entangled pair into any of the Bell states. In particular, if she uses zx to select I, X, Z, or $ZX = Y$ (corresponding to $zx = 00, 01, 10, 11$ respectively) and applies it to just her qubit, she can generate the corresponding Bell state $|\beta_{zx}\rangle$. I've picked this correspondence because of the simple relation between the bits z, x and the application of the operators Z, X, but this is not necessary; any other 1-1 correspondence between the two bits and the four operators could be used. When Alice applies this transformation to her qubit, Bob's qubit is unaffected, and so the transformation on the entangled pair is $I \otimes I$, $X \otimes I$, $Z \otimes I$, or $ZX \otimes I$. We can check the results as follows:

bits	transformation	result	
00	$I \otimes I$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	$= \|\beta_{00}\rangle$
01	$X \otimes I$	$\frac{1}{\sqrt{2}}(\|10\rangle + \|01\rangle)$	$= \|\beta_{01}\rangle$
10	$Z \otimes I$	$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	$= \|\beta_{10}\rangle$
11	$ZX \otimes I$	$\frac{1}{\sqrt{2}}(-\|10\rangle + \|01\rangle)$	$= \|\beta_{11}\rangle$

For example, in the second-to-last case, since $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$, we see $Z \otimes I \left[\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\right] = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$. Make sure you can explain the results in the other cases (Exer. III.44).

When Alice wants to send her information, she applies the appropriate operator to her qubit and sends her single transformed qubit to Bob, which he uses with his qubit to recover the information by measuring the pair of qubits in the Bell basis. This can be done by inverting the Bell state generator, which, since the CNOT and H are self-adjoint, is simply:

$$(H \otimes I)\text{CNOT}|\beta_{zx}\rangle = |zx\rangle.$$

This translates the Bell basis into the computational basis, so Bob can measure the bits exactly.

C.6.b Quantum teleportation

Quantum teleportation is not quite as exciting as it sounds! Its goal is to transfer the exact quantum state of a particle from Alice to Bob by means a classical channel (Figs. III.19, III.20). Of course, the No Cloning Theorem says we cannot copy a quantum state, but we can "teleport" it by destroying

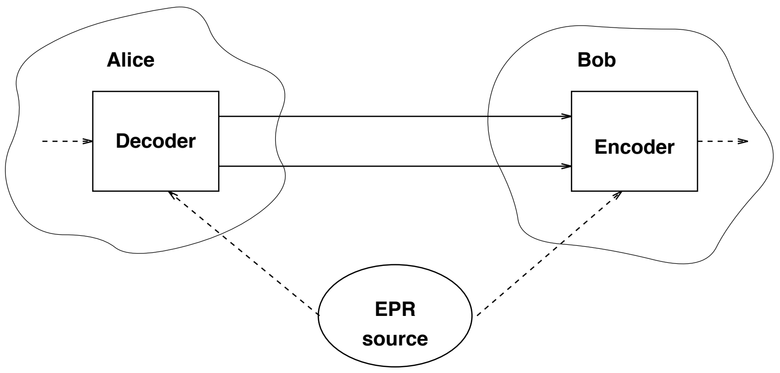


Figure III.19: Quantum teleportation. (Rieffel & Polak, 2000)

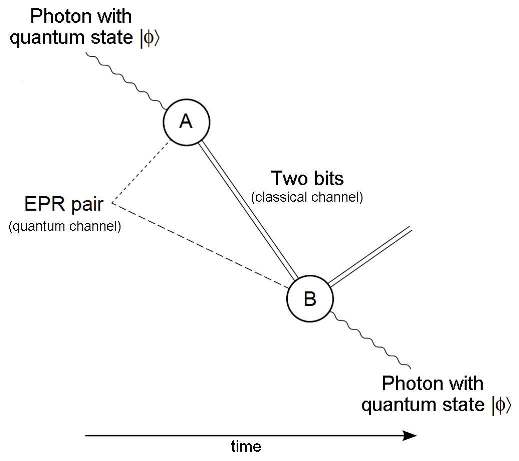


Figure III.20: Possible setup for quantum teleportation. [from wikipedia commons]

the original and recreating it elsewhere. Single-qubit quantum teleportation was described by Bennett in 1993 and first demonstrated experimentally in the late 1990s.

This is how it works. Alice and Bob begin by sharing the halves of an entangled pair, $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Suppose that the quantum state that Alice wants to share is $|\psi\rangle = a|0\rangle + b|1\rangle$. The composite system comprising the unknown state and the Bell state is

$$\begin{aligned}
|\psi_0\rangle &\stackrel{\text{def}}{=} |\psi, \beta_{00}\rangle \\
&= (a|0\rangle + b|1\rangle)\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\
&= \frac{1}{\sqrt{2}}[a|0\rangle(|00\rangle + |11\rangle) + b|1\rangle(|00\rangle + |11\rangle)] \\
&= \frac{1}{\sqrt{2}}(a|0,00\rangle + a|0,11\rangle + b|1,00\rangle + b|1,11\rangle).
\end{aligned}$$

Alice applies the decoding circuit used for superdense coding to the unknown state and her qubit from the entangled pair. This function is $(H \otimes I)$CNOT; it measures her two qubits in the Bell basis. When Alice applies CNOT to her two qubits (leaving Bob's qubit alone) the resulting composite state is:

$$\begin{aligned}
|\psi_1\rangle &\stackrel{\text{def}}{=} (\text{CNOT} \otimes I)|\psi_0\rangle \\
&= (\text{CNOT} \otimes I)\left[\frac{1}{\sqrt{2}}(a|00,0\rangle + a|01,1\rangle + b|10,0\rangle + b|11,1\rangle)\right] \\
&= \frac{1}{\sqrt{2}}(a|00,0\rangle + a|01,1\rangle + b|11,0\rangle + b|10,1\rangle).
\end{aligned}$$

Notice that the amplitude a of $|\psi\rangle$ has been transferred to the components of the shared pair having the same parity ($|00\rangle$ and $|11\rangle$), whereas the amplitude b has been transferred to the components having the opposite parity ($|10\rangle$ and $|01\rangle$). When Alice applies $H \otimes I$ to her qubits the result is:

$$\begin{aligned}
|\psi_2\rangle &\stackrel{\text{def}}{=} (H \otimes I \otimes I)|\psi_1\rangle \\
&= (H \otimes I \otimes I)\frac{1}{\sqrt{2}}(a|0,00\rangle + a|0,11\rangle + b|1,10\rangle + b|1,01\rangle) \\
&= \frac{1}{2}\left[a(|0,00\rangle + |1,00\rangle + |0,11\rangle + |1,11\rangle)\right. \\
&\quad \left. + b(|0,10\rangle - |1,10\rangle + |0,01\rangle - |1,01\rangle)\right].
\end{aligned}$$

This is because $H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Rearranging and factoring to separate Alice's qubits from Bob's, we have:

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{2}[|00\rangle(a|0\rangle + b|1\rangle) + |01\rangle(a|1\rangle + b|0\rangle) \\ &\quad +|10\rangle(a|0\rangle - b|1\rangle) + |11\rangle(a|1\rangle - b|0\rangle)]\,. \end{aligned}$$

Thus the unknown amplitudes have been transferred from the first qubit (Alice's) to the third (Bob's), which now incorporates the amplitudes a and b, but in different ways depending on the first two qubits. In fact you can see that the amplitudes are transformed by the Pauli matrices, and Bob can restore the quantum state by applying the correct Pauli matrix. Therefore Alice measures the first two bits (completing measurement in the Bell basis) and sends them to Bob over the classical channel. This measurement partially collapses the state, which includes Bob's qubit, but in a way that is determined by the first two qubits.

When Bob receives the two classical bits from Alice, he uses them to select a transformation for his qubit, which restores the amplitudes to the correct basis vectors. These transformations are the Pauli matrices (which are their own inverses):

bits	gate	input	
00	I	$a\|0\rangle + b\|1\rangle$	(identity)
01	X	$a\|1\rangle + b\|0\rangle$	(exchange)
10	Z	$a\|0\rangle - b\|1\rangle$	(flip)
11	ZX	$a\|1\rangle - b\|0\rangle$	(exchange–flip)

In each case, applying the specified gate to its input yields $|\psi\rangle = a|0\rangle + b|1\rangle$, Alice's original quantum state. This is obvious in the 00 case, but you should verify the others (Exer. III.45). Notice that since Alice had to measure her qubits, the original quantum state of her particle has collapsed. Thus it has been "teleported," not copied.

The quantum circuit in Fig. III.21 is slightly different from what we've described, since it uses the fact that the appropriate transformations can be expressed in the form $Z^{M_1}X^{M_2}$, where M_1 and M_2 are the two classical bits. You should verify that $ZX = Y$ (Exer. III.46).

Both superdense coding and teleportation indicate that with an entangled pair, two bits can be interchanged with one qubit. This is one example of a method of *interchanging resources.* However, quantum teleportation does

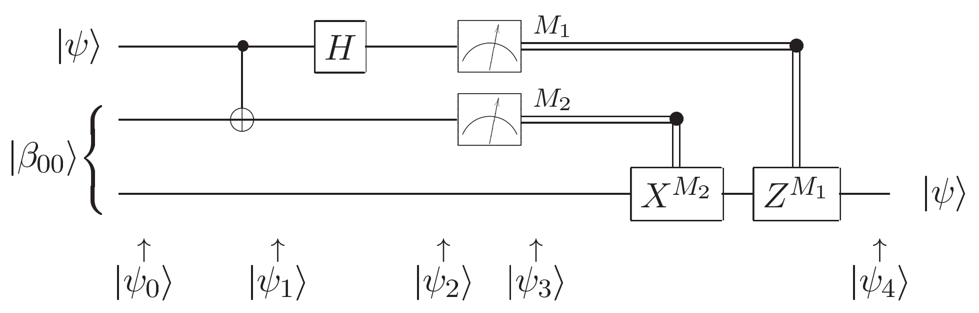


Figure III.21: Circuit for quantum teleportation. [from Nielsen & Chuang (2010)]

not allow faster-than-light communication, since Alice has to transmit her two classical bits to Bob.

Entangled states can be teleported in a similar way. Free-space quantum teleportation has been demonstrated over 143 km between two of the Canary Islands (*Nature*, 13 Sept. 2012).[7] In Sept. 2015 teleportation was achieved over 101 km through supercooled nanowire. For teleporting material systems, the current record is 21 m.

C.7 Universal quantum gates

We have seen several interesting examples of quantum computing using gates such as CNOT and the Hadamard and Pauli operators.[8] Since the implementation of each of these is a technical challenge, it raises the important question: What gates are sufficient for implementing *any* quantum computation?

Both the Fredkin (controlled swap) and Toffoli (controlled-controlled-NOT) gates are sufficient for classical logic circuits. In fact, they can operate as well on qubits in superposition. But what about other quantum operators?

It can be proved that single-qubit unitary operators can be approximated arbitrarily closely by the Hadamard gate and the T ($\pi/8$) gate, which is

[7]`http://www.nature.com/nature/journal/v489/n7415/full/nature11472.html` (accessed 12-09-18).

[8]This lecture follows Nielsen & Chuang (2010, §4.5).

defined:

$$T = \left(\begin{array}{cc} 1 & 0 \\ 0 & e^{i\pi/4} \end{array} \right) \cong \left(\begin{array}{cc} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{array} \right) \tag{III.19}$$

(ignoring global phase). To approximate within ϵ any single-qubit unitary operation, you need $\mathcal{O}(\log^c(1/\epsilon))$ gates, where $c \approx 2$. For an m-gate circuit (of CNOTs and single-qubit unitaries) and an accuracy of ϵ, $\mathcal{O}(m \log^c(m/\epsilon))$, where $c \approx 2$, gates are needed (Solovay-Kitaev theorem).

A *two-level operation* is a unitary operator on a d-dimensional Hilbert space that non-trivially affects only two qubits out of n (where $d = 2^n$). It can be proved that any two-level unitary operation can be computed by a combination of CNOTs and single-qubit operations. This requires $\mathcal{O}(n^2)$ single-qubit and CNOT gates.

It also can be proved that an arbitrary d-dimensional unitary matrix can be decomposed into a product of two-level unitary matrices. At most $d(d-1)/2$ of them are required. Therefore a unitary operator on an n-qubit system requires at most $2^{n-1}(2^n - 1)$ two-level matrices.

In conclusion, the H (Hadamard), CNOT, and $\pi/8$ gates are sufficient for quantum computation. For fault-tolerance, either the *standard set* — H (Hadamard), CNOT, $\pi/8$, and S (phase) — can be used, or H, CNOT, Toffoli, and S. The latter *phase gate* is defined:

$$S = T^2 = \left(\begin{array}{cc} 1 & 0 \\ 0 & i \end{array} \right). \tag{III.20}$$

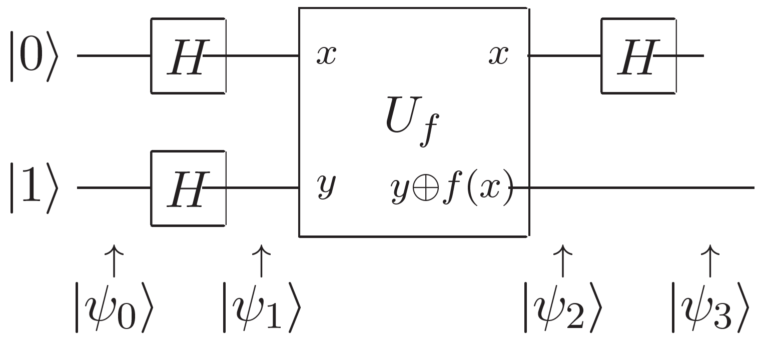


Figure III.22: Quantum circuit for Deutsch algorithm. [fig. from Nielsen & Chuang (2010)]

D Quantum algorithms

D.1 Deutsch-Jozsa algorithm

D.1.a Deutsch's algorithm

In this section you will encounter your first example of a quantum algorithm that can compute faster than a classical algorithm for the same problem. This is a simplified version of Deutsch's original algorithm, which shows how it is possible to extract global information about a function by using quantum parallelism and interference (Fig. III.22).[9]

Suppose we have a function $f : \mathbf{2} \rightarrow \mathbf{2}$, as in Sec. C.5. The goal is to determine whether $f(0) = f(1)$ with a *single* function evaluation. This is not a very interesting problem (since there are only four such functions), but it is a warmup for the Deutsch-Jozsa algorithm. Simple as it is, it could be expensive to decide on a classical computer. For example, suppose $f(0) =$ the billionth bit of π and $f(1) =$ the billionth bit of e. Then the problem is to decide if the billionth bits of π and e are the same. It is mathematically simple, but computationally complex.

To see how we might solve this problem, suppose we have a quantum gate array U_f for f; that is, $U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$. In particular, $U_f|x\rangle|0\rangle =$

[9]This is the 1998 improvement by Cleve et al. to Deutsch's 1985 algorithm (Nielsen & Chuang, 2010, p. 59).

$|x\rangle|f(x)\rangle$ and $U_f|x\rangle|1\rangle = |x\rangle|\neg f(x)\rangle$. Usually we set $y = 0$ to get the result $|f(x)\rangle$, but here you will see an application in which we want $y = 1$.

Now consider the result of applying U_f to $|x\rangle$ in the data register and to the superposition $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ in the target register.

$$U_f|x\rangle|-\rangle = \frac{1}{\sqrt{2}}|x\rangle|f(x)\rangle - \frac{1}{\sqrt{2}}|x\rangle|\neg f(x)\rangle = \frac{1}{\sqrt{2}}|x\rangle[|f(x)\rangle - |\neg f(x)\rangle].$$

Now the rightmost square bracket is $|0\rangle - |1\rangle$ if $f(x) = 0$ or $|1\rangle - |0\rangle$ if $f(x) = 1$. Therefore, we can write

$$U_f|x\rangle|-\rangle = \frac{1}{\sqrt{2}}|x\rangle(-)^{f(x)}(|0\rangle - |1\rangle) = (-)^{f(x)}|x\rangle|-\rangle. \qquad \text{(III.21)}$$

[Here, $(-)^x$ is an abbreviation for $(-1)^x$ when we want to emphasize that the sign is all that matters.] Since $U_f|x\rangle|-\rangle = (-)^{f(x)}|x\rangle|-\rangle$, the result of applying it to an equal superposition of $x = 0$ and $x = 1$ is:

$$\frac{1}{\sqrt{2}}\sum_{x\in\mathbf{2}} U_f|x\rangle|-\rangle = \frac{1}{\sqrt{2}}\sum_{x\in\mathbf{2}}(-)^{f(x)}|x\rangle|-\rangle.$$

If f is a constant function, then $f(0) = f(1)$, and the summation is $\pm\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|-\rangle = \pm|+\rangle|-\rangle$ because both components have the same sign.. On the other hand, if $f(0) \neq f(1)$, then the summation is $\pm\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|-\rangle = \pm|-\rangle|-\rangle$ because the components have opposite signs. That is, a constant function gives the $|0\rangle$ and $|1\rangle$ components of the data qubit the same phase, and otherwise gives them the opposite phase. Therefore, we can determine whether the function is constant or not by measuring the first qubit in the sign basis; we get $|+\rangle$ if $f(0) = f(1)$ and $|-\rangle$ otherwise. With this background, we can state Deutsch's algorithm.

algorithm Deutsch:

Initial state: Begin with the qubits $|\psi_0\rangle \overset{\text{def}}{=} |01\rangle$.

Superposition: Transform it to a pair of superpositions

$$|\psi_1\rangle \overset{\text{def}}{=} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |+-\rangle. \qquad \text{(III.22)}$$

by a pair of Hadamard gates. Recall that $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$.

Function application: Next apply U_f to $|\psi_1\rangle = |+-\rangle$. As we've seen, $U_f|x\rangle|0\rangle = |x\rangle|0 \oplus f(x)\rangle = |x\rangle|f(x)\rangle$, and $U_f|x\rangle|1\rangle = |x\rangle|1 \oplus f(x)\rangle = |x\rangle|\neg f(x)\rangle$. Therefore, expand Eq. III.22 and apply U_f:

$$\begin{aligned}
|\psi_2\rangle &\stackrel{\text{def}}{=} U_f|\psi_1\rangle \\
&= U_f\left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right] \\
&= \frac{1}{2}[U_f|00\rangle - U_f|01\rangle + U_f|10\rangle - U_f|11\rangle] \\
&= \frac{1}{2}[|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(1)\rangle - |1, \neg f(1)\rangle]
\end{aligned}$$

There are two cases: $f(0) = f(1)$ and $f(0) \neq f(1)$.

Equal (constant function): If $f(0) = f(1)$, then

$$\begin{aligned}
|\psi_2\rangle &= \frac{1}{2}[|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, f(0)\rangle - |1, \neg f(0)\rangle] \\
&= \frac{1}{2}[|0\rangle(|f(0)\rangle - |\neg f(0)\rangle) + |1\rangle(|f(0)\rangle - |\neg f(0)\rangle)] \\
&= \frac{1}{2}(|0\rangle + |1\rangle)(|f(0)\rangle - |\neg f(0)\rangle) \\
&= \pm\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) \\
&= \pm\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|-\rangle \\
&= |+-\rangle.
\end{aligned}$$

The last line applies because global phase (including $\pm$) doesn't matter.

Unequal (balanced function): If $f(0) \neq f(1)$, then

$$|\psi_2\rangle = \frac{1}{2}[|0, f(0)\rangle - |0, \neg f(0)\rangle + |1, \neg f(0)\rangle - |1, f(0)\rangle]$$

$$\begin{aligned}
&= \frac{1}{2}[|0\rangle(|f(0)\rangle - |\neg f(0)\rangle) + |1\rangle(|\neg f(0)\rangle - |f(0)\rangle)] \\
&= \frac{1}{2}[|0\rangle(|f(0)\rangle - |\neg f(0)\rangle) - |1\rangle(|f(0)\rangle - |\neg f(0)\rangle)] \\
&= \frac{1}{2}(|0\rangle - |1\rangle)(|f(0)\rangle - |\neg f(0)\rangle) \\
&= \pm\frac{1}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle) \\
&= \pm\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|-\rangle \\
&= |--\rangle
\end{aligned}$$

Clearly we can discriminate between the two cases by measuring the first qubit in the sign basis. In particular, note that in the unequal case, the $|1\rangle$ component has the opposite phase from the $|0\rangle$ component.

Measurement: Therefore we can determine whether $f(0) = f(1)$ or not by measuring the first bit of $|\psi_2\rangle$ in the sign basis, which we can do with the Hadamard gate (recall $H|+\rangle = |0\rangle$ and $H|-\rangle = |1\rangle$):

$$\begin{aligned}
|\psi_3\rangle &\overset{\text{def}}{=} (H \otimes I)|\psi_2\rangle \\
&= \begin{cases} \pm|0\rangle|-\rangle, & \text{if } f(0) = f(1) \\ \pm|1\rangle|-\rangle, & \text{if } f(0) \neq f(1) \end{cases} \\
&= \pm|f(0) \oplus f(1)\rangle|-\rangle.
\end{aligned}$$

□

Notice that the information we need is in the *data* register, not the target register. This technique is called *phase kick-back* (i.e., kicked back into the phase of the data register).

In conclusion, we can determine whether or not $f(0) = f(1)$ with a *single evaluation* of f, which is quite remarkable. In effect, we are evaluating f on a superposition of $|0\rangle$ and $|1\rangle$ and determining how the results interfere with each other. As a result we get a definite (not probabilistic) determination of a global property with a single evaluation.

This is a clear example where a quantum computer can do something faster than a classical computer. However, note that U_f has to uncompute

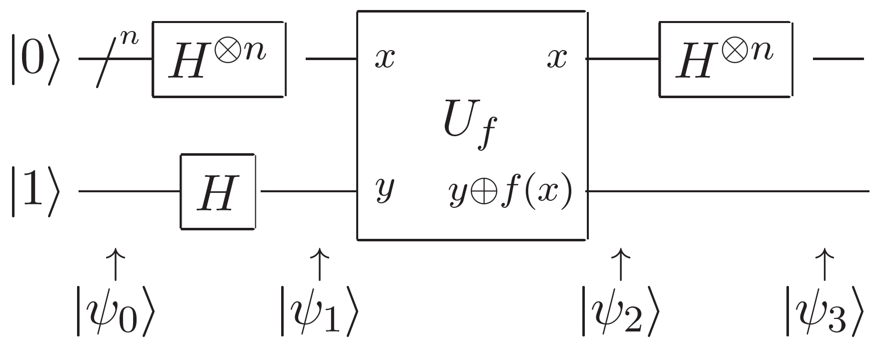


Figure III.23: Quantum circuit for Deutsch-Jozsa algorithm. [fig. from NC]

f, which takes as much time as computing it, but we will see other cases (Deutsch-Jozsa) where the speedup is much more than $2\times$.

D.1.b The Deutsch-Jozsa algorithm

The Deutsch-Jozsa algorithm is a generalization of the Deutsch algorithm to n bits, which they published it in 1992; here we present the improved version of Nielsen & Chuang (2010, p. 59).

This is the problem: Suppose we are given an unknown function $f : \mathbf{2}^n \to \mathbf{2}$ in the form of a unitary transform $U_f \in \mathcal{L}(\mathcal{H}^{n+1}, \mathcal{H})$ (Fig. III.23). We are told only that f is either constant or *balanced*, which means that it is 0 on half its domain and 1 on the other half. Our task is to determine into which class the given f falls.

Consider first the classical situation. We can try different input bit strings $\mathbf{x}$. We might (if we're lucky) discover after the second query of f that it is not constant. But we might require as many as $2^n/2 + 1$ queries to answer the question. So we're facing $\mathcal{O}(2^{n-1})$ function evaluations.

algorithm Deutsch-Jozsa:

Initial state: As in the Deutsch algorithm, prepare the initial state $|\psi_0\rangle \overset{\text{def}}{=} |0\rangle^{\otimes n}|1\rangle$.

Superposition: Use the Walsh-Hadamard transformation to create a superposition of all possible inputs:

$$|\psi_1\rangle \stackrel{\text{def}}{=} (H^{\otimes n} \otimes H)|\psi_0\rangle = \sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{\sqrt{2^n}}|\mathbf{x},-\rangle.$$

Claim: Similarly to the single qubit case (Eq. III.21), we can see that $U_f|\mathbf{x},-\rangle = (-)^{f(\mathbf{x})}|\mathbf{x}\rangle|-\rangle$, where $(-)^n$ is an abbreviation for $(-1)^n$. From the definition of $|-\rangle$ and U_f, $U_f|\mathbf{x},-\rangle = |\mathbf{x}\rangle\frac{1}{\sqrt{2}}(|f(\mathbf{x})\rangle - |\neg f(\mathbf{x})\rangle)$. Since $f(\mathbf{x}) \in \mathbf{2}$, $\frac{1}{\sqrt{2}}(|f(\mathbf{x})\rangle - |\neg f(\mathbf{x})\rangle) = |-\rangle$ if $f(\mathbf{x}) = 0$, and it $= -|-\rangle$ if $f(\mathbf{x}) = 1$. This establishes the claim.

Function application: Therefore, you can see that:

$$|\psi_2\rangle \stackrel{\text{def}}{=} U_f|\psi_1\rangle = \sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{\sqrt{2^n}}(-)^{f(\mathbf{x})}|\mathbf{x}\rangle|-\rangle. \tag{III.23}$$

In the case of a constant function, all the components of the data state have the same phase, otherwise they do not.

To see how we can make use of this information, let's consider the state in more detail. For a *single* bit you can show (Exer. III.47):

$$H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-)^x|1\rangle) = \frac{1}{\sqrt{2}}\sum_{z\in\mathbf{2}}(-)^{xz}|z\rangle = \sum_{z\in\mathbf{2}}\frac{1}{\sqrt{2}}(-)^{xz}|z\rangle.$$

(This is just another way of writing $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.) Therefore, the general formula for the Walsh transform of n bits is:

$$\begin{aligned} H^{\otimes n}|x_1, x_2, \ldots, x_n\rangle &= \frac{1}{\sqrt{2^n}} \sum_{z_1,\ldots,z_n\in\mathbf{2}} (-)^{x_1 z_1+\cdots+x_n z_n}|z_1, z_2, \ldots, z_n\rangle \\ &= \frac{1}{\sqrt{2^n}}\sum_{\mathbf{z}\in\mathbf{2}^n}(-)^{\mathbf{x}\cdot\mathbf{z}}|\mathbf{z}\rangle, \end{aligned} \tag{III.24}$$

where $\mathbf{x}\cdot\mathbf{z}$ is the bitwise inner product. (It doesn't matter if you do addition or $\oplus$ since only the parity of the result is significant.) *Remember this formula!* Combining this and the result in Eq. III.23,

$$|\psi_3\rangle \stackrel{\text{def}}{=} (H^{\otimes n} \otimes I)|\psi_2\rangle = \sum_{\mathbf{z}\in\mathbf{2}^n}\sum_{\mathbf{x}\in\mathbf{2}^n}\frac{1}{2^n}(-)^{\mathbf{x}\cdot\mathbf{z}+f(\mathbf{x})}|\mathbf{z}\rangle|-\rangle.$$

Measurement: Consider the first n qubits and the amplitude of one particular basis state, $\mathbf{z} = |\mathbf{0}\rangle = |0\rangle^{\otimes n}$, which we separate from the rest of the summation:

$$|\psi_3\rangle = \sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{2^n}(-)^{f(\mathbf{x})}|\mathbf{0}\rangle|-\rangle + \sum_{\mathbf{z}\in\mathbf{2}^n-\{\mathbf{0}\}} \sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{2^n}(-)^{\mathbf{x}\cdot\mathbf{z}+f(\mathbf{x})}|\mathbf{z}\rangle|-\rangle$$

Hence, the amplitude of $|0\rangle^{\otimes n}$, the all-$|0\rangle$ qubit string, is $\sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{2^n}(-)^{f(\mathbf{x})}$. Recall how in the basic Deutsch algorithm we got a sum of signs (either all the same or not) for all the function evaluations. The result is similar here, but we have 2^n values rather than just two. We now have two cases:

Constant function: If the function is constant, then all the exponents of -1 will be the same (either all 0 or all 1), and so the amplitude will be ± 1. Therefore all the other amplitudes are 0 and any measurement must yield 0 for all the qubits (since only $|0\rangle^{\otimes n}$ has nonzero amplitude).

Balanced function: If the function is not constant then (*ex hypothesi*) it is balanced, but more specifically, if it is balanced, then there must be an equal number of $+1$ and -1 contributions to the amplitude of $|0\rangle^{\otimes n}$, so its amplitude is 0. Therefore, when we measure the state, at least one qubit must be nonzero (since the all-0s state has amplitude 0).
$\square$

The *good news* about the Deutsch-Jozsa algorithm is that with one quantum function evaluation we have got a result that would require between 2 and $\mathcal{O}(2^{n-1})$ classical function evaluations (exponential speedup!). The *bad news* is that the algorithm has no known applications! Even if it were useful, however, the problem could be solved probabilistically on a classical computer with only a few evaluations of f: for an error probability of ϵ, it takes $\mathcal{O}(\log \epsilon^{-1})$ function evaluations. However, it illustrates principles of quantum computing that can be used in more useful algorithms.

D.2 Simon's algorithm

Simon's algorithm was first presented in 1994 and can be found in Simon, D. (1997), "On the power of quantum computation," *SIAM Journ. Computing*, 26 (5), pp. 1474–83.[10] For breaking RSA we will see that its useful to know the *period* of a function: that r such that $f(x+r) = f(x)$. Simon's problem is a warmup for this.

Simon's Problem: Suppose we are given an unknown function $f : \mathbf{2}^n \to \mathbf{2}^n$ and we are told that it is *two-to-one*. This means $f(\mathbf{x}) = f(\mathbf{y})$ iff $\mathbf{x} \oplus \mathbf{y} = \mathbf{r}$ for some fixed $\mathbf{r} \in \mathbf{2}^n$. The vector $\mathbf{r}$ can be considered the *period* of f, since $f(\mathbf{x} \oplus \mathbf{r}) = f(\mathbf{x})$. The problem is to determine the period $\mathbf{r}$ of a given unknown f.

Consider first the classical solution. Since we don't know anything about f, the best we can do is evaluate it on random inputs. If we are ever lucky enough to find $\mathbf{x}$ and $\mathbf{x}'$ such that $f(\mathbf{x}) = f(\mathbf{x}')$, then we have our answer, $\mathbf{r} = \mathbf{x} \oplus \mathbf{x}'$. After testing m values, you will have eliminated about $m(m-1)/2$ possible $\mathbf{r}$ vectors (namely, $\mathbf{x} \oplus \mathbf{x}'$ for every pair of these m vectors). You will be done when $m^2 \approx 2^n$. Therefore, on the average you need to do $2^{n/2}$ function evaluations, which is exponential in the size of the input. For $n = 100$, it would require about $2^{50} \approx 10^{15}$ evaluations. "At 10 million calls per second it would take about three years" (Mermin, 2007, p. 55). We will see that a quantum computer can determine $\mathbf{r}$ with high probability ($> 1 - 10^{-6}$) in about 120 evaluations. At 10 million calls per second, this would take about 12 microseconds!

algorithm Simon:

Input superposition: As before, start by using the Walsh-Hadamard transform to create a superposition of all possible inputs:

$$|\psi_1\rangle \stackrel{\text{def}}{=} H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{2^{n/2}} \sum_{\mathbf{x} \in \mathbf{2}^n} |\mathbf{x}\rangle.$$

[10]The following presentation follows Mermin's *Quantum Computer Science* (Mermin, 2007, §2.5, pp. 55–8).

Function evaluation: Suppose that U_f is the quantum gate array implementing f and recall $U_f|\mathbf{x}\rangle|\mathbf{y}\rangle = |\mathbf{x}\rangle|\mathbf{y} \oplus f(\mathbf{x})\rangle$. Therefore:

$$|\psi_2\rangle \stackrel{\text{def}}{=} U_f|\psi_1\rangle|0\rangle^{\otimes n} = \frac{1}{2^{n/2}} \sum_{\mathbf{x}\in\mathbf{2}^n} |\mathbf{x}\rangle|f(\mathbf{x})\rangle.$$

Therefore we have an equal superposition of corresponding input-output values.

Output measurement: Measure the output register (in the computational basis) to obtain some $|\mathbf{z}\rangle$. Since the function is two-to-one, the projection will have a superposition of two inputs:

$$\frac{1}{\sqrt{2}}(|\mathbf{x}_0\rangle + |\mathbf{x}_0 \oplus \mathbf{r}\rangle)|\mathbf{z}\rangle,$$

where $f(\mathbf{x}_0) = \mathbf{z} = f(\mathbf{x}_0 \oplus \mathbf{r})$. The information we need is contained in the input register,

$$|\psi_3\rangle \stackrel{\text{def}}{=} \frac{1}{\sqrt{2}}(|\mathbf{x}_0\rangle + |\mathbf{x}_0 \oplus \mathbf{r}\rangle),$$

but it cannot be extracted directly. If we measure it, we will get either $\mathbf{x}_0$ or $\mathbf{x}_0 \oplus \mathbf{r}$, but not both, and we need both to get $\mathbf{r}$. (We cannot make two copies, due to the no-cloning theorem.)

Suppose we apply the Walsh-Hadamard transform to this superposition:

$$\begin{aligned} H^{\otimes n}|\psi_3\rangle &= H^{\otimes n}\frac{1}{\sqrt{2}}(|\mathbf{x}_0\rangle + |\mathbf{x}_0 \oplus \mathbf{r}\rangle) \\ &= \frac{1}{\sqrt{2}}(H^{\otimes n}|\mathbf{x}_0\rangle + H^{\otimes n}|\mathbf{x}_0 \oplus \mathbf{r}\rangle). \end{aligned}$$

Now, recall (D.1.b, p. 130) that

$$H^{\otimes n}|\mathbf{x}\rangle = \frac{1}{2^{n/2}} \sum_{\mathbf{y}\in\mathbf{2}^n} (-1)^{\mathbf{x}\cdot\mathbf{y}}|\mathbf{y}\rangle.$$

(This is the general expression for the Walsh transform of a bit string. The phase depends on the number of common 1-bits.) Therefore,

$$\begin{aligned} H^{\otimes n}|\psi_3\rangle &= \frac{1}{\sqrt{2}}\left[\frac{1}{2^{n/2}} \sum_{\mathbf{y}\in\mathbf{2}^n} (-1)^{\mathbf{x}_0\cdot\mathbf{y}}|\mathbf{y}\rangle + \frac{1}{2^{n/2}} \sum_{\mathbf{y}\in\mathbf{2}^n} (-1)^{(\mathbf{x}_0+\mathbf{r})\cdot\mathbf{y}}|\mathbf{y}\rangle\right] \\ &= \frac{1}{2^{(n+1)/2}} \sum_{\mathbf{y}\in\mathbf{2}^n} \left[(-1)^{\mathbf{x}_0\cdot\mathbf{y}} + (-1)^{(\mathbf{x}_0+\mathbf{r})\cdot\mathbf{y}}\right]|\mathbf{y}\rangle. \end{aligned}$$

Note that
$$(-1)^{(\mathbf{x}_0+\mathbf{r})\cdot\mathbf{y}} = (-1)^{\mathbf{x}_0\cdot\mathbf{y}}(-1)^{\mathbf{r}\cdot\mathbf{y}}.$$
Therefore, if $\mathbf{r}\cdot\mathbf{y} = 1$, then the bracketed expression is 0 (since the terms have opposite sign and cancel). However, if $\mathbf{r}\cdot\mathbf{y} = 0$, then the bracketed expression is $2(-1)^{\mathbf{x}_0\cdot\mathbf{y}}$ (since they don't cancel). Hence the result of the Walsh-Hadamard transform is
$$|\psi_4\rangle = H^{\otimes n}|\psi_3\rangle = \frac{1}{2^{(n-1)/2}} \sum_{\mathbf{y}\text{ s.t. }\mathbf{r}\cdot\mathbf{y}=0} (-1)^{\mathbf{x}_0\cdot\mathbf{y}}|\mathbf{y}\rangle.$$

Measurement: Measuring the input register (in the computational basis) will collapse it with equal probability into a state $|\mathbf{y}^{(1)}\rangle$ such that $\mathbf{r}\cdot\mathbf{y}^{(1)} = 0$.

First equation: Since we know $\mathbf{y}^{(1)}$, this gives us some information about $\mathbf{r}$, expressed in the equation:
$$y_1^{(1)}r_1 + y_2^{(1)}r_2 + \cdots + y_n^{(1)}r_n = 0 \pmod 2.$$

Iteration: The quantum computation can be repeated, producing a series of bit strings $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots$ such that $\mathbf{y}^{(k)}\cdot\mathbf{r} = 0$. From them we can build up a system of n linearly-independent equations and solve for $\mathbf{r}$. (If you get a linearly-dependent equation, you have to try again.) Note that each quantum step (involving one evaluation of f) produces an equation (except in the unlikely case $\mathbf{y}^{(k)} = 0$ or that it's linearly dependent), and therefore determines one of the bits in terms of the other bits. That is, each iteration reduced the candidates for $\mathbf{r}$ by approximately one-half.
$\square$

A mathematical analysis (Mermin, 2007, App. G) shows that with $n+m$ iterations the probability of having enough information to determine $\mathbf{r}$ is $> 1 - \frac{1}{2^{m+1}}$. "Thus the odds are more than a million to one that with $n + 20$ invocations of $\mathbf{U}_f$ we will learn $[\mathbf{r}]$, no matter how large n may be."

(Mermin, 2007, p. 57) Note that the "extra" evaluations are independent of n. Therefore Simon's problem can be solved in *linear* time on a quantum computer, but requires *exponential* time on a classical computer.

D.3 Shor's algorithm

If computers that you build are quantum,
Then spies everywhere will all want 'em.
Our codes will all fail,
And they'll read our email,
Till we get crypto that's quantum, and daunt 'em.
— Jennifer and Peter Shor (Nielsen & Chuang, 2010, p. 216)

The widely used RSA public-key cryptography system is based on the difficulty of factoring large numbers.[11] The best classical algorithms are nearly exponential in the size of the input, $m = \ln M$. Specifically, the best current (2006) algorithm (the *number field sieve algorithm*) runs in time $e^{\mathcal{O}(m^{1/3} \ln^{2/3} m)}$. This is subexponential but very inefficient. Shor's quantum algorithm is bounded error-probability quantum polynomial time (BQP), specifically, $\mathcal{O}(m^3)$. Shor's algorithm was invented in 1994, inspired by Simon's algorithm.

Shor's algorithm reduces factoring to finding the period of a function. The connection between factoring and period finding can be understood as follows. Assume you are trying to factor M. Suppose you can find x such that $x^2 = 1 \pmod{M}$. Then $x^2 - 1 = 0 \pmod{M}$. Therefore $(x+1)(x-1) = 0 \pmod{M}$. Therefore both $x+1$ and $x-1$ have common factors with M (except in the trivial case $x = 1$, and so long as neither is a multiple of M). Now pick an a that is coprime (relatively prime) to M. If $a^r = 1 \pmod{M}$ and r happens to be even, we're done (since we can find a factor of M as explained above). (The smallest such r is called the *order* of a.) This r is the period of $a^x \pmod{M}$, since $a^{x+r} = a^x a^r = a^x \pmod{M}$.

In summary, if we can find the order of an appropriate a and it is even, then we can probably factor the number. To accomplish this, we need to find the period of $a^x \pmod{M}$, which can be determined through a Fourier transform.

Like the classical Fourier transform, the *quantum Fourier transform* puts all the amplitude of the function into multiples of the frequency (reciprocal period). Therefore, measuring the state yields the period with high probability.

[11]These section is based primarily on Rieffel & Polak (2000).

D.3.a Quantum Fourier transform

Before explaining Shor's algorithm, it's necessary to explain the quantum Fourier transform, and to do so it's helpful to begin with a review of the classical Fourier transform.

Let f be a function defined on $[0, 2\pi)$. We know it can be represented as a Fourier series,

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos(kx + \phi_k),$$

where $k = 0, 1, 2, \ldots$ represents the overtone series (natural number multiples of the fundamental frequency). You know also that the Fourier transform can be represented in the *ciscoid* (cosine + i sine) basis, where we define $u_k(x) \overset{\text{def}}{=} \text{cis}(-kx) = e^{-ikx}$. (The "$-$" sign is irrelevant, but will be convenient later.) The u_k are orthogonal but not normalized, so we divide them by 2π, since $\int_0^{2\pi} \cos^2 x + \sin^2 x \; \mathrm{d}x = 2\pi$. The Fourier series in this basis is $f(x) = \sum_{k=-\infty}^{\infty} \hat{f}_k \,\text{cis}(-kx)$. The Fourier coefficients are given by $\hat{f}_k = \frac{1}{2\pi}\langle u_k \mid f\rangle = \frac{1}{2\pi}\int_0^{2\pi} e^{ikx} f(x)\mathrm{d}x$. They give the amplitude and phase of the component signals u_k.

For the *discrete Fourier transform* (DFT) we suppose that f is represented by N samples, $f_j \overset{\text{def}}{=} f(x_j)$, where $x_j = 2\pi\frac{j}{N}$, with $j \in \mathbf{N} \overset{\text{def}}{=} \{0, 1, \ldots, N-1\}$. Let $\mathbf{f} = (f_0, \ldots, f_{N-1})^{\mathrm{T}}$. Note that the x_j are the $1/N$ segments of a circle. (Realistically, N is big.)

Likewise each of the basis functions is represented by a vector of N samples: $\mathbf{u}_k \overset{\text{def}}{=} (u_{k,0}, \ldots, u_{k,N-1})^{\mathrm{T}}$. Thus we have a matrix of all the basis samples:

$$u_{kj} \overset{\text{def}}{=} \text{cis}(-kx_j) = e^{-2\pi ikj/N},\; j \in \mathbf{N}.$$

In $e^{-2\pi ikj/N}$, note that $2\pi i$ represents a full cycle, k is the overtone, and j/N represents the fraction of a full cycle.

Recall that every complex number has N principal N^{th}-roots, and in particular the number 1 (unity) has N principal N^{th}-roots. Notice that N samples of the fundamental period correspond to the N principal N^{th}-roots of unity, that is, ω^j where (for a particular N) $\omega = e^{2\pi i/N}$. Hence, $u_{kj} = \omega^{-kj}$. That is, $\mathbf{u}_k = (\omega^{-k\cdot 0}, \omega^{-k\cdot 1} \ldots, \omega^{-k\cdot(N-1)})^{\mathrm{T}}$. It is easy to show that the vectors $\mathbf{u}_k$ are orthogonal, and in fact that $\mathbf{u}_k/\sqrt{N}$ are ON (exercise). Therefore, $\mathbf{f}$

can be represented by a Fourier series,

$$\mathbf{f} = \frac{1}{\sqrt{N}} \sum_{k \in \mathbf{N}} \hat{f}_k \mathbf{u}_k = \frac{1}{N} \sum_{k \in \mathbf{N}} (\mathbf{u}_k^\dagger \mathbf{f}) \mathbf{u}_k.$$

Define the *discrete Fourier transform* of the vector $\mathbf{f}$, $\hat{\mathbf{f}} = \mathrm{F}\mathbf{f}$, to be the vector of Fourier coefficients, $\hat{f}_k = \mathbf{u}_k^\dagger \mathbf{f} / \sqrt{N}$. Determine F as follows:

$$\hat{\mathbf{f}} = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} \mathbf{u}_0^\dagger \mathbf{f} \\ \mathbf{u}_1^\dagger \mathbf{f} \\ \vdots \\ \mathbf{u}_{N-1}^\dagger \mathbf{f} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} \mathbf{u}_0^\dagger \\ \mathbf{u}_1^\dagger \\ \vdots \\ \mathbf{u}_{N-1}^\dagger \end{pmatrix} \mathbf{f}.$$

Therefore let

$$\mathrm{F} \stackrel{\text{def}}{=} \frac{1}{\sqrt{N}} \begin{pmatrix} \mathbf{u}_0^\dagger \\ \mathbf{u}_1^\dagger \\ \vdots \\ \mathbf{u}_{N-1}^\dagger \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} \omega^{0 \cdot 0} & \omega^{0 \cdot 1} & \cdots & \omega^{0 \cdot (N-1)} \\ \omega^{1 \cdot 0} & \omega^{1 \cdot 1} & \cdots & \omega^{1 \cdot (N-1)} \\ \omega^{2 \cdot 0} & \omega^{2 \cdot 1} & \cdots & \omega^{2 \cdot (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{(N-1) \cdot 0} & \omega^{(N-1) \cdot 1} & \cdots & \omega^{(N-1) \cdot (N-1)} \end{pmatrix}. \tag{III.25}$$

That is, $\mathrm{F}_{kj} = \overline{u_{kj}} / \sqrt{N} = \omega^{kj} / \sqrt{N}$ for $k, j \in \mathbf{N}$. Note that the "$-$" signs in the complex exponentials were eliminated by the conjugate transpose. F is unitary transformation (exercise).

The *fast Fourier transform* (FFT) reduces the number of operations required from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.[12] It does this with a recursive algorithm that avoids recomputing values. However, it is restricted to powers of two, $N = 2^n$.

The *quantum Fourier transform* (QFT) is even faster, $\mathcal{O}(\log^2 N)$, that is, $\mathcal{O}(n^2)$. However, because the spectrum is encoded in the amplitudes of the quantum state, we cannot get them all. Like the FFT, the QFT is restricted to powers of two, $N = 2^n$. The QFT transforms the amplitudes of a quantum state:

$$U_{\text{QFT}} \sum_{j \in \mathbf{N}} f_j |j\rangle = \sum_{k \in \mathbf{N}} \hat{f}_k |k\rangle,$$

[12]The FFT is $\mathcal{O}(N \log N)$, but $N > M^2 = e^{2m}$. Therefore the FFT is $\mathcal{O}(M^2 \log M^2) = \mathcal{O}(M^2 \log M) = \mathcal{O}(me^{2m})$

where $\hat{\mathbf{f}} \stackrel{\text{def}}{=} \mathrm{F}\mathbf{f}$.

Suppose f has period r, and suppose that the period evenly divides the number of samples, $r \mid N$. Then all the amplitude of $\hat{f}$ should be at multiples of its fundamental frequency, N/r. If $r \not\mid N$, then the amplitude will be concentrated *near* multiples of N/r. The approximation is improved by using larger n.

The FFT (and QFT) are implemented in terms of additions and multiplications by various roots of unity (powers of ω). In QFT, these are phase shifts. In fact, the QFT can be implemented with $n(n+1)/2$ gates of two types: (1) One is H_j, the Hadamard transformation of the jth qubit. (2) The other is a controlled phase-shift $S_{j,k}$, which uses qubit x_j to control whether it does a particular phase shift on the $|1\rangle$ component of qubit x_k. That is, $S_{j,k}|x_j x_k\rangle \mapsto |x_j x'_k\rangle$ is defined by

$$S_{j,k} \stackrel{\text{def}}{=} |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| + e^{i\theta_{k-j}}|11\rangle\langle 11|,$$

where $\theta_{k-j} = \pi/2^{k-j}$. That is, the phase shift depends on the indices j and k.

It can be shown that the QFT can be defined:[13]

$$U_{\text{QFT}} = \prod_{j=0}^{n-1} H_j \prod_{k=j+1}^{n-1} S_{j,k}.$$

This is $\mathcal{O}(n^2)$ gates.

D.3.b Shor's algorithm step by step

Shor's algorithm depends on many results from number theory, which are outside of the scope of this course. Since this is not a course in cryptography or number theory, I will just illustrate the ideas.

algorithm Shor:

[13]See Rieffel & Polak (2000) for this, with a detailed explanation in Nielsen & Chuang (2010, §5.1, pp. 517–21).

Input: Suppose we are factoring M (and $M = 21$ will be used for concrete examples, but of course the goal is to factor very large numbers). Let $m \overset{\text{def}}{=} \lceil \lg M \rceil = 5$ in the case $M = 21$.

Step 1: Pick a random number $a < M$. If a and M are not coprime (relatively prime), we are done. (Euclid's algorithm is $\mathcal{O}(m^2) = \mathcal{O}(\log^2 M)$.) For our example, suppose we pick $a = 11$, which is relatively prime with 21.

Modular exponentiation: Let $g(x) \overset{\text{def}}{=} a^x \pmod{M}$, for $x \in \mathbf{M} \overset{\text{def}}{=} \{0, 1, \ldots, M-1\}$. This takes $\mathcal{O}(m^3)$ gates and is the most complex part of the algorithm! (Reversible circuits typically use m^3 gates for m qubits.) In our example case, $g(x) = 11^x \pmod{21}$, so

$$g(x) = \underbrace{1, 11, 16, 8, 4, 2,}_{\text{period}} 1, 11, 16, 8, 4, \ldots$$

In order to get a good QFT approximation, pick n such that $M^2 \leq 2^n < 2M^2$. Let $N \overset{\text{def}}{=} 2^n$. Equivalently, pick n such that $2 \lg M \leq n < 2 \lg M + 1$, that is, roughly twice as many qubits as in M. Note that although the number of samples is $N = 2^n$, we need only n qubits (thanks to the tensor product). This is where quantum computation gets its speedup over classical computation; M is very large, so $N > M^2$ is extremely large. The QFT computes all these in parallel. For our example $M = 21$, and so we pick $n = 9$ for $N = 512$ since $441 \leq 512 < 882$. Therefore $m = 5$.

Step 2 (quantum parallelism): Apply U_g to the superposition

$$|\psi_0\rangle \overset{\text{def}}{=} H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x \in \mathbf{N}} |x\rangle$$

to get

$$|\psi_1\rangle \overset{\text{def}}{=} U_g|\psi_0\rangle|0\rangle^{\otimes m} = \frac{1}{\sqrt{N}} \sum_{x \in \mathbf{N}} |x, g(x)\rangle.$$

For our example problem, 14 qubits are required [$n = 9$ for x and $m = 5$ for $g(x)$]. The quantum state looks like this (note the periodicity):

$$|\psi_1\rangle = \frac{1}{\sqrt{512}} \quad (\quad |0,1\rangle + |1,11\rangle + |2,16\rangle + |3,8\rangle + |4,4\rangle + |5,2\rangle +$$

$$|6,1\rangle + |7,11\rangle + |8,16\rangle + |9,8\rangle + |10,4\rangle + |11,2\rangle + \cdots)$$

Step 3 (measurement): The function g has a period r, which we want to transfer to the amplitudes of the state so that we can apply the QFT. This is accomplished by measuring (and discarding) the result register (as in Simon's algorithm).[14] Suppose the result register collapses into state g^* (e.g., $g^* = 8$). The input register will collapse into a superposition of all x such that $g(x) = g^*$. We can write it:

$$|\psi_2\rangle \stackrel{\text{def}}{=} \frac{1}{\mathcal{Z}} \sum_{x \in \mathbf{N} \text{ s.t. } g(x)=g^*} |x, g^*\rangle = \frac{1}{\mathcal{Z}} \sum_{x \in \mathbf{N}} f_x |x, g^*\rangle = \left[\frac{1}{\mathcal{Z}} \sum_{x \in \mathbf{N}} f_x |x\rangle \right] |g^*\rangle,$$

where

$$f_x \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } g(x) = g^* \\ 0, & \text{otherwise} \end{cases},$$

and $\mathcal{Z} \stackrel{\text{def}}{=} \sqrt{|\{x \mid g(x) = g^*\}|}$ is a normalization factor. For example,

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\mathcal{Z}}(|3,8\rangle + |9,8\rangle + |15,8\rangle + \cdots) \\ &= \frac{1}{\mathcal{Z}}(|3\rangle + |9\rangle + |15\rangle + \cdots)|8\rangle \end{aligned}$$

Note that the values x for which $f_x \neq 0$ differ from each other by the period; This produces a function f of very strong periodicity. As in Simon's algorithm, if we could measure two such x, we would have useful information, but we can't. Suppose we measure the result register and get $g^* = 8$. Fig. III.24 shows the corresponding $\mathbf{f} = (0,0,0,0,1,0,0,0,0,0,1,0,\ldots)$.

Step 4 (QFT): Apply the QFT to obtain,

$$\begin{aligned} |\psi_3\rangle &\stackrel{\text{def}}{=} U_{\text{QFT}} \left(\frac{1}{\mathcal{Z}} \sum_{x \in \mathbf{N}} f_x |x\rangle \right) \\ &= \frac{1}{\mathcal{Z}} \sum_{\hat{x} \in \mathbf{N}} \hat{f}_{\hat{x}} |\hat{x}\rangle. \end{aligned}$$

[14]As it turns out, this measurement of the result register can be avoided. This is in general true for "internal" measurement processes in quantum algorithms (Bernstein & Vazirani 1997).

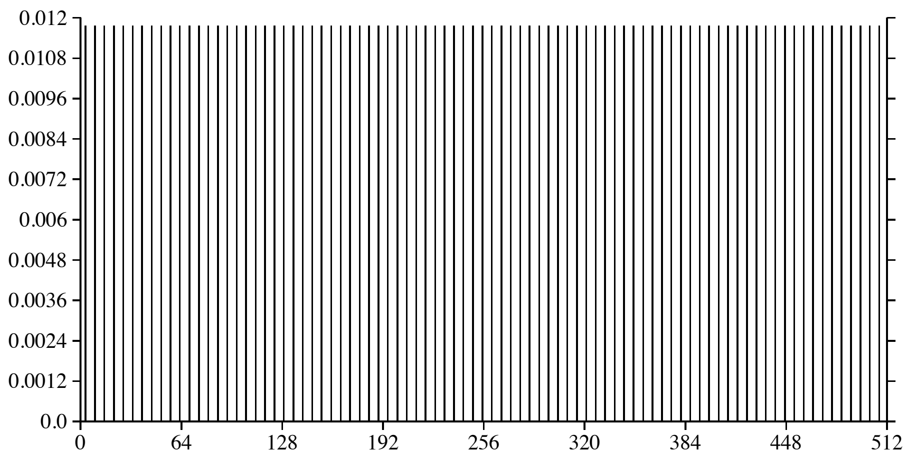


Figure III.24: Example probability distribution $|f_x|^2$ for state $Z^{-1} \sum_{x \in \mathbf{N}} f_x |x, 8\rangle$. In this example the period is $r = 6$ (e.g., at $x = 3, 9, 15, \ldots$). [fig. source: Rieffel & Polak (2000)]

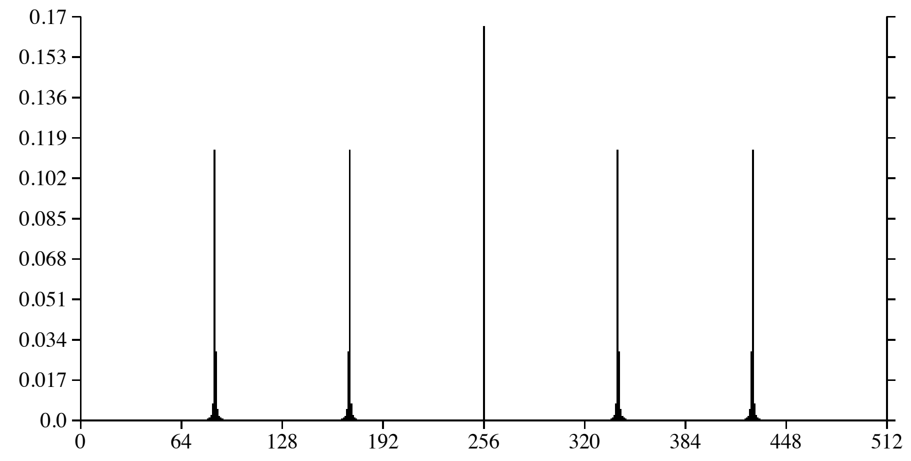


Figure III.25: Example probability distribution $|\hat{f}_{\hat{x}}|^2$ of the quantum Fourier transform of $\mathbf{f}$. The spectrum is concentrated near multiples of $N/6 = 512/6 = 85\ 1/3$, that is $85\ 1/3, 170\ 2/3, 256$, etc. [fig. source: Rieffel & Polak (2000)]

(The collapsed result register $|g^*\rangle$ has been omitted.)

If the period r divides $N = 2^n$, then $\hat{\mathbf{f}}$ will be nonzero only at multiples of the fundamental frequency N/r. That is, the nonzero components will be $|kN/r\rangle$. If it doesn't divide evenly, then the amplitude will be concentrated around these $|kN/r\rangle$. See Fig. III.24 and Fig. III.25 for examples of the probability distributions $|f_x|^2$ and $|\hat{f}_{\hat{x}}|^2$.

Step 5 (period extraction): Measure the state in the computational basis.

Period a power of 2: If $r \mid N$, then the resulting state will be $v \overset{\text{def}}{=} |kN/r\rangle$ for some $k \in \mathbf{N}$. Therefore $k/r = v/N$. If k and r are relatively prime, as is likely, then reducing the fraction v/N to lowest terms will produce r in the denominator. In this case the period is discovered.

Period not a power of 2: In the case r does not divide N, it's often possible to guess the period from a continued fraction expansion of v/N.[15] If v/N is sufficiently close to p/q, then a continued fraction expansion of v/N will contain the continued fraction expansion of p/q. For example, suppose the measurement returns $v = 427$, which is not a power of two. This is the result of the continued fraction expansion of v/N (in this case, 427/512) (see IQC):

i	a_i	p_i	q_i	ϵ_i
0	0	0	1	0.8339844
1	1	1	1	0.1990632
2	5	5	6	0.02352941
3	42	211	253	0.5

"which terminates with $6 = q_2 < M \leq q_3$. Thus, $q = 6$ is likely to be the period of f." [IQC]

Step 6 (finding a factor): The following computation applies however we

[15]See Rieffel & Polak (2000, App. B) for an explanation of this procedure and citations for why it works.

got the period q in Step 5. If the guess q is even, then $a^{q/2}+1$ and $a^{q/2}-1$ are likely to have common factors with M. Use the Euclidean algorithm to check this. The reason is as follows. If q is the period of $g(x) = a^x \pmod M$, then $a^q = 1 \pmod M$. This is because, if q is the period, then for all x, $g(x+q) = g(x)$, that is, $a^{q+x} = a^q a^x = a^x \pmod M$ for all x. Therefore $a^q - 1 = 0 \pmod M$. Hence,

$$(a^{q/2}+1)(a^{q/2}-1) = 0 \pmod M.$$

Therefore, unless one of the factors is a multiple of M (and hence $= 0$ mod M), one of them has a nontrivial common factor with M.

In the case of our example, the continued fraction gave us a guess $q = 6$, so with $a = 11$ we should consider $11^3+1 = 1332$ and $11^3 - 1 = 1330$. For $M = 21$ the Euclidean algorithm yields $\gcd(21, 1332) = 3$ and $\gcd(21, 1330) = 7$. We've factored 21!

Iteration: There are several reasons that the preceding steps might not have succeeded: (1) The value v projected from the spectrum might not be close enough to a multiple of N/r (D.3.b). (2) In D.3.b, k and r might not be relatively prime, so that the denominator is only a factor of the period, but not the period itself. (3) In D.3.b, one of the two factors turns out to be a multiple of M. (4) In D.3.b, q was odd. In these cases, a few repetitions of the preceding steps yields a factor of M.
□

D.3.c Recent progress

To read our E-mail, how mean
of the spies and their quantum machine;
be comforted though,
they do not yet know
how to factorize twelve or fifteen.
— Volker Strassen (Nielsen & Chuang, 2010, p. 216)

In this section we review recent progress in hardware implementation of

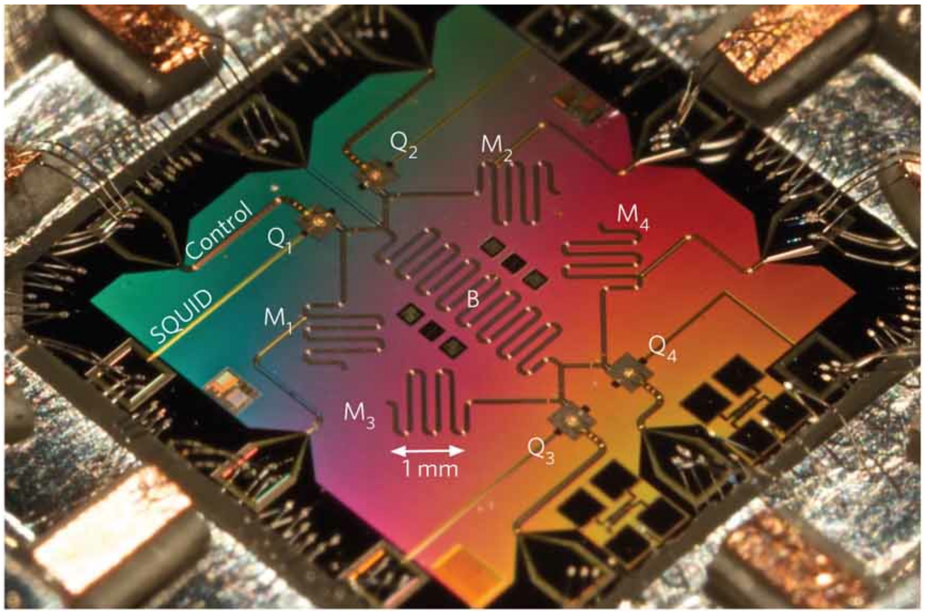


Figure III.26: Hardware implementation of Shor's algorithm developed at UCSB (2012). The M_j are quantum memory elements, B is a quantum "bus," and the Q_j are phase qubits that can be used to implement qubit operations between the bus and memory elements. [source: CPF]

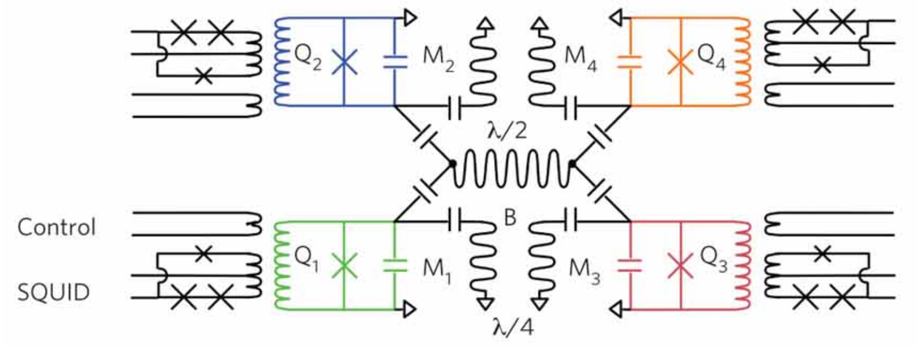


Figure III.27: Circuit of hardware implementation of Shor's algorithm developed at UCSB. [source: CPF]

Shor's algorithm.[16] In Aug. 2012 a group at UC Santa Barbara described a quantum implementation of Shor's algorithm that correctly factored 15 about 48% of the time (50% being the theoretical success rate). (There have been NMR hardware factorizations of 15 since 2001, but there is some doubt if entanglement was involved.) This is a 3-qubit compiled version of Shor's algorithm, where "compiled" means that the implementation of modular exponentiation is for fixed M and a. This compiled version used $a = 4$ as the coprime to $M = 15$. In this case the correct period $r = 2$. The device (Fig. III.26) has nine quantum devices, including four phase qubits and five superconducing co-planar waveguide (CPW) microwave resonators. The four CPWs (M_j) can be used as memory elements and fifth (B) can be used as a "bus" to mediate entangling operations. In effect the qubits Q_j can be read and written. Radio frequency pulses in the bias coil can be used to adjust the qubit's frequency, and gigahertz pulses can be used to manipulate and measure the qubit's state. SQUIDs are used for one-shot readout of the qubits. The qubits Q_j can be tuned into resonance with the bus B or memory elements M_j. The quantum processor can be used to implement the single-qubit gates X, Y, Z, H, and the two-qubit swap (iSWAP) and controlled-phase (C_ϕ) gates. The entanglement protocol can be scaled to an arbitrary number of qubits. The relaxation and dephasing times are about 200ns.

Another group has reported the quantum factoring of 21.[17] Their procedure operates by using one qubit instead of the n qubits in the (upper) control qubits. It does this by doing all the unitaries associated with the lowest-order control qubit, then for the next control qubit, updating the work register after each step, for n interations.

[16]This section is based primarily on Erik Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O'Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland & John M. Martinis, "Computing prime factors with a Josephson phase qubit quantum processor." *Nature Physics* **8**, 719–723 (2012) `doi:10.1038/nphys2385` [CPF].

[17]See Martin-Lópex et al., "Experimental realization of Shor's quantum factoring algorithm using qubit recycling," *Nature Photonics* 6, 773–6 (2012).

D.4 Search problems

D.4.a Overview

Many problems can be formulated as search problems over a solution space $\mathcal{S}$.[18] That is, find the $x \in \mathcal{S}$ such that some predicate $P(x)$ is true. For example, hard problems such as the Hamiltonian path problem and Boolean satisfiability can be formulated this way. An *unstructured search problem* is a problem that makes no assumptions about the *structure* of the search space, or for which there is no known way to make use of it (also called a *needle in a haystack problem*). That is, information about a particular value $P(x_0)$ does not give us usable information about another value $P(x_1)$. In contrast, a *structured search problem* is a problem in which the structure of the solution space can be used to guide the search, for example, searching an alphabetized array. In general, unstructured search takes $\mathcal{O}(M)$ evaluations, where $M = |\mathcal{S}|$ is the size of the solution space (which is often exponential in the size of the problem). On the average it will be $M/2$ (think of searching an unordered array) to find a solution with 50% probability.

We will see that Grover's algorithm can do unstructured search on a quantum computer with bounded probability in $\mathcal{O}(\sqrt{M})$ time, that is, quadratic speedup. This is provably more efficient than any algorithm on a classical computer, which is good (but not great). Unfortunately, it has been proved that Grover's algorithm is optimal for unstructured search. Therefore, to do better requires exploiting the structure of the solution space. *Quantum computers do not exempt us from understanding the problems we are trying to solve!* Shor's algorithm is an excellent example of exploiting the structure of a problem domain. Later we will take a look at *heuristic quantum search algorithms* that do make use of problem structure.

D.4.b Grover's Algorithm

algorithm Grover:

Input: Let M be the size of the solution space and pick n such that $2^n \geq M$.

[18]This section is based primarily on Rieffel & Polak (2000).

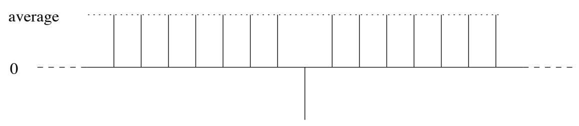


Figure III.28: Depiction of the result of phase rotation (changing the sign) of solutions in Grover's algorithm. [source: Rieffel & Polak (2000)]

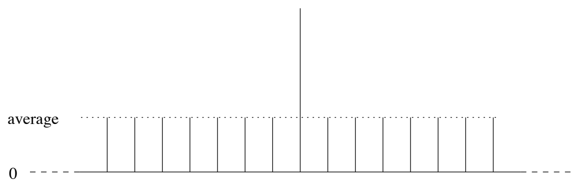


Figure III.29: Depiction of result of inversion about the mean in Grover's algorithm. [source: Rieffel & Polak (2000)]

Let $N \overset{\text{def}}{=} 2^n$ and let $\mathbf{N} \overset{\text{def}}{=} \mathbf{2}^n = \{0, 1, \ldots, N-1\}$, the set of n-bit strings. We are given a computable predicate $P : \mathbf{N} \to \mathbf{2}$. Suppose we have a quantum gate array U_P (an *oracle*) that computes the predicate:

$$U_P|x, y\rangle = |x, y \oplus P(x)\rangle.$$

Application: Consider what happens if, as usual, we apply the function to a superposition of all possible inputs $|\psi_0\rangle$:

$$U_P|\psi_0\rangle|0\rangle = U_P\left[\frac{1}{\sqrt{N}}\sum_{x\in\mathbf{N}}|x,0\rangle\right] = \frac{1}{\sqrt{N}}\sum_{x\in\mathbf{N}}|x, P(x)\rangle.$$

Notice that the components we want, $|x, 1\rangle$, and the components we don't want, $|x, 0\rangle$, all have the same amplitude, $\frac{1}{\sqrt{N}}$. So if we measure the state, the chances of getting a hit are very small, $\mathcal{O}(2^{-n})$. The trick, therefore, is to amplify the components that we want at the expense of the ones we don't want; this is what Grover's algorithm accomplishes.

Sign-change: To do this, first we change the sign of every solution (a phase rotation of π). That is, if the state is $\sum_j a_j|x_j, P(x_j)\rangle$, then we want to change a_j to $-a_j$ whenever $P(x_j) = 1$. See Fig. III.28. I'll get to the technique in a moment.

Inversion about mean: Next, we invert all the components around their mean amplitude (which is a little less than the amplitudes of the non-solutions); the result is shown in Fig. III.29. As a result of this operation, amplitudes of non-solutions go from a little above the mean to a little below it, but amplitudes of solutions go from far below the mean to far above it. This amplifies the solutions.

Iteration: This *Grover iteration* (the sign change and inversion about the mean) is repeated $\frac{\pi\sqrt{N}}{4}$ times. Thus the algorithm is $\mathcal{O}(\sqrt{N})$.

Measurement: Measurement yields an x_0 for which $P(x_0) = 1$ with high probability. Specifically, if there is exactly one solution $x_0 \in \mathcal{S}$, then $\frac{\pi\sqrt{N}}{8}$ iterations will yield it with probability 1/2. With $\frac{\pi\sqrt{N}}{4}$ iterations, the probability of failure drops to $1/N = 2^{-n}$. Unlike with most classical algorithms, additional iterations will give a *worse* result! This is because Grover iterations are unitary rotations, and so excessive rotations can rotate past the solution. Therefore it is critical to know when to stop. Fortunately there is a quantum technique (Brassard et al. 1998) for determining the optimal stopping point. Grover's iteration can be used for a wide variety of problems as a part of other quantum algorithms.
$\square$

In the following geometric analysis, I will suppose that there is just one answer α such that $P(\alpha) = 1$; then $|\alpha\rangle$ is the desired answer vector. Let $|\omega\rangle$ be a uniform superposition of all the other (non-answer) states, and observe that $|\alpha\rangle$ and $|\omega\rangle$ are orthonormal. Therefore, initially the state is $|\psi_0\rangle = \frac{1}{\sqrt{N}}|\alpha\rangle + \sqrt{\frac{N-1}{N}}|\omega\rangle$. In general, after k iterations the state is $|\psi_k\rangle =$

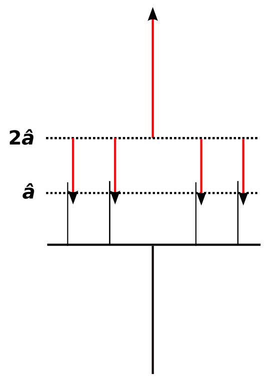


Figure III.30: Process of inversion about the mean in Grover's algorithm. The black lines represent the original amplitudes a_j. The red lines represent $2\bar{a} - a_j$, with the arrow heads indicating the new amplitudes a'_j.

$a|\alpha\rangle + w|\omega\rangle$, for some a, w with $|a|^2 + |w|^2 = 1$.

The sign change operation transforms the state as follows:

$$|\psi_k\rangle = a|\alpha\rangle + w|\omega\rangle \quad \mapsto \quad -a|\alpha\rangle + w|\omega\rangle = |\psi'_k\rangle,$$

where I've called the result $|\psi'_k\rangle$. This is a reflection across the $|\omega\rangle$ vector, which means that it will be useful to look at reflections more generally.

Suppose that $|\phi\rangle$ and $|\phi^\perp\rangle$ are orthonormal vectors and that $|\psi\rangle = a|\phi^\perp\rangle + b|\phi\rangle$ is an arbitrary vector in the space they span (Fig. III.31). The reflection of $|\psi\rangle$ across $|\phi\rangle$ is $|\psi'\rangle = -a|\phi^\perp\rangle + b|\phi\rangle$. Since $a = \langle\phi^\perp \mid \psi\rangle$ and $b = \langle\phi \mid \psi\rangle$, we know $|\psi\rangle = |\phi\rangle\langle\phi \mid \psi\rangle + |\phi^\perp\rangle\langle\phi^\perp \mid \psi\rangle$, and you can see that $|\psi'\rangle = |\phi\rangle\langle\phi \mid \psi\rangle - |\phi^\perp\rangle\langle\phi^\perp \mid \psi\rangle$. Hence the operator to reflect across $|\phi\rangle$ is $R_\phi \overset{\text{def}}{=} |\phi\rangle\langle\phi| - |\phi^\perp\rangle\langle\phi^\perp|$. Alternate forms of this operator are $2|\phi\rangle\langle\phi| - I$ and $I - 2|\phi^\perp\rangle\langle\phi^\perp|$, that is, subtract twice the perpendicular component.

The sign change can be expressed as a reflection:

$$R_\omega = |\omega\rangle\langle\omega| - |\alpha\rangle\langle\alpha| = I - 2|\alpha\rangle\langle\alpha|,$$

which expresses the sign-change of the answer vector clearly. Of course we don't know $|\alpha\rangle$, which is why we will have to use a different process to accomplish this reflection (see p. 152). We also will see that the inversion about the mean is equivalent to reflecting that state vector across $|\psi_0\rangle$.

But first, taking this for granted, let's see the effect of the Grover iteration (Fig. III.32). Let θ be the angle between $|\psi_0\rangle$ and $|\omega\rangle$. It's given by the inner

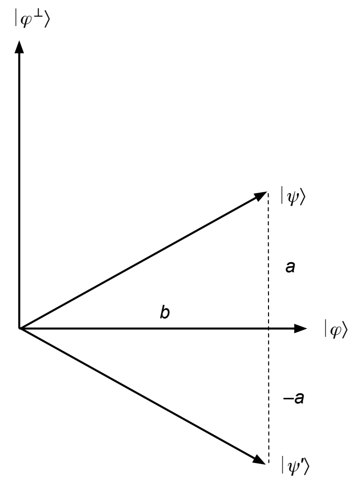


Figure III.31: Reflection across arbitrary vector. Reflection of $|\psi\rangle$ across $|\phi\rangle$ in plane with $|\phi^\perp\rangle$. $|\psi\rangle = a|\phi^\perp\rangle + b|\phi\rangle$ becomes $|\psi'\rangle = -a|\phi^\perp\rangle + b|\phi\rangle$.

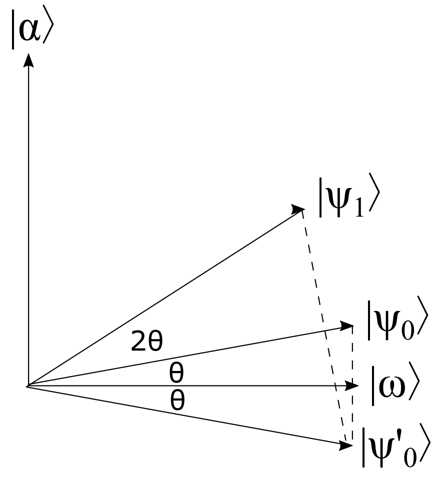


Figure III.32: Geometry of first Grover iteration.

product $\cos\theta = \langle\psi_0 \mid \omega\rangle = \sqrt{\frac{N-1}{N}}$. Therefore the sign change reflects $|\psi_0\rangle$ from θ above $|\omega\rangle$ into $|\psi_0'\rangle$, which is θ below it. Inversion about the mean reflects $|\psi_0'\rangle$ from 2θ below $|\psi_0\rangle$ into a state we call $|\psi_1\rangle$, which is 2θ above it. Therefore, in going from $|\psi_0\rangle$ to $|\psi_1\rangle$ the state vector has rotated 2θ closer to $|\alpha\rangle$.

You can see that after k iterations, the state vector $|\psi_k\rangle$ will be $(2k+1)\theta$ above $|\omega\rangle$. We can solve $(2k+1)\theta = \pi/2$ to get the required number of iterations to bring $|\psi_k\rangle$ to $|\alpha\rangle$. Note that for small θ, $\theta \approx \sin\theta = \frac{1}{\sqrt{N}}$ (which is certainly small). Hence, we want $(2k+1)/\sqrt{N} \approx \pi/2$, or $2k+1 \approx \pi\sqrt{N}/2$. That is, $k \approx \pi\sqrt{N}/4$ is the required number of iterations. Note that after $\pi\sqrt{N}/8$ iterations, we are about halfway there (i.e., $\pi/4$), so the probability of success is 50%. In general, the probability of success is about $\sin^2 \frac{2k+1}{\sqrt{N}}$.

Now for the techniques for changing the sign and inversion about the mean. Let $|\psi_k\rangle$ be the state after k iterations ($k \geq 0$). To change the sign, simply apply U_P to $|\psi_k\rangle|-\rangle$. To see the result, let $X_0 = \{x \mid P(x) = 0\}$ and $X_1 = \{x \mid P(x) = 1\}$, the solution set. Then:

$$
\begin{aligned}
& U_P|\psi_k\rangle|-\rangle \\
&= U_P\left[\sum_{x\in\mathbf{N}} a_x|x,-\rangle\right] \\
&= U_P\left[\frac{1}{\sqrt{2}}\sum_{x\in\mathbf{N}} a_x|x,0\rangle - a_x|x,1\rangle\right] \\
&= \frac{1}{\sqrt{2}}\,U_P\left[\sum_{x\in X_0} a_x|x,0\rangle + \sum_{x\in X_1} a_x|x,0\rangle - \sum_{x\in X_0} a_x|x,1\rangle - \sum_{x\in X_1} a_x|x,1\rangle\right] \\
&= \frac{1}{\sqrt{2}}\left[\sum_{x\in X_0} a_x U_P|x,0\rangle + \sum_{x\in X_1} a_x U_P|x,0\rangle \right. \\
&\qquad \left. - \sum_{x\in X_0} a_x U_P|x,1\rangle - \sum_{x\in X_1} a_x U_P|x,1\rangle\right] \\
&= \frac{1}{\sqrt{2}}\left[\sum_{x\in X_0} a_x|x,0\rangle + \sum_{x\in X_1} a_x|x,1\rangle \right. \\
&\qquad \left. - \sum_{x\in X_0} a_x|x,1\oplus 0\rangle - \sum_{x\in X_1} a_x|x,1\oplus 1\rangle\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{\sqrt{2}}\left[\sum_{x\in X_0} a_x|x\rangle|0\rangle + \sum_{x\in X_1} a_x|x\rangle|1\rangle - \sum_{x\in X_0} a_x|x\rangle|1\rangle - \sum_{x\in X_1} a_x|x\rangle|0\rangle\right]\\
&= \frac{1}{\sqrt{2}}\left(\sum_{x\in X_0} a_x|x\rangle - \sum_{x\in X_1} a_x|x\rangle\right)(|0\rangle - |1\rangle)\\
&= \left(\sum_{x\in X_0} a_x|x\rangle - \sum_{x\in X_1} a_x|x\rangle\right)|-\rangle.
\end{aligned}
$$

Therefore the signs of the solutions have been reversed (they have been rotated by π). Notice how $|-\rangle$ in the target register can be used to separate the 0 and 1 results by rotation. This is a useful idea!

It remains to show the connection between inversion about the mean and reflection across $|\psi_0\rangle$. This reflection is given by $R_{\psi_0} = 2|\psi_0\rangle\langle\psi_0| - I$. Note that:

$$
|\psi_0\rangle\langle\psi_0| = \left(\frac{1}{\sqrt{N}}\sum_{\mathbf{x}\in\mathbf{N}}|\mathbf{x}\rangle\right)\left(\frac{1}{\sqrt{N}}\sum_{\mathbf{y}\in\mathbf{N}}\langle\mathbf{y}|\right) = \frac{1}{N}\sum_{\mathbf{x}\in\mathbf{N}}\sum_{\mathbf{y}\in\mathbf{N}}|\mathbf{x}\rangle\langle\mathbf{y}|.
$$

This is the *diffusion matrix*:

$$
\begin{pmatrix}
\frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N}\\
\frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N}\\
\vdots & \vdots & \ddots & \vdots\\
\frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N}
\end{pmatrix},
$$

which, as we will see, does the averaging.

To perform inversion about the mean, let $\bar{a}$ be the average of the a_j (see Fig. III.30). Inversion about the mean is accomplished by the transformation:

$$
\sum_{j\in\mathbf{N}} a_j|x_j\rangle \mapsto \sum_{j\in\mathbf{N}}(2\bar{a} - a_j)|x_j\rangle.
$$

To see this, write $a_j = \bar{a} \pm \delta_j$, that is, as a difference from the mean. Then $2\bar{a} - a_j = 2\bar{a} - (\bar{a} \pm \delta_j) = \bar{a} \mp \delta_j$. Therefore an amplitude δ_j below the mean will be transformed to δ_j above, and vice verse. But an amplitude that is negative, and thus very far below the mean, will be transformed to an amplitude much above the mean. This is exactly what we want in order to amplify the negative components, which correspond to solutions.

Inversion about the mean is accomplished by a "Grover diffusion transformation" D. To derive the matrix D, consider the new amplitude a'_j as a function of all the others:

$$a'_j \overset{\text{def}}{=} 2\bar{a} - a_j = 2\left(\frac{1}{N}\sum_{k=0}^{N-1} a_k\right) - a_j = \sum_{k \neq j} \frac{2}{N} a_k + \left(\frac{2}{N} - 1\right) a_j .$$

This matrix has $\frac{2}{N} - 1$ on the main diagonal and $\frac{2}{N}$ in the off-diagonal elements:

$$D = \begin{pmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} - 1 \end{pmatrix} .$$

Note that $D = 2|\psi_0\rangle\langle\psi_0| - I = R_{\psi_0}$. It is easy to confirm that $DD^\dagger = I$ (Exer. III.50), so the matrix is unitary and therefore a possible quantum operation, but it remains to be seen if it can be implemented efficiently.

We claim $D = WRW$, where $W = H^{\otimes n}$ is the n-qubit Walsh-Hadamard transform and R is the *phase rotation matrix*:

$$R \overset{\text{def}}{=} \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{pmatrix} .$$

To see this, let

$$R' \overset{\text{def}}{=} R + I = \begin{pmatrix} 2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} .$$

Then $WRW = W(R' - I)W = WR'W - WW = WR'W - I$. It is easy to show (Exer. III.51) that:

$$WR'W = \begin{pmatrix} \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \end{pmatrix} .$$

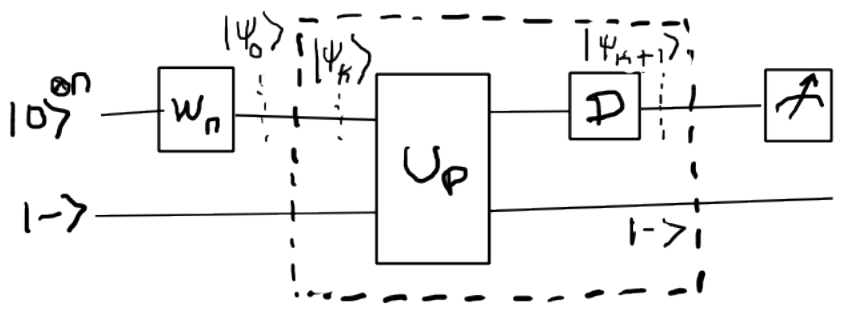


Figure III.33: Circuit for Grover's algorithm. The Grover iteration in the dashed box is repeated $\frac{\pi\sqrt{N}}{4}$ times.

It therefore follows that $D = WR'W - I = WRW$. See Fig. III.33 for a diagram of Grover's algorithm.

It remains to consider the possibility that there may be several solutions to the problem. If there are s solutions, then run the Grover iteration $\frac{\pi\sqrt{N/s}}{4}$ times, which is optimal (Exer. III.52). It can be done in $\sqrt{N/s}$ iterations even if s is unknown.

D.4.c Hogg's heuristic search algorithms

Many important problems can be formulated as *constraint satisfaction problems*, in which we try to find a set of assignments to variables that satisfy specified *constraints*. More specifically let $V \stackrel{\text{def}}{=} \{v_1, \ldots, v_n\}$ be a set of variables, and let $X \stackrel{\text{def}}{=} \{x_1, \ldots, x_n\}$ be a set of values that can be assigned to the variables, and let $C_1, \ldots, C_l$ be the constraints. The set of all possible assignments of values to variables is $V \times X$. Subsets of this set correspond to full or partial assignments, including inconsistent assignments. The set of all such assignments is $\mathcal{P}(V \times X)$.

The sets of assignments form a *lattice* under the $\subseteq$ partial order (Fig. III.34). By assigning bits to the elements of $V \times X$, elements of $\mathcal{P}(V \times X)$ can be represented by mn-element bit strings (i.e., integers in the set $\mathbf{MN} = \{0, \ldots, 2^{mn} - 1\}$); see Fig. III.35. Hogg's algorithms are based on the observation that if an assignment violates the constraints, then so do all those assignments above it in the lattice.

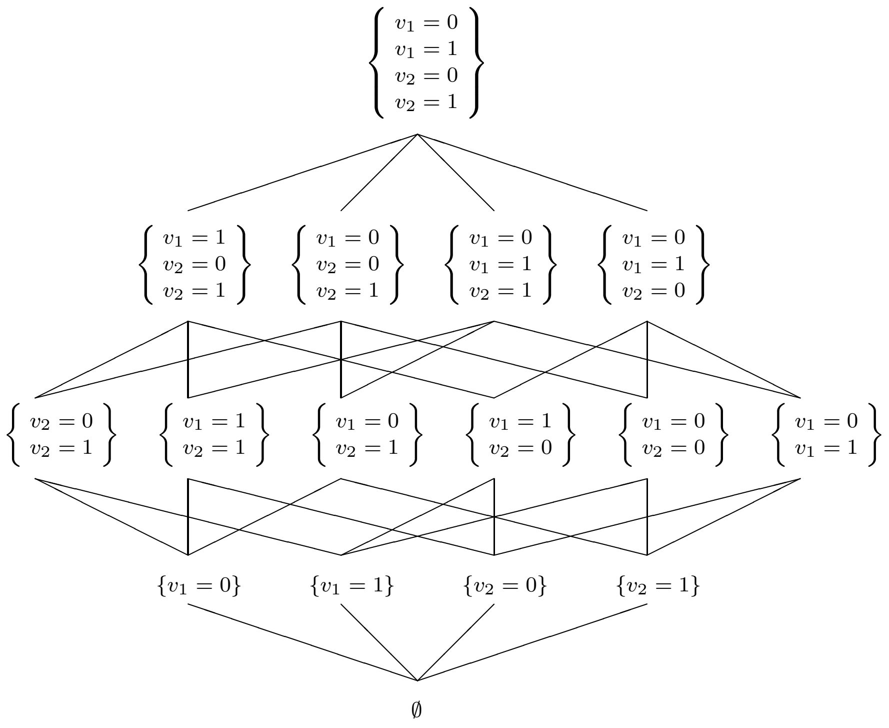


Figure III.34: Lattice of variable assignments. [source: Rieffel & Polak (2000)]

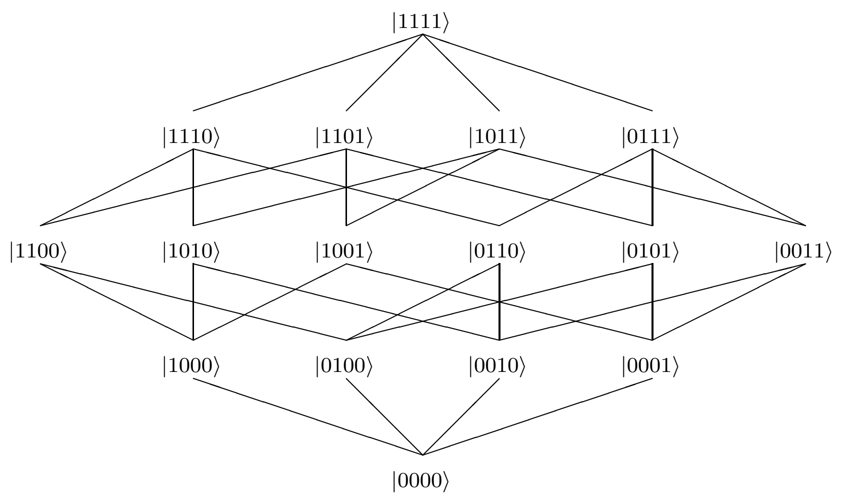


Figure III.35: Lattice of binary strings corresponding to all subsets of a 4-element set. [source: Rieffel & Polak (2000)]

algorithm Hogg:

Initialization: The algorithm begins with all the amplitude concentrated in the bottom of the lattice, $|0 \cdots 0\rangle$ (i.e., the empty set of assignments).

Movement: The algorithm proceeds by moving amplitude up the lattice, while avoiding assignments that violate the constraints; that is, we want to move amplitude from a set to its supersets. For example, we want to redistribute the amplitude from $|1010\rangle$ to $|1110\rangle$ and $|1011\rangle$. Hogg has developed several methods. One method is based on the assumption that the transformation has the form WDW, where $W = H^{\otimes mn}$, the mn-dimensional Walsh-Hadamard transformation, and D is diagonal. The elements of D depend on the size of the sets. Recall (D.1.b, p. 130) that

$$W|x\rangle = \frac{1}{\sqrt{2^{mn}}} \sum_{z \in \mathbf{MN}} (-)^{x \cdot z} |z\rangle.$$

As shown in Sec. A.2.c (p. 70), we can derive a matrix representation for W:

$$\begin{aligned}
W_{jk} &= \langle j \mid W \mid k \rangle \\
&= \langle j | \frac{1}{\sqrt{2^{mn}}} \sum_{z \in \mathbf{MN}} (-)^{k \cdot z} | z \rangle \\
&= \frac{1}{\sqrt{2^{mn}}} \sum_{z \in \mathbf{MN}} (-)^{k \cdot z} \langle j \mid z \rangle \\
&= \frac{1}{\sqrt{2^{mn}}} (-1)^{k \cdot j}.
\end{aligned}$$

Note that $k \cdot j = |k \cap j|$, where on the right-hand side we interpret the bit strings as sets.
□

The general approach is to try to steer amplitude away from sets that violate the constraints, but the best technique depends on the particular problem. One technique is to invert the phase on bad subsets so that they tend to cancel the contribution of good subsets to supersets. This could be done by a process like Grover's algorithm using a predicate that tests for violation of constraints. Another approach is to assign random phases to bad sets.

It is difficult to analyze the probability that an iteration of a heuristic algorithm will produce a solution, and so its efficiency is usually evaluated empirically, but empirical tests will be difficult to apply to quantum heuristic search until larger quantum computers are available, since classical computers require exponential time to simulate quantum systems. Small simulations, however, indicate that Hogg's algorithms may provide polynomial speedup over Grover's algorithm.

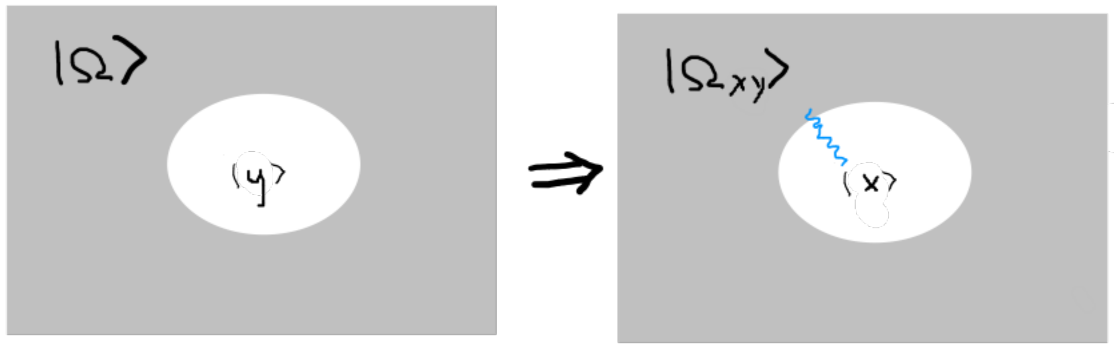


Figure III.36: Effects of decoherence on a qubit. On the left is a qubit $|y\rangle$ that is mostly isoloated from its environment $|\Omega\rangle$. On the right, a weak interaction between the qubit and the environment has led to a possibly altered qubit $|x\rangle$ and a correspondingly (slightly) altered environment $|\Omega_{xy}\rangle$.

D.5 Quantum error correction

D.5.a Motivation

Quantum coherence is very difficult to maintain for long.[19] Even weak interactions with the environment can affect the quantum state, and we've seen that the amplitudes of the quantum state are critical to quantum algorithms. On classical computers, bits are represented by very large numbers of particles (but that is changing). On quantum computers, qubits are represented by atomic-scale states or objects (photons, nuclear spins, electrons, trapped ions, etc.). They are very likely to become entangled with computationally irrelevant states of the computer and its environment, which are out of our control. Quantum error correction is similar to classical error correction in that additional bits are introduced, creating redundancy that can be used to correct errors. It is different from classical error correction in that: (a) We want to restore the entire quantum state (i.e., the continuous amplitudes), not just 0s and 1s. Further, errors are continuous and can accumulate. (b) It must obey the no-cloning theorem. (c) Measurement destroys quantum information.

[19]This section follows Rieffel & Polak (2000).

D.5.b Effect of decoherence

Ideally the environment $|\Omega\rangle$, considered as a quantum system, does not interact with the computational state. But if it does, the effect can be categorized as a unitary transformation on the environment-qubit system. Consider decoherence operator D describing a bit flip error in a single qubit (Fig. III.36):

$$D : \left\{ \begin{array}{lcl} |\Omega\rangle|0\rangle & \Longrightarrow & |\Omega_{00}\rangle|0\rangle + |\Omega_{10}\rangle|1\rangle \\ |\Omega\rangle|1\rangle & \Longrightarrow & |\Omega_{01}\rangle|0\rangle + |\Omega_{11}\rangle|1\rangle \end{array} \right. .$$

In this notation the state vectors $|\Omega_{xy}\rangle$ are not normalized, but incorporate the amplitudes of the various outcomes. In the case of no error, $|\Omega_{00}\rangle = |\Omega_{11}\rangle = |\Omega\rangle$ and $|\Omega_{01}\rangle = |\Omega_{10}\rangle = \mathbf{0}$. If the entanglement with the environment is small, then $\|\Omega_{01}\|, \|\Omega_{10}\| \ll 1$ (small exchange of amplitude).

Define *decoherence operators* $D_{xy}|\Omega\rangle \stackrel{\text{def}}{=} |\Omega_{xy}\rangle$, for $x, y \in \mathbf{2}$, which describe the effect of the decoherence on the environment. (These are not unitary, but are the products of scalar amplitudes and unitary operators for the various outcomes.) Then the evolution of the joint system is defined by the equations:

$$\begin{aligned} D|\Omega\rangle|0\rangle &= (D_{00} \otimes I + D_{10} \otimes X)|\Omega\rangle|0\rangle, \\ D|\Omega\rangle|1\rangle &= (D_{01} \otimes X + D_{11} \otimes I)|\Omega\rangle|1\rangle. \end{aligned}$$

Alternately, we can define it:

$$D = D_{00} \otimes |0\rangle\langle 0| + D_{10} \otimes |1\rangle\langle 0| + D_{01} \otimes |0\rangle\langle 1| + D_{11} \otimes |1\rangle\langle 1|.$$

Now, it's easy to show (Exer. III.20):

$$|0\rangle\langle 0| = \frac{1}{2}(I+Z), |0\rangle\langle 1| = \frac{1}{2}(X-Y), |1\rangle\langle 0| = \frac{1}{2}(X+Y), |1\rangle\langle 1| = \frac{1}{2}(I-Z),$$

where $Y = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$. Therefore

$$\begin{aligned} D &= \frac{1}{2}[D_{00} \otimes (I+Z) + D_{01} \otimes (X-Y) + \\ &\quad D_{10} \otimes (X+Y) + D_{11} \otimes (I-Z)] \\ &= \frac{1}{2}[(D_{00}+D_{11}) \otimes I + (D_{10}+D_{01}) \otimes X + \\ &\quad (D_{10}-D_{01}) \otimes Y + (D_{00}-D_{11}) \otimes Z]. \end{aligned}$$

Therefore the bit flip error can be described as a linear combination of the Pauli matrices. It is generally the case that the effect of decoherence on a single qubit can be described by a linear combination of the Pauli matrices, which is important, since qubits are subject to various errors beside bit flips. This is a distinctive feature about quantum errors: they have a finite basis, and because they are unitary, they are therefore invertible. In other words, single-qubit errors can be characterized in terms of a linear combination of the Pauli matrices (which span the space of 2×2 self-adjoint unitary matrices: C.2.a, p. 106): I (no error), X (bit flip error), Y (phase error), and $Z = YX$ (bit flip phase error). Therefore a single qubit error is represented by $e_0\sigma_0 + e_1\sigma_1 + e_2\sigma_2 + e_3\sigma_3 = \sum_{j=0}^{3} e_j\sigma_j$, where the σ_j are the Pauli matrices (Sec. C.2.a, p. 106).

D.5.c Correcting the quantum state

We consider a basis set of unitary "error operators" E_j, so that the error transformation is a superposition $E \stackrel{\text{def}}{=} \sum_j e_j E_j$. In the more general case of quantum registers, the E_j affect the entire register, not just a single qubit.

algorithm quantum error correction:

Encoding: An n-bit register is encoded in $n+m$ bits, where the extra bits are used for error correction. Let $y \stackrel{\text{def}}{=} C(x) \in \mathbf{2}^{m+n}$ be the $n+m$ bit code for $x \in \mathbf{2}^n$. As in classical error correcting codes, we embed the message in a space of higher dimension.

Error process: Suppose $\tilde{y} \in \mathbf{2}^{m+n}$ is the result of error type k, $\tilde{y} = E_k(y)$.

Syndrome: Let $k = S(\tilde{y})$ be a function that determines the error syndrome, which identifies the error E_k from the corrupted code. That is, $S(E_k(y)) = k$.

Correction: Since the errors are unitary, and the syndrome is known, we can invert the error and thereby correct it: $y = E_{S(\tilde{y})}^{-1}(\tilde{y})$.

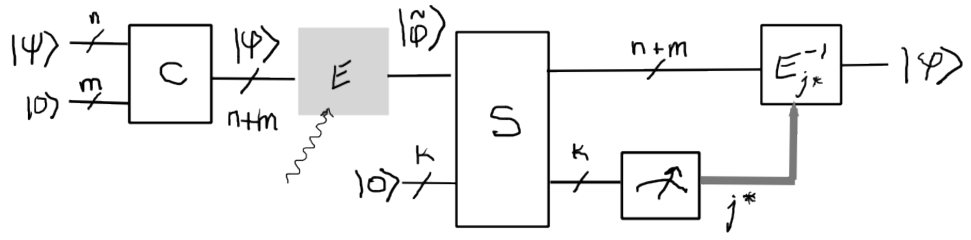


Figure III.37: Circuit for quantum error correction. $|\psi\rangle$ is the n-qubit quantum state to be encoded by C, which adds m error-correction qubits to yield the encoded state $|\phi\rangle$. E is a unitary superposition of error operators E_j, which alter the quantum state to $|\tilde{\phi}\rangle$. S is the syndrome extraction operator, which computes a superposition of codes for the errors E. The syndrome register is measured, to yield a particular syndrome code j^*, which is used to select a corresponding inverse error transformation $E_{j^*}^{-1}$ to correct the error.

Quantum case: Now consider the quantum case, in which the state $|\psi\rangle$ is a superposition of basis vectors, and the error is a superposition of error types, $E = \sum_j e_j E_j$. This is an orthogonal decomposition of E (see Fig. III.37).

Encoding: The encoded state is $|\phi\rangle \overset{\text{def}}{=} C|\psi\rangle|\mathbf{0}\rangle$. There are several requirements for a useful quantum error correcting code. Obviously, the codes for orthogonal inputs must be orthogonal; that is, if $\langle \psi \mid \psi' \rangle = 0$, then $C|\psi, \mathbf{0}\rangle$ and $C|\psi', \mathbf{0}\rangle$ are orthogonal: $\langle \psi, \mathbf{0}|C^\dagger C|\psi', \mathbf{0}\rangle = 0$. Next, if $|\phi\rangle$ and $|\phi'\rangle$ are the codes of distinct inputs, we do not want them to be confused by the error processes, so $\langle \phi|E_j^\dagger E_k|\phi'\rangle = 0$ for all i, j. Finally, we require that for each pair of error indices j, k, there is a number m_{jk} such that $\langle \phi|E_j^\dagger E_k|\phi\rangle = m_{jk}$ for every code $|\phi\rangle$. This means that the error syndromes are independent of the codes, and therefore the syndromes can be measured without collapsing superpositions in the codes, which would make them useless for quantum computation.

Error process: Let $|\tilde{\phi}\rangle \overset{\text{def}}{=} E|\phi\rangle$ be the code corrupted by error.

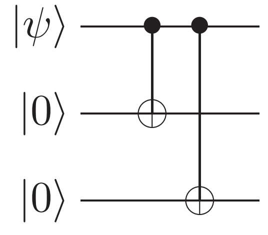


Figure III.38: Quantum encoding circuit for triple repetition code. [source: NC]

Syndrome extraction: Apply the syndrome extraction operator to the encoded state, augmented with enough extra qubits to represent the set of syndromes. This yields a superposition of syndromes:

$$S|\tilde{\phi}, \mathbf{0}\rangle = S\left(\sum_j e_j E_j |\phi\rangle\right) \otimes |\mathbf{0}\rangle = \sum_j e_j (S E_j |\phi\rangle|\mathbf{0}\rangle) = \sum_j e_j (E_j |\phi\rangle|j\rangle).$$

Measurement: Measure the syndrome register to obtain some j^* and the collapsed state $E_{j^*}|\phi\rangle|j^*\rangle$.[20]

Correction: Apply $E_{j^*}^{-1}$ to correct the error.
□

Note the remarkable fact that although there was a superposition of errors, we only have to correct one of them to get the original state back. This is because measurement of the error syndrome collapses into a state affected by just that one error.

D.5.d Example

We'll work through an example to illustrate the error correction process. For an example, suppose we use a simple triple redundancy code that assigns

[20]As we mentioned the discussion of in Shor's algorithm (p. 141), it is not necessary to actually perform the measurement; the same effect can be obtained by unitary operations.

$|0\rangle \mapsto |000\rangle$ and $|1\rangle \mapsto |111\rangle$. This is accomplished by a simple quantum gate array:

$$C|0\rangle|00\rangle = |000\rangle, \quad C|1\rangle|00\rangle = |111\rangle.$$

This is not a sophisticated code! It's called a *repetition code.* The three-qubit codes are called *logical zero* and *logical one* (See Fig. III.38). This code can correct single bit flips (*by majority voting*); the errors are represented by the operators:

$$\begin{aligned} E_0 &= I \otimes I \otimes I \\ E_1 &= I \otimes I \otimes X \\ E_2 &= I \otimes X \otimes I \\ E_3 &= X \otimes I \otimes I. \end{aligned}$$

The following works as a syndrome extraction operator:

$$S|x_3, x_2, x_1, 0, 0, 0\rangle \overset{\text{def}}{=} |x_3, x_2, x_1, x_1 \oplus x_2, x_1 \oplus x_3, x_2 \oplus x_3\rangle.$$

The $\oplus$s compare each pair of bits, and so the $\oplus$ will be zero if the two bits are the same (the majority). The following table shows the bit flipped (if any), the corresponding syndrome, and the operator to correct it (which is the same as the operator that caused the error):

bit flipped	syndrome	error correction
none	$\|000\rangle$	$I \otimes I \otimes I$
1	$\|110\rangle$	$I \otimes I \otimes X$
2	$\|101\rangle$	$I \otimes X \otimes I$
3	$\|011\rangle$	$X \otimes I \otimes I$

(Note that the correction operators need not be the same as the error operators, although they are in this case.)

For an example, suppose we want to encode the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Its code is $|\phi\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. Suppose the following error occurs: $E = \frac{4}{5} X \otimes I \otimes I + \frac{3}{5} I \otimes X \otimes I$ (that is, the bit 3 flips with probability 16/25, and bit 2 with probability 9/25). The resulting error state is

$$\begin{aligned} |\tilde{\phi}\rangle &= E|\phi\rangle \\ &= \left(\frac{4}{5} X \otimes I \otimes I + \frac{3}{5} I \otimes X \otimes I\right) \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) \end{aligned}$$

$$
\begin{aligned}
&= \frac{4}{5\sqrt{2}} X \otimes I \otimes I(|000\rangle - |111\rangle) + \frac{3}{5\sqrt{2}} I \otimes X \otimes I(|000\rangle - |111\rangle) \\
&= \frac{4}{5\sqrt{2}}(|100\rangle - |011\rangle) + \frac{3}{5\sqrt{2}}(|010\rangle - |101\rangle).
\end{aligned}
$$

Applying the syndrome extraction operator yields:

$$
\begin{aligned}
S|\tilde{\phi}, 000\rangle &= S\left[\frac{4}{5\sqrt{2}}(|100000\rangle - |011000\rangle) + \frac{3}{5\sqrt{2}}(|010000\rangle - |101000\rangle)\right] \\
&= \frac{4}{5\sqrt{2}}(|100011\rangle - |011011\rangle) + \frac{3}{5\sqrt{2}}(|010101\rangle - |101101\rangle) \\
&= \frac{4}{5\sqrt{2}}(|100\rangle - |011\rangle) \otimes |011\rangle + \frac{3}{5\sqrt{2}}(|010\rangle - |101\rangle) \otimes |101\rangle
\end{aligned}
$$

Measuring the syndrome register yields either $|011\rangle$ (representing an error in bit 3) or $|101\rangle$ (representing an error in bit 2). Suppose we get $|011\rangle$. The state collapses into:

$$
\frac{1}{\sqrt{2}}(|100\rangle - |011\rangle) \otimes |011\rangle.
$$

Note that we have projected into a subspace for just one of the two bit-flip errors that occurred (the flip in bit 3). The measured syndrome $|011\rangle$ tells us to apply $X \otimes I \otimes I$ to the first three bits, which restores $|\phi\rangle$:

$$
(X \otimes I \otimes I)\frac{1}{\sqrt{2}}(|100\rangle - |011\rangle) = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = |\phi\rangle.
$$

We can do something similar to correct single phase flip (Z) errors by using the encoding $|0\rangle \mapsto |{+}{+}{+}\rangle$, $|1\rangle \mapsto |{-}{-}{-}\rangle$ (Exer. III.55). To see this, recall that Z in the sign basis is the analog of X is the computational basis.

D.5.e Discussion

There is a nine-qubit code, called the *Shor code*, that can correct arbitrary errors on a single qubit, even replacing the entire qubit by garbage (Nielsen & Chuang, 2010, §10.2). The essence of this code is that triple redundancy is used to correct X errors, and triple redundancy again to correct Z errors, thus requiring nine code qubits for each logical qubut. Since $Y = ZX$ and the Pauli matrices are a basis, this code is able to correct all errors.

Quantum error correction is remarkable in that an entire continuum of errors can be corrected by correcting only a discrete set of errors. This works in quantum computation, but not classical analog computing. The general goal in syndrome extraction is to separate the syndrome information from the computational information in such a way that the syndrome can be measured without collapsing any of the computational information. Since the syndrome is unitary, it can be inverted to correct the error.

What do we do about noise in the gates that do the encoding and decoding? It is possible to do *fault-tolerant quantum computation.* "Even more impressively, fault-tolerance allow us to perform logical operations on encoded quantum states, in a manner which tolerates faults in the underlying gate operations." (Nielsen & Chuang, 2010, p. 425) Indeed, "provided the noise in individual quantum gates is below a certain constant threshold it is possible to efficiently perform an arbitrarily large quantum computation." (Nielsen & Chuang, 2010, p. 425)[21]

[21]See Nielsen & Chuang (2010, §10.6.4).

E Abrams-Lloyd theorem

E.1 Overview

All experiments to date confirm the linearity of QM, but what would be the consequences of slight nonlinearities?[22] We will see that nonlinearities can be exploited to solve NP problems (and in fact harder problems) in polynomial time. This fact demonstrates clearly that computability and complexity are not purely mathematical matters. Because computation is inherently physical, fundamental physics is intertwined with fundamental computation theory. It also exposes the fact that there are hidden physical assumptions in the traditional theory of computation.

How could nonlinearities be exploited in quantum computation? Recall that quantum state vectors lie on the unit sphere and unitary transformations preserve "angles" (inner products) between vectors. Nonunitary transformations would, in effect, stretch the sphere, so the angles between vectors could change. Unitary transformations could be used to position the vectors to the correct position for subsequent nonunitary transformations. The following algorithm exploits a nonlinear operator to separate vectors that are initially close together.

E.2 Basic algorithm

The *Lyapunov exponent* λ describes the divergence of trajectories in a dynamical system. If $\Delta\theta(0)$ is the initial separation, then the separation after time t is given by $|\Delta\theta(t)| \approx e^{\lambda t}|\Delta\theta(0)|$. If $\lambda > 0$, the system is usually chaotic.

Suppose there is some nonlinear operation $\mathfrak{N}$ with a positive Lyapunov exponent over some finite region of the unit sphere. Further suppose we have an *oracle* $P : \mathbf{2}^n \rightarrow \mathbf{2}$. We want to determine if there is an $\mathbf{x}$ such that $P(\mathbf{x}) = 1$. Next suppose we are given a quantum gate array U_P as in

[22]This section is based on Daniel S. Abrams and Seth Lloyd (1998), "Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems." *Phys. Rev. Lett.* 81, 3992–3995 (1998), preprint available at `http://arxiv.org/abs/quant-ph/9801041v1`. See also Scott Aaronson, "NP-complete Problems and Physical Reality," *SIGACT News*, Complexity Theory Column, March 2005. `quant-ph/0502072`. `http://www.scottaaronson.com/papers/npcomplete.pdf` (accessed 2012-10-27).

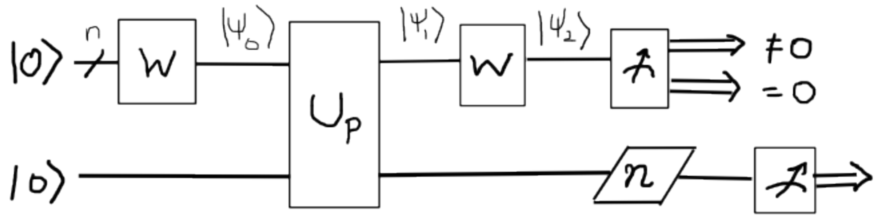


Figure III.39: Quantum circuit for Abrams-Lloyd algorithm. The first measurement produces **0** with probability greater than 1/4, but if it yields a nonzero state, we try again. The $\mathfrak{N}$ parallelogram represents a hypothetical nonlinear quantum state transformation, which may be repeated to yield a macroscopically observable separation of the solution and no-solution vectors.

Grover's algorithm. It is defined on a n-qubit data register and a 1-qubit result register. See Fig. III.39.

algorithm Abrams-Lloyd:

Step 1: As usual, apply the Walsh-Hadamard transform to a zero data register to get a superposition of all possible inputs:

$$|\psi_0\rangle = (W_n|\mathbf{0}\rangle)|0\rangle = \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}\in\mathbf{2}^n}|\mathbf{x},0\rangle.$$

Step 2 (apply oracle): Apply the oracle to get a superposition of input-output pairs:

$$|\psi_1\rangle = U_P|\psi_0\rangle = \frac{1}{\sqrt{2^n}}\sum_{\mathbf{x}\in\mathbf{2}^n}|\mathbf{x},P(\mathbf{x})\rangle.$$

Step 3 (measure data register): First, apply the Walsh transformation to the data register to get:

$$|\psi_2\rangle \;=\; (W_n\otimes I)|\psi_1\rangle$$

$$\begin{aligned} &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}\in\mathbf{2}^n} (W_n|\mathbf{x}\rangle)|P(\mathbf{x})\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}\in\mathbf{2}^n} \left[\frac{1}{\sqrt{2^n}} \sum_{\mathbf{z}\in\mathbf{2}^n} (-)^{\mathbf{x}\cdot\mathbf{z}}|\mathbf{z}\rangle\right] |P(\mathbf{x})\rangle. \end{aligned}$$

The last step applies by Eq. III.24 (p. 130). That is,

$$|\psi_2\rangle = \sum_{\mathbf{x}\in\mathbf{2}^n}\sum_{\mathbf{z}\in\mathbf{2}^n} \frac{1}{2^n}(-)^{\mathbf{x}\cdot\mathbf{z}}|\mathbf{z}\rangle|P(\mathbf{x})\rangle.$$

Separate out the state $\mathbf{z} = \mathbf{0}$ in order to see its amplitude:

$$\sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{2^n}(-)^{\mathbf{x}\cdot\mathbf{0}}|\mathbf{0}\rangle|P(\mathbf{x})\rangle = \sum_{\mathbf{x}\in\mathbf{2}^n} \frac{1}{2^n}|\mathbf{0}\rangle|P(\mathbf{x})\rangle.$$

At least half of the 2^n vectors $\mathbf{x}$ must have the same value, $a = P(\mathbf{x})$ (since $P(\mathbf{x}) \in \mathbf{2}$). Therefore the amplitude of $|\mathbf{0}, a\rangle$ is at least $1/2$, and the probability of observing $|\mathbf{0}, a\rangle$ is at least $1/4$. We get a non-zero data register with probability $\leq 3/4$. (If we happen to observe $|\mathbf{x}_0, 1\rangle$, then of course we have our answer.)

In the case in which we get a zero data register, measurement of the data register yields the state:

$$|\psi_2\rangle \xrightarrow{\geq\frac{1}{4}} \mathcal{Z}^{-1}\left(\frac{s}{2^n}|\mathbf{0}\rangle|1\rangle + \frac{2^n - s}{2^n}|\mathbf{0}\rangle|0\rangle\right) = \mathcal{Z}^{-1}|\mathbf{0}\rangle\left(\frac{s}{2^n}|1\rangle + \frac{2^n - s}{2^n}|0\rangle\right),$$

where s is the number of solutions (the number of $\mathbf{x}$ such that $P(\mathbf{x}) = 1$) and $\mathcal{Z}^{-1}$ renormalizes after the state collapse.

The information we want is in the result qubit, but if s is small (as expected), then measurement will almost always yield $|0\rangle$. Recall what we did in Grover's algorithm. For $s \ll 2^n$, the vector $\mathcal{Z}^{-1}|\mathbf{0}\rangle\left(\frac{s}{2^n}|1\rangle + \frac{2^n-s}{2^n}|0\rangle\right)$ is very close to the vector $|\mathbf{0}, 0\rangle$. Therefore, we would like to drive them apart.

Step 4: Applying the nonlinear operator $\mathfrak{N}$ repeatedly will separate the vectors at an exponential rate. "[E]ventually, at a time determined by a polynomial function of the number of qubits n, the number of solutions s, and the rate of spreading (Lyapunov exponent) λ, the two cases will become macroscopically distinguishable."

Step 5 (measure result register): Measure the result qubit. If the vectors have been sufficiently separated, there will be a significant probability of observing $|1\rangle$ in the $s \neq 0$ case.
$\square$

If η is the angular extent of the nonlinear region, it might take $\mathcal{O}((\pi/\eta)^2)$ trials to get $|1\rangle$ with high probability. For large η, just one iteration might be sufficient.

E.3 Discussion of further results

The preceding algorithm depends on exponential precision, but Abrams and Lloyd present another algorithm that is robust against small errors. Each iteration doubles the number of components that have $|1\rangle$ in the result qubit, and after n iterations it yields a result with probability 1, and so the algorithm is linear.

Scott Aaronson has expressed doubts that the required nonlinear OR gate can be implemented. Abrams and Lloyd summarize: "We have demonstrated that nonlinear time evolution can in fact be exploited to allow a quantum computer to solve NP-complete and #P problems in polynomial time." (#P or "number-P" asks how many accepting paths in a NTM running in polynomial time. #P problems are at least as hard as corresponding NP problems.) Nevertheless, they continue, "we believe that quantum mechanics is in all likelihood exactly linear, and that the above conclusions might be viewed most profitably as further evidence that this is indeed the case."

F Universal quantum computers

Hitherto we have used a practical definition of universality: since conventional digital computers are implemented in terms of binary digital logic, we have taken the ability to implement binary digital logic as sufficient for universality. This leave open the question of the relation of quantum computers to the theoretical standard of computational universality: the Turing machine. Therefore, a natural question is: What is the power of a quantum computer? Is it super-Turing or sub-Turing? Another question is: What is its efficiency? Can it solve NP problems efficiently? There are a number of universal quantum computing models for both theoretical and practical purposes.

F.1 Feynman on quantum computation

F.1.a Simulating quantum systems

In 1982 Richard Feynman discussed what would be required to simulate a quantum mechanical system on a digital computer.[23] First he considered a classical probabilistic physical system. Suppose we want to use a conventional computer to calculate the probabilities as the system evolves in time. Further suppose that the system comprises R particles that are confined to N locations in space, and that each configuration c has a probability $p(c)$. There are N^R possible configurations, since a configuration assigns a location N to each of the R particles (i.e., the number of functions $R \rightarrow N$). Therefore to simulate all the possibilities would require keeping track of a number of quantities (the probabilities) that grows exponentially with the size of the system. This is infeasible.

So let's take a weaker goal: we want a simulator that exhibits the same probabilistic behavior as the system. Our goal is that if we run both of them over and over, we will see the same distribution of behaviors. This we can do. You can implement this by having a nondeterministic computer that has the same state transition probabilities as the primary system.

Let's try the same trick with quantum systems, that is, have a conventional computer that exhibits the same probabilities as the quantum system. If you do the math (which we won't), it turns out that this is impossible. The reason is that, in effect, some of the state transitions would have to have

[23]This section is based primarily on Feynman (1982).

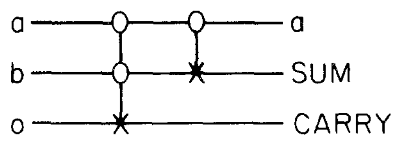


Figure III.40: Simple adder using reversible logic. [fig. from Feynman (1986)]

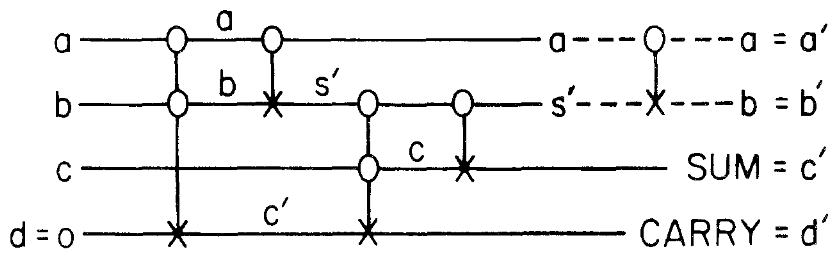


Figure III.41: Full adder using reversible logic. [fig. from Feynman (1986)]

what amount to negative probabilities, and we don't know how to do this classically. We've seen how in quantum mechanics, probabilities can in effect cancel by destructive interference of the wavefunctions. The conclusion is that no conventional computer can efficiently simulate a quantum computer. Therefore, if we want to (efficiently) simulate any physical system, we need a quantum computer.

F.1.b Universal quantum computer

In 1985 Feynman described several possible designs for a universal quantum computer.[24] He observes that NOT, CNOT, and CCNOT are sufficient for any logic gate, as well as for COPY and EXCHANGE, and therefore for universal computation. He exhibits circuits for a simple adder (Fig. III.40) and a full adder (Fig. III.41).

The goal is to construct a Hamiltonian to govern the operation of a quantum computer. Feynman describes quantum logic gates in terms of two primitive operations, which change the state of an "atom" (two-state system or "wire"). Letters near the beginning of the alphabet ($a, b, c, \ldots$) are used for

[24]This section is based primarily on Feynman (1986).

data or *register atoms*, and those toward the end $(p, q, r, \ldots)$ for *program atoms* (which are used for sequencing operations). In this simple sequential computer, only one program atom is set at a time.

For a single line a, the *annihilation operator* is defined:

$$a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 1|.$$

The *annihilator* changes the state $|1\rangle$ to $|0\rangle$; typically it lowers the energy of a quantum system. Applied to $|0\rangle$, it leaves the state unchanged and returns the *zero vector* $\mathbf{0}$ (which is not a meaningful quantum state). It *matches* $|1\rangle$ and *resets* it to $|0\rangle$. The operation is not unitary (because not norm preserving). It is a "partial NOT" operation.

Its conjugate transpose it the *creation operation*:[25]

$$a^* = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = |1\rangle\langle 0|.$$

The *creator* transforms $|0\rangle$ to $|1\rangle$, but leaves $|1\rangle$ alone, returning $\mathbf{0}$; typically it raises the energy of a quantum system. It matches $|0\rangle$ and sets it to $|1\rangle$. This is the other half of NOT $= a + a^*$.

Feynman also defines a *number operation* or *1-test*: Consider[26]

$$N_a = a^* a = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = |1\rangle\langle 1|.$$

This has the effect of returning $|1\rangle$ for input $|1\rangle$, but $\mathbf{0}$ for $|0\rangle$:

$$N_a = a^* a = |1\rangle\langle 0|\ |0\rangle\langle 1| = |1\rangle\langle 0 \mid 0\rangle\langle 1| = |1\rangle\langle 1|.$$

Thus it's a test for $|1\rangle$. (This is a partial identity operation.)

Similarly, the *0-test* is defined:[27]

$$\mathbf{1} - N_a = a a^* = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = |0\rangle\langle 0|.$$

[25]Note that Feynman uses a^* for the adjoint (conjugate transpose) of a.

[26]This matrix is not the same as that given in Feynman (1982, 1986), since Feynman uses the basis $|1\rangle = (1,0)^{\mathrm{T}}, |0\rangle = (0,1)^{\mathrm{T}}$, whereas we use $|0\rangle = (1,0)^{\mathrm{T}}$ and $|1\rangle = (0,1)^{\mathrm{T}}$.

[27]This matrix is different from that given in Feynman (1982, 1986), as explained in the previous footnote.

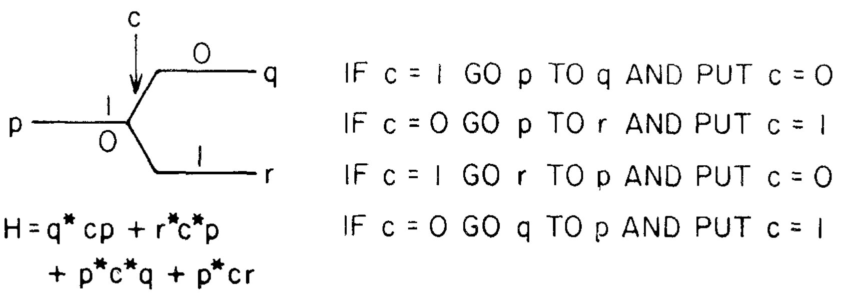

c
0
q
p
1
0
1
r
H = q* cp + r*c*p
+ p*c*q + p*cr
IF c = 1 GO p TO q AND PUT c = 0
IF c = 0 GO p TO r AND PUT c = 1
IF c = 1 GO r TO p AND PUT c = 0
IF c = 0 GO q TO p AND PUT c = 1

Figure III.42: Switch element. 0/1 annotations on the wires show the c values. [fig. from Feynman (1986)]

(Feynman writes this $\mathbf{1} - N_a$ because he writes $\mathbf{1} = I$.) This has the effect of returning $|0\rangle$ for input $|0\rangle$, but $\mathbf{0}$ for $|1\rangle$. This is test for $|0\rangle$. (It is the rest of the identity operation.)

The two operations a and a^* are sufficient for creating all 2×2 matrices, and therefore all transformations on a single qubit. Note that

$$\begin{pmatrix} w & x \\ y & z \end{pmatrix} = waa^* + xa + ya^* + za^*a.$$

Feynman writes A_a for the negation (NOT) operation applied to a. Obviously, $A_a = a + a^*$ (it annihilates $|1\rangle$ and creates from $|0\rangle$) and $\mathbf{1} = aa^* + a^*a$ (it passes $|0\rangle$ and passes $|1\rangle$. We can prove that $A_aA_a = \mathbf{1}$ (Exer. III.56).

Feynman writes $A_{a,b}$ for the CNOT operation applied to lines a and b. Observe, $A_{a,b} = a^*a(b + b^*) + aa^*$. Notice that this is a tensor product on the register $|a, b\rangle$: $A_{a,b} = a^*a \otimes (b + b^*) + aa^* \otimes \mathbf{1}$. You can write this formula $N_a \otimes A_b + (\mathbf{1} - N_a) \otimes \mathbf{1}$. That is, if N_a detects $|1\rangle$, then it negates b. If $\mathbf{1} - N_a$ detects $|0\rangle$, then it leaves b alone.

Feynman writes $A_{ab,c}$ for the CCNOT operation applied to lines a, b, and c. Note that $A_{ab,c} = \mathbf{1} + a^*ab^*b(c + c^* - \mathbf{1})$ (Exer. III.57). This formula is more comprehensible in this form:

$$A_{ab,c} = \mathbf{1} + N_aN_b(A_c - \mathbf{1}).$$

One of Feynman's universal computers is based on only two logic gates, Not and Switch (Fig. III.42). If $c = |1\rangle$, then the "cursor" (locus of control)

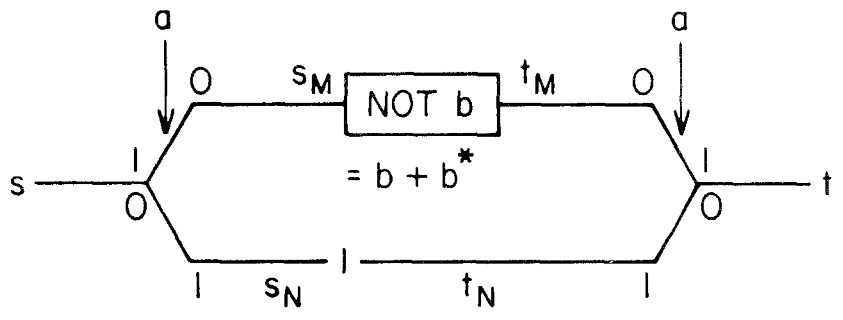


Figure III.43: CNOT implemented by switches. 0/1 annotations on the wires show the a values. [fig. from Feynman (1986)]

at p moves to q, but if $c = |0\rangle$ it moves to r. It also negates c in the process. It's also reversible (see Fig. III.42).

The switch is a tensor product on $|c, p, q, r\rangle$:

$$q^*cp + r^*c^*p + [p^*c^*q + p^*cr].$$

(The bracketed expression is just the complex conjugate of the first part, required for reversibility.) Read the factors in each term from right to left:
(1) q^*cp: if p and c are set, then unset them and set q.
(2) r^*c^*p: if p is set and c is not set, the unset p and set c and r.

Fig. III.43 shows CNOT implemented by switches. This is the controlled-NOT applied to data a, b and sequenced by cursor atoms s, t (= start, terminate). If $a = 1$ the cursor state moves along the top line, and if $a = 0$ along the bottom. If it moves along the top, then it applies $b + b^*$ to negate b (otherwise leaving it alone). In either case, the cursor arrives at the reversed switch, where sets the next cursor atom t. We can write it

$$H_{a,b}(s,t) = s_M^*as + t^*a^*t_M + t_M^*(b+b^*)s_M + s_N^*a^*s + t^*at_N + t_N^*s_N + \text{c.c},$$

where "c.c" means to add the complex conjugates of the preceding terms. Read the factors in each term from right to left:
(1) s_M^*as: if s and a are set, then unset them and set s_M.
(4) $s_N^*a^*s$: if s is set and a in unset, then unset s and set s_N and a.
(6) $t_N^*s_N$: if s_N is set, then unset it and set t_N.
(3) $t_M^*(b+b^*)s_M$: if s_M is set, then unset it, negate b and set t_M.

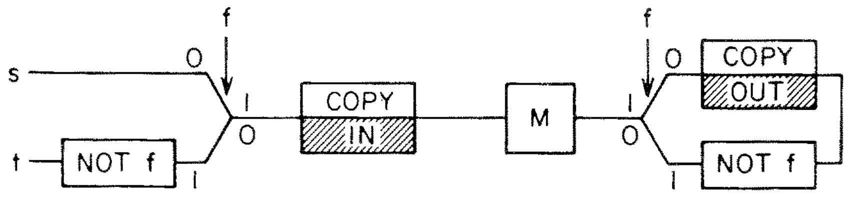


Figure III.44: Garbage clearer. 0/1 annotations on the wires show the f values. [fig. from Feynman (1986)]

(5) $t^* a t_N$: if t_N and a are set (as a must be to get here), then unset them and set t.
(2) $t^* a^* t_M$: if t_M is set and a is unset (as it must be to get here), then reverse their states and set t.
(The $t_N^* s_N$ term can be eliminated by setting $t_N = s_N$.)

F.1.c Garbage clearer

Instead of having a separate copy of the machine to clear out the garbage, it's possible to run the same machine backwards (Fig. III.44). An external register In contains the input, and the output register Out and all machine registers are initialized to 0s. Let s be the starting program atom. The flag f is initially 0.

The $f = 0$ flag routes control through the reversed switch (setting $f = 1$) to Copy. The Copy box uses CNOTs to copy the external input into M. Next M operates, generating the result in an internal register. At the end of the process M contains garbage.

The $f = 1$ flag directs control into the upper branch (resetting $f = 0$), which uses CNOTs to copy the result into the external output register Out. Control then passes out from the upper branch of the switch down and back into the lower branch, which negates f, setting it to $f = 1$. Control passes back into the machine through the lower switch branch (resetting $f = 0$), and backwards through M, clearing out all the garbage, restoring all the registers to 0s. It passes backwards through the Copy box, copying the input back from M to the external input register In. This restores the internal registers to 0s. Control finally passes out through the lower branch of the left switch (setting $f = 1$), but it negates f again, so $f = 0$. It arrives at the terminal program atom t. At the end of the process, everything is reset to the initial conditions, except that we have the result in the Out register. Feynman also discusses how to do subroutines and other useful programming constructs, but we won't go into those details.

F.2 Benioff's quantum Turing machine

In 1980 Paul Benioff published the first design for a universal quantum computer, which was based on the Turing machine (Benioff, 1980). The tape is represented by a finite lattice of quantum spin systems with eigenstates corresponding to the tape symbols. (Therefore, he cannot implement an open-ended TM tape, but neither can an ordinary digital computer.) The read/write head is a spinless system that moves along the lattice. The state of the TM was represented by another spin system. Benioff defined unitary operators for doing the various operations (e.g., changing the tape). In 1982 he extended his model to erase the tape, as in Bennett's model (Benioff, 1982). Each step was performed by measuring both the tape state under the head and the internal state (thus collapsing them) and using this to control the unitary operator applied to the tape and state. As a consequence, the computer does not make much use of superposition.

F.3 Deutsch's universal quantum computer

Benioff's computer is effectively classical; it can be simulated by a classical Turing machine. Moreover, Feynman's construction is not a true universal computer, since you need to construct it for each computation, and it's not obvious how to get the required dynamical behavior. Deutsch sought a broader definition of quantum computation, and a universal quantum computer $\mathcal{Q}$.[28] M binary observables are used to represent the processor, $\{\check{n}_i\}$ for $i \in \mathbf{M}$, where $\mathbf{M} = \{0, \ldots, M-1\}$. Collectively they are called $\check{\mathbf{n}}$. An infinite sequence of binary observables implements the memory, $\{\check{m}_i\}$ $(i \in \mathbb{Z})$ Collectively the sequence is called $\check{\mathbf{m}}$. An observable $\check{x}$, with spectrum $\mathbb{Z}$, represents the tape position (address) of the head. The *computational basis states* have the form:

$$|x; \mathbf{n}; \mathbf{m}\rangle \overset{\text{def}}{=} |x; n_0, n_1, \ldots, n_{M-1}; \ldots, m_{-1}, m_0, m_1, \ldots\rangle.$$

[28]This section is based on Deutsch (1985).

Here the eigenvectors are labeled by their eigenvalues $x, \mathbf{n}$, and $\mathbf{m}$.

The dynamics of computation is described by a unitary operator U, which advances the computation by one step of duration T:

$$|\psi(nT)\rangle = U^n|\psi(0)\rangle.$$

Initially, only a finite number of the memory elements are prepared in a non-zero state.

$$|\psi(0)\rangle = \sum_m \lambda_m|0;\mathbf{0};\mathbf{m}\rangle, \text{ where } \sum_m |\lambda_m|^2 = 1,$$

That is, only finitely many $\lambda_m \neq 0$, and in particular $\lambda_m = 0$ when an infinite number of the $\mathbf{m}$ are non-zero. The non-zero entries are the program and its input. Note that the initial state may be a superposition of initial tapes.

The matrix elements of U (relating the new state to the current state) have the form:

$$\begin{aligned}&\langle x';\mathbf{n}';\mathbf{m}' \mid U \mid x;\mathbf{n};\mathbf{m}\rangle\\ &= [\delta_{x'}^{x+1}U^+(\mathbf{n}', m'_x|\mathbf{n}, m_x) + \delta_{x'}^{x-1}U^-(\mathbf{n}', m'_x|\mathbf{n}, m_x)] \prod_{y\neq x} \delta_{m'_y}^{m_y}.\end{aligned}$$

U^+ and U^- represent moves to the right and left, respectively. The first two δ functions ensure that the tape position cannot move by more than one position in a step. The final product of deltas ensures that all the other tape positions are unchanged; it's equivalent to: $\forall y \neq x : m'_y = m_y$. The U^+ and U^- functions define the actual transitions of the machine in terms of the processor state and the symbol under the tape head. Each choice defines a quantum computer $\mathcal{Q}[U^+, U^-]$.

The machine cannot be observed before it has halted, since that would generally alter its state. Therefore one of the processor's bits is chosen as a halt indicator. It can be observed from time to time without affecting the computation.

$\mathcal{Q}$ can simulate TMs, but also any other quantum computer to arbitrary precision. In fact, it can simulate any finitely realizable physical system to arbitrary precision, and it can simulate some physical systems that go beyond the power of TMs (hypercomputation).

G Quantum probability in cognition

There are interesting connections between the mathematics of quantum mechanics and information processing in the brain. This is not so remarkable when we recall that the foundations of quantum mechanics are matters of information and knowledge.[29]

G.1 Theories of decision making

How do people make decisions under uncertainty? There have been three major phases of models.

(i) Logic. From Aristotle's time, the most common model of human thinking has been formal logic, especially deductive logic. For example, the title of George Boole's book, in which he introduced Boolean algebra, was called *The Laws of Thought*, and that is what he supposed it to be. The first AI program (1956) was the LOGIC THEORIST, and formal deductive logic still dominates many AI systems. Since the 1960s, however, there has been accumulating psychological evidence that classical logic is not a good model of everyday reasoning.

An additional, more technical problem is that classical logic is *monotonic*, that is, the body of derived theorems can only increase. But everyday reasoning is *nonmonotonic*: propositions that were previously taken to be true can become false (either because the facts have changed or an assumption has been invalidated. As a consequence, the body of truths can shrink or change in other ways. Existing truths can be nullified. An additional problem is that much of our reasoning is *inductive* rather than deductive, that is, it moves from more particular premises to more general conclusions, rather than vice versa, as deductive logic does. But after many years of research, there really isn't an adequate inductive logic that accounts for scientific reasoning as well as everyday generalization.

(ii) Classical probability (CP). The most common models of human decision making have been based on classical probability theory (CP) and Bayesian inference. Amos Tversky and Daniel Kahneman were pioneers (from the 1970s) in the study of how people actually make decisions and judgments. (In 2002 Kahneman recieved the Nobel Prize in Economics for

[29]This chapter is based primarily on Emmanuel M. Pothos and Jerome R. Busemeyer, "Can quantum probability provide a new direction for cognitive modeling?" *Behavioral and Brain Sciences* (Pothos & Busemeyer, 2013), including MacLennan (2013).

this work; Tversky had already died.) Since then many other psychologists have confirmed and extended their findings. They concluded that everyday human reasoning follows the laws of neither classical logic nor classical probability theory. "Many of these findings relate to order/context effects, violations of the law of total probability (which is fundamental to Bayesian modeling), and failures of compositionality." (Pothos & Busemeyer, 2013)

(iii) Quantum probability (QP). The mathematics of quantum mechanics provides an alternative system (axiomatization) of probability which has the potential to account for these violations of CP, as we will see. Note that QP is just a probability theory; there is no presumption that physical quantum phenomena are significant in the brain (although they might be). In this sense, our brains appear to be using a kind of quantum computation.

G.2 Framework

G.2.a Questions & outcomes

Just as CP begins by defining a sample space, QP begins by defining a Hilbert space, which defines all possible answers that could be produced for all possible questions (addressed by the model). Corresponding to the *quantum state* is the *cognitive state*, which you can think of as the indeterminate state of the brain before there is a decision or determination to act in some way (such as answering a question). Corresponding to *observables* in quantum mechanics, we have *questions* in QP. More generally, we might refer to *quandries*, that is, unsettled dispositions to act. Corresponding to *projectors* into *subspaces* we have *decisions*. Often the subspaces are one-dimensional, that is, *rays*.

G.2.b Happiness example

Consider asking a person whether they are happy or not. Before asking the question, they might be in an indefinite (superposition) state (Fig. III.45(a)):

$$|\psi\rangle = a|\text{happy}\rangle + b|\text{unhappy}\rangle.$$

It is not just that we do not know whether the person is happy or not; rather the person "is in an indefinite state regarding happiness, simultaneously entertaining both possibilities, but being uncommitted to either" (Pothos & Busemeyer, 2013). More realistically "happy" and "unhappy" are likely to

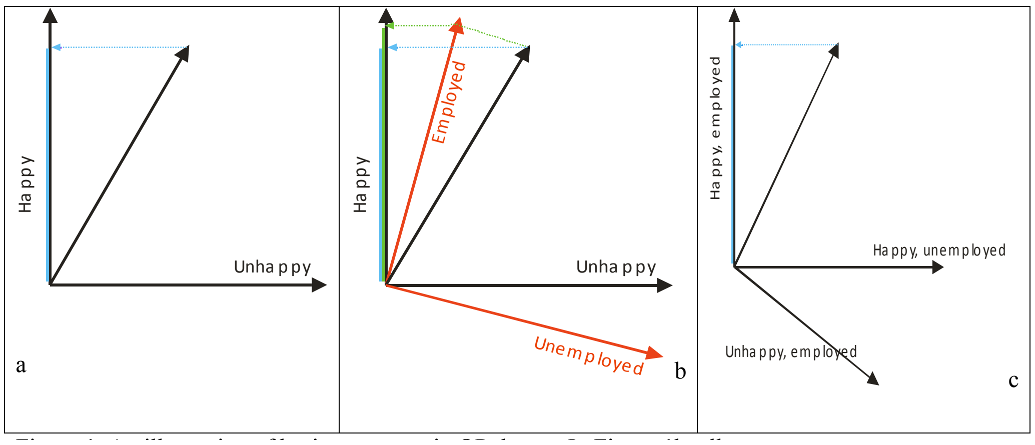


Figure III.45: “An illustration of basic processes in QP theory. In Figure [b], all vectors are coplanar, and the figure is a two-dimensional one. In Figure [c], the three vectors ‘Happy, employed’, ‘Happy, unemployed’, and ‘Unhappy, employed’ are all orthogonal to each other, so that the figure is a three-dimensional one. (The fourth dimension, ‘unhappy, unemployed’ is not shown).” (Pothos & Busemeyer, 2013)

be complex subspaces, not rays, but for the sake of the example, we use a 2D outcome space.

Asking the question is equivalent to measuring the state in the "happiness basis," which comprises two projectors P_{happy} and P_{unhappy}:

$$\begin{aligned} P_{\text{happy}} &= |\text{happy}\rangle\langle\text{happy}|, \\ P_{\text{unhappy}} &= |\text{unhappy}\rangle\langle\text{unhappy}|. \end{aligned}$$

The probability that the person responds "happy" is, as expected:

$$\|P_{\text{happy}}|\psi\rangle\|^2 = \||\text{happy}\rangle\langle\text{happy} \mid \psi\rangle\|^2 = |a|^2.$$

Measurement (decision) collapses the indefinite state to a definite basis state, $|\text{happy}\rangle$, with probability $|a|^2$. The judgment or decision is not just a "read out"; it is *constructed* from the state and the question, which actively disambiguates the superposition state.

G.2.c Incompatibility

As in quantum mechanics, questions can be *compatible* or *incompatible.* In fact, Neils Bohr borrowed the notion of *incompatible questions* from the psychologist William James. Compatible questions can be asked in any order; they commute; incompatible questions do not commute.

In CP it is always possible to specify a joint probability distribution over the four possible pairs of answers (the *unicity principle*). In QP you can do this for compatible questions, but not for incompatible ones. Psychologically, in the incompatible case the person cannot form a single thought for all combinations of possible outcomes (because they are linearly dependent). In the incompatible case, asking the first question alters the *context* of the second question, and thus affects its answer. Therefore, in applying QP in psychology, we can ask whether one decision is likely to affect the other.

Suppose we are going to ask a person two questions, whether they are happy or not and whether they are employed or not. It is plausible that happiness and employment are related, so we postulate a single 2D space spanned by both bases (Fig. III.45(b)). The angle between the two bases reflects the fact that happiness is likely to be correlated to employment. Notice that once we get an answer regarding happiness, we will be in an indefinite state regarding employment, and vice versa.

Suppose we ask if the subject is employed and then ask if they are happy. The probability that they answer "yes" to both is given by:[30]

$$\mathcal{P}\{\text{employed \&\& happy}\} = \mathcal{P}\{\text{employed}\} \times \mathcal{P}\{\text{happy} \mid \text{employed}\}.$$

The rules of QP give the first probability: $\mathcal{P}\{\text{employed}\} = \|P_{\text{employed}}|\psi\rangle\|^2$. Asking about employment has collapsed the state, which is now

$$|\psi_{\text{employed}}\rangle = \frac{P_{\text{employed}}|\psi\rangle}{\|P_{\text{employed}}|\psi\rangle\|}.$$

The probability of a happy response is then

$$\mathcal{P}\{\text{happy} \mid \text{employed}\} = \|P_{\text{happy}}|\psi_{\text{employed}}\rangle\|^2.$$

Hence the probability of the two responses is

$$\mathcal{P}\{\text{employed \&\& happy}\} = \|P_{\text{happy}}P_{\text{employed}}|\psi\rangle\|^2.$$

From this example, we can see that the law for conditional probability in QP, called *Lüder's Law*, is:

$$\mathcal{P}\{A \mid B\} = \frac{\|P_A P_B|\psi\rangle\|^2}{\|P_B|\psi\rangle\|^2} = \frac{\mathcal{P}\{B \text{ \&\& } A\}}{\mathcal{P}\{B\}}.$$

Look at Fig. III.45(b). You can see that

$$\mathcal{P}\{\text{happy}\} < \mathcal{P}\{\text{employed \&\& happy}\},$$

which cannot happen in CP (since $\mathcal{P}\{A\} \geq \mathcal{P}\{A \wedge B\}$ always). The psychological interpretation would be that the subject's consciousness of being employed makes her more likely to say she is happy. This is because happiness and employment are correlated, but this correlation does not affect the outcome without the prior question about employment. In general, $\mathcal{P}\{A \text{ \&\& } B\} \neq \mathcal{P}\{B \text{ \&\& } A\}$, which cannot happen in CP. That is, conjunction is not commutative. You can see

$$\mathcal{P}\{\text{happy \&\& employed}\} < \mathcal{P}\{\text{employed \&\& happy}\}.$$

This is because the subject was more uncertain about their happiness than their employment, and therefore the state vector lost a lot of its amplitude

[30]As in C++, " && " should be read "and then" (sequential "and").

via its projection first onto $|\text{happy}\rangle$. "The size of such angles and the relative dimensionality of the subspaces are the cornerstones of QP cognitive models and are determined by the known psychology of the problem. These angles (and the initial state vector) have a role in QP theory analogous to that of prior and conditional distributions in Bayesian modeling." (Pothos & Busemeyer, 2013)

G.2.d Compatible questions

Fig. III.45(c) displays the case where the questions are compatible (only three of the four basis vectors are shown). In this case we have a tensor product between the space spanned by $\{|\text{happy}\rangle, |\text{unhappy}\rangle\}$ and the space spanned by $\{|\text{employed}\rangle, |\text{unemployed}\rangle\}$. For compatible questions the states are composite vectors, e.g.,

$$\begin{aligned}
|\text{H}\rangle &= \eta|\text{happy}\rangle + \eta'|\text{unhappy}\rangle, \\
|\text{E}\rangle &= \epsilon|\text{employed}\rangle + \epsilon'|\text{unemployed}\rangle, \\
|\Psi\rangle &= |\text{H}\rangle \otimes |\text{E}\rangle \\
&= \eta\epsilon|\text{happy}\rangle|\text{employed}\rangle + \eta\epsilon'|\text{happy}\rangle|\text{unemployed}\rangle \\
&\quad + \eta'\epsilon|\text{unhappy}\rangle|\text{employed}\rangle + \eta'\epsilon'|\text{unhappy}\rangle|\text{unemployed}\rangle.
\end{aligned}$$

Then, for example, as in CP the joint probability

$$\mathcal{P}\{\text{happy} \wedge \text{employed}\} = |\eta\epsilon|^2 = \mathcal{P}\{\text{happy}\}\mathcal{P}\{\text{employed}\}.$$

G.2.e Structured representations and entanglement

Many concepts seem to have *structured representations*, that is, components, properties, or attributes,, which are "aligned" when concepts are compared.[31] Structured concepts are naturally represented in QP by tensored spaces representing the concept's components. However QP also permits entangled (non-product) states, such as

$$\alpha|\text{happy}\rangle|\text{employed}\rangle + \beta|\text{unhappy}\rangle|\text{unemployed}\rangle.$$

This represents a state in which happiness and employment are strongly interdependent. It represents a stronger degree of dependency than can be expressed in CP. In CP you can construct a complete joint probability out of pairwise joints, but this is not possible in QP.

[31] Think of the variable components (fields) of a C++ class.

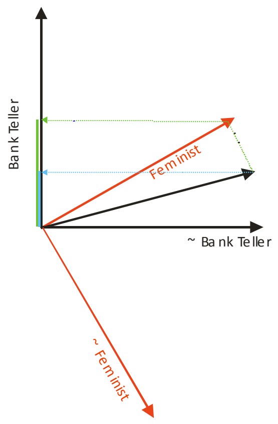


Figure III.46: Hypothetical basis state space of the "Linda experiment." (Pothos & Busemeyer, 2013)

G.2.f Time evolution

Time evolution is CP is defined by "a transition matrix (the solution to Kolmogorov's forward equation)" (Pothos & Busemeyer, 2013). It transforms the probabilities without violating the law of total probability. In QP amplitudes change by a unitary transformation.

G.3 Experimental evidence

The "Linda experiment." In 1983 Tversky and Kahneman reported on experiments in which subjects read a description of a hypothetical person named Linda that suggested she was a feminist. Subjects were asked to compare the probability of two statements: "Linda is a bank teller" (extremely unlikely given Linda's description), and "Linda is a bank teller and a feminist." Most subjects concluded:

$$\mathcal{P}\{\text{bank teller}\} < \mathcal{P}\{\text{bank teller} \wedge \text{feminist}\},$$

which violates CP; it is an example of the *conjunction fallacy*. Many experiments of this sort have shown that everyday reasoning commits this fallacy. Tversky and Kahneman proposed that people use heuristics rather than formal CP, but it can also be explained by QP.

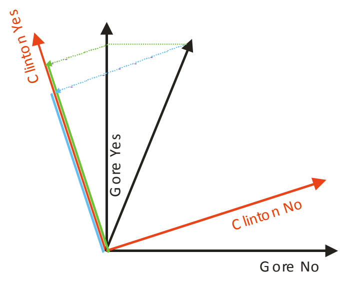


Figure III.47: Example of order effects in Gallop polls. [fig. from PB]

The QP explanation is as follows. We suppose that the written description makes it a priori likely that Linda is a feminist and unlikely that she is a bank teller; these priors are depicted in Fig. III.46. However, notice that being a feminist is largely independent of being a bank teller. In making a judgment like "Linda is a bank teller and a feminist" it is supposed that it is a sequential conjunction, with the most likely judgment evaluated first, in this case, "feminist && bank teller." Look at the figure. The green projection onto $|\text{feminist}\rangle$ and then onto $|\text{bank teller}\rangle$ is longer than the blue projection directly onto $|\text{bank teller}\rangle$. Projection can be thought of as an *abstraction* process, and so the projection of Linda onto $|\text{feminist}\rangle$ throws away details about her (it stereotypes her, we might say), and makes it more likely that she is a bank teller (since there is not a strong correlation between feminists and bank tellers). This may be compared to decoherence and loss of information in a quantum system. "In general, QP theory does not always predict an overestimation of conjunction. However, given the details of the Linda problem, an overestimation of conjunction necessarily follows. Moreover, the same model was able to account for several related empirical findings, such as the disjunction fallacy, event dependencies, order effects, and unpacking effects. . . " (Pothos & Busemeyer, 2013)

G.3.a Failure of commutativity

The "Clinton-Gore experiment." A Gallup poll asked "Is Clinton honest?" and "Is Gore honest?" Results depended on the order in which they were asked:

order	Clinton	Gore
Clinton — Gore	50%	68%
Gore — Clinton	57%	60%

This is also a common characteristic of everyday judgment; it is also common in the assessment of evidence for a hypothesis. QP explains this as follows. The "Yes" basis vectors have a smaller angle reflecting an expected correlation between the answers (since Clinton and Gore ran together). The initial state vector is a little closer to the $|\text{Gore Yes}\rangle$ vector reflecting the assumption that Gore's honesty is a priori more likely than Clinton's. You can see this by looking at the green projection onto $|\text{Gore Yes}\rangle$, which is longer than its blue projection onto $|\text{Clinton Yes}\rangle$. Note further that the two-step blue projection onto $|\text{Clinton Yes}\rangle$ is longer than the direct projection onto it. That is, judging Gore to be honest increases the probability of also judging Clinton to be honest.

G.3.b Violations of the sure-thing principle

"The sure thing principle is the expectation that human behavior ought to conform to the law of total probability." (Pothos & Busemeyer, 2013) In 1992 Shafir and Tversky reported experiments showing violations of the sure-thing principle in the *one-shot prisoner's dilemma*: The subject has to decide whether to cooperate or defect, as does their opponent. This is a typical payoff matrix; it shows the payoff for you and your opponent for each pair of choices:

	opponent	
↓ you	cooperate	defect
cooperate	3, 3	0, 5
defect	5, 0	1, 1

If you are told what your opponent is going to do, then you should defect. This is what subjects usually do. If you don't know, then the optimal strategy is still to defect. This is the "sure thing": you should defect in either case. However, some subjects decide to cooperate anyway (thus violating the sure-thing principle). One explanation is "wishful thinking." If you have a bias toward cooperation, you might suppose (in the absence of evidence) that your opponent has a similar bias.

The QP explanation is as follows. Suppose $|\psi_C\rangle$ and $|\psi_D\rangle$ are the states of knowing that your opponent will cooperate and defect, respectively. Suppose

P_C and P_D are projections representing your decision to cooperate or defect. Under the condition where you know what your opponent is going to do, the probability of you deciding to defect in the two cases is:

$$\begin{aligned} \mathcal{P}\{\text{you defect}\} &= \|P_D|\psi_C\rangle\|^2, \\ \mathcal{P}\{\text{you defect}\} &= \|P_D|\psi_D\rangle\|^2. \end{aligned}$$

In the unknown condition, we can suppose the state is $|\psi\rangle = \frac{1}{\sqrt{2}}(|\psi_C\rangle + |\psi_D\rangle)$. Hence, in this case the probability of you deciding to defect is:

$$\begin{aligned} \mathcal{P}\{\text{you defect}\} &= \left\| \frac{1}{\sqrt{2}}(P_D|\psi_C\rangle + P_D|\psi_D\rangle) \right\|^2 \\ &= \frac{1}{2}(\langle\psi_C| + \langle\psi_D|)P_D^\dagger P_D(|\psi_C\rangle + |\psi_D\rangle) \\ &= \frac{1}{2}\|P_D|\psi_C\rangle\|^2 + \frac{1}{2}\|P_D|\psi_D\rangle\|^2 + \langle\psi_D \mid P_D^\dagger P_D \mid \psi_C\rangle. \end{aligned}$$

The interference term $\langle\psi_D \mid P_D^\dagger P_D \mid \psi_C\rangle$ could be positive or negative, in the latter case decreasing the probability below unity.

G.3.c Asymmetric similarity

In 1977 Tversky showed that similarity judgments violate metric axioms, in particular, symmetry.

China-Korea experiment. For example, North Korea was judged more similar to China, than China was judged to be similar to North Korea:

$$\text{Sim}(\text{North Korea}, \text{China}) > \text{Sim}(\text{China}, \text{North Korea}).$$

The QP explanation is that concepts correspond to subspaces of various dimensions, with the dimension of the subspace roughly corresponding to the number of known properties of the concept (i.e., how much someone knows about it). The judgment of the similarity of A to B is modeled by the projection of the initial state into A and then into B. It's assumed that the initial state is neutral with respect to A and B (i.e., the subject hasn't been thinking about either). If $|\psi\rangle$ is the initial state, then

$$\text{Sim}(A, B) = \|P_B P_A|\psi\rangle\|^2 = \mathcal{P}\{A \mathbin{\&\&} B\}.$$

The subjects in this case are assumed to be more familiar with China than with North Korea, so the China subspace is larger (see Fig. III.48). When

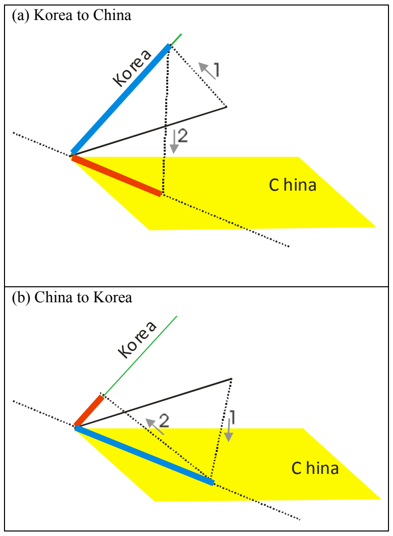


Figure III.48: QP model of China – (North) Korea experiment. [fig. from PB]

North Korea is compared to China, more of its amplitude is retained by the final projection into the higher dimensional subspace corresponding to China: Fig. III.48(a). In the opposite case, the projection into the lower dimensional North Korea subspace loses more amplitude: Fig. III.48(b). This is not universally true.

G.4 Cognition in Hilbert space

Pothos & Busemeyer (2013) defend the application of QP in a function-first or top-down approach to modeling cognition.[32] This is done by postulating vectors in a low-dimensional space. I argue that consideration of the high-dimensional complex-valued wavefunction underlying the state vector will expand the value of QP in cognitive science.

[32]Material in this section is adapted from MacLennan (2013), my commentary on Pothos & Busemeyer (2013).

G.4.a QM premises

To this end, application of QP in cognitive science would be aided by importing two premises from quantum mechanics:

The first premise is that the fundamental reality is the wavefunction. In cognitive science this corresponds to postulating a spatially-distributed pattern of neural activity as the elements of the cognitive state space. Therefore the basis vectors used in QP are in fact basis functions for an infinite (or very high) dimensional Hilbert space.

The second important fact is that the wavefunction is complex-valued and that wavefunctions combine with complex coefficients. This is the main reason for interference and other non-classical properties. The authors acknowledge this, but do not make explicit use of complex numbers in the target article.

G.4.b Possible neural substrates

What is the analog of the complex-valued wavefunction in neurophysiology? There are several possibilities, but perhaps the most obvious is the distribution of neural activity across a region of cortex; even a square millimeter of which can have hundreds of thousands of neurons. The dynamics will be defined by a time-varying Hamiltonian, with each eigenstate being a spatial distribution of neurons firing at a particular rate. The most direct representation of the magnitude and phase (or argument) of a complex quantity is frequency and phase of neural impulses.

G.4.c Projection

Possible neural mechanisms: Pothos & Busemeyer (2013) specify that a judgment or decision corresponds to measurement of a quantum state, which projects it into a corresponding subspace, but it is informative to consider possible mechanisms. For example, the need to act definitely (such as coming to a conclusion in order to answer a question) can lead to mutually competitive mechanisms, such as among the minicolumns in a macrocolumn, which creates dynamical attractors corresponding to measurement eigenspaces. Approach to the attractor amplifies certain patterns of activity at the expense of others. Orthogonal projectors filter the neural activity and win the competition with a probability proportional to the squared amplitude of the patterns to which they are matched. (In the case where the phases of neural impulses encode complex phases, matching occurs when the phases are delayed in such a way that the impulses reinforce.) The winner positively reinforces its matched signal and the loser negatively reinforces the signal to which it is matched. Regardless of mechanism, during collapse the energy of the observed eigenstate of the decision (measurement) operator receives the energy of the orthogonal eigenstates (this is the effect of renormalization). The projection switches a jumble of frequencies and phases into a smaller, more coherent collection, corresponding to the answer (observed) eigenspace.

No inherent bases: The target article suggests that a QP model of a process begins by postulating basis vectors and qualitative angles between alternative decision bases (significantly, only real rotations are discussed). As a consequence, a QP model is treated as a low-dimensional vector space. This is a reasonable, top-down strategy for defining a QP cognitive model, but it can be misleading. There is no reason to suppose that particular decision bases are inherent to a cognitive Hilbert space. There may be a small number of "hard-wired" decisions, such as fight-or-flight, but the vast majority are learned. Certainly this is the case for decisions corresponding to lexical items such as (un-)happy and (un-)employed.

Creation/modification of observables: Investigation of the dynamics of cognitive wavefunction collapse would illuminate the mechanisms of decision making but also of the processes by which observables form. This would allow modeling changes in the decision bases, either temporary through context effects or longer lasting through learning. Many decision bases are ad hoc, as when we ask, "Do you admire Telemachus in the *Odyssey*?" How such ad hoc projectors are organized requires looking beneath a priori state basis vectors to the underlying neural wavefunctions and the processes shaping them.

G.4.d Incompatible decisions

The commutator and anti-commutator: In quantum mechanics the uncertainty principle is a consequence of non-commuting measurement operators, and the degree of non-commutativity can be quantified (see Sec. B.7, p. 94). Two measurement operators P and Q commute if $PQ = QP$, that is, if the operator $PQ - QP$ is identically $\mathbf{0}$. If they do not commute, then $PQ - QP$ measures the degree of non-commutativity. This is expressed in quantum mechanics by the "commutator" $[P, Q] = PQ - QP$. It is relatively easy to show that this implies an uncertainty relation: $\Delta P \, \Delta Q \geq |\langle [P,Q] \rangle|$. That is, the product of the uncertainties on a state is bounded below by the absolute mean value of the commutator on the state. Suppose H is a measurement that returns 1 for $|\text{happy}\rangle$ and 0 for $|\text{unhappy}\rangle$, and E is a measurement that returns 1 for $|\text{employed}\rangle$ and 0 for $|\text{unemployed}\rangle$. If $|\text{employed}\rangle = a|\text{happy}\rangle + b|\text{unhappy}\rangle$, then the commutator is

$$[H,E] = ab \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

The absolute mean value of this commutator (applied to a state) gives a minimum joint uncertainty. If we could measure $[P,Q]$ for various pairs of questions, P and Q, we could make quantitative empirical predictions of the joint uncertainty in decisions.

Might we design experiments to measure the commutators and so quantify incompatibility among decisions? Certainly there are difficulties, such as making independent measurements of both PQ and QP for a single subject, or accounting for intersubject variability in decision operators. But making such measurements would put more quantitative teeth into QP as a cognitive model.

G.4.e Suggestions

Pothos & Busemeyer (2013) do an admirable job of defending QP as a fruitful top-down model of decision making, but I believe it would be more valuable if it paid greater attention to the complex-valued wavefunction that underlies QP in both quantum mechanics and cognition. This would allow a more detailed account of the origin of interference effects and of the structure of both learned and ad hoc decision operators. Finally, the treatment of incompatible decisions can be made more rigorous by treating them quantitatively as noncommuting operators.

G.5 Conclusions

You might wonder why it is so important to understand the less-then-perfect inferential abilities of humans. There are at least two reasons, *scientific* and *technological.* First, it is important to understand human inference as both *pure* and *applied* science. It reveals much about our human nature, and specifically provides hints as to how the brain works. From a more applied perspective, we need to understand how humans determine their actions in order to predict (and even influence) human behavior. In terms of technology, it might seem that the last thing we might want to do would be to emulate in our machine intelligence the “imperfect, fallacious” reasoning of humans. It might be the case, however, that QP-based reasoning is better than CP for real-time purposeful action in natural, complex situations, where the premisses of CP are inaccurate.

H Exercises

Exercise III.1 Compute the probability of measuring $|0\rangle$ and $|1\rangle$ for each of the following quantum states:

1. $0.6|0\rangle + 0.8|1\rangle$.
2. $\frac{1}{\sqrt{3}}|0\rangle + \sqrt{2/3}|1\rangle$.
3. $\frac{\sqrt{3}}{2}|0\rangle - \frac{1}{2}|1\rangle$.
4. $-\frac{1}{25}(24|0\rangle - 7|1\rangle)$.
5. $-\frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\pi/6}}{\sqrt{2}}|1\rangle$.

Exercise III.2 Compute the probability of the four states if the following are measured in the computational basis:

1. $(e^i|00\rangle + \sqrt{2}|01\rangle + \sqrt{3}|10\rangle + 2e^{2i}|11\rangle)/\sqrt{10}$.
2. $\frac{1}{2}(-|0\rangle + |1\rangle) \otimes (e^{\pi i}|0\rangle + e^{-\pi i}|1\rangle)$.
3. $$(\sqrt{1/3}|0\rangle - \sqrt{2/3}|1\rangle) \otimes \sqrt{2}\left(\frac{e^{\pi i/4}}{2}|0\rangle + \frac{e^{\pi i/2}}{2}|1\rangle\right).$$

Exercise III.3 Suppose that a two-qubit register is in the state

$$|\psi\rangle = \frac{3}{5}|00\rangle - \frac{\sqrt{7}}{5}|01\rangle + \frac{e^{i\pi/2}}{\sqrt{5}}|10\rangle - \frac{2}{5}|11\rangle.$$

1. Suppose we measure just the first qubit. Compute the probability of measuring a $|0\rangle$ or a $|1\rangle$ and the resulting register state in each case.
2. Do the same, but supposing instead that we measure just the second qubit.

Exercise III.4 Prove that projectors are idempotent, that is, $P^2 = P$.

Exercise III.5 Prove that a normal matrix is Hermitian if and only if it has real eigenvalues.

Exercise III.6 Prove that $U(t) \overset{\text{def}}{=} \exp(-iHt/\hbar)$ is unitary.

Exercise III.7 Use spectral decomposition to show that $K = -i\log(U)$ is Hermitian for any unitary U, and thus $U = \exp(iK)$ for some Hermitian K.

Exercise III.8 Show that the commutators ($[L, M]$ and $\{L, M\}$) are bilinear (linear in both of their arguments).

Exercise III.9 Show that $[L, M]$ is anticommutative, i.e., $[M, L] = -[L, M]$, and that $\{L, M\}$ is commutative.

Exercise III.10 Show that $LM = \frac{[L,M]+\{L,M\}}{2}$.

Exercise III.11 In Sec. B.5 we proved the no-cloning theorem with single ancillary constant qubit. Prove that the cloning is still impossible if multiple ancillary qubits are provided. That is, show that we cannot have a unitary operator $U(|\psi\rangle \otimes |C\rangle) = |\psi\rangle|\psi\rangle \otimes |D\rangle$, where $|C\rangle$ is an $n > 1$ dimensional vector and $|D\rangle$ is an $n - 1$ dimensional vector. (Note that $|D\rangle$ might depend on $|\psi\rangle$.)

Exercise III.12 Show that the four Bell states are orthonormal (i.e., both orthogonal and normalized).

Exercise III.13 Prove that $|\beta_{11}\rangle$ is entangled.

Exercise III.14 Prove that $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ is entangled.

Exercise III.15 What is the effect of Y (imaginary definition) on the computational basis vectors? What is its effect if you use the real definition (C.2.a, p. 106)?

Exercise III.16 Prove that I, X, Y, and Z are unitary. Use either the imaginary or real definition of Y (C.2.a, p. 106).

Exercise III.17 What is the matrix for H in the *sign basis*?

Exercise III.18 Show that the X, Y, Z and H gates are Hermitian (their own inverses) and prove your answers. Use either the imaginary or real definition of Y (C.2.a, p. 106).

Exercise III.19 Prove the following useful identities:

$$HXH = Z, HYH = -Y, HZH = X.$$

Exercise III.20 Show (using the real definition of Y, C.2.a, p. 106):
$|0\rangle\langle 0| = \frac{1}{2}(I+Z), |0\rangle\langle 1| = \frac{1}{2}(X+Y), |1\rangle\langle 0| = \frac{1}{2}(X-Y), |1\rangle\langle 1| = \frac{1}{2}(I-Z)$.

Exercise III.21 Prove that the Pauli matrices span the space of 2×2 matrices.

Exercise III.22 Prove $|\beta_{xy}\rangle = (P \otimes I)|\beta_{00}\rangle$, where $xy = 00, 01, 11, 10$ for $P = I, X, Y, Z$, respectively.

Exercise III.23 Suppose that P is one of the Pauli operators, but you don't know which one. However, you are able to pick a 2-qubit state $|\psi_0\rangle$ and operate on it, $|\psi_1\rangle = (P \otimes I)|\psi_0\rangle$. Further, you are able to select a unitary operation U to apply to $|\psi_1\rangle$, and to measure the 2-qubit result, $|\psi_2\rangle = U|\psi_1\rangle$, in the computational basis. Select $|\psi_0\rangle$ and U so that you can determine with certainty the unknown Pauli operator P.

Exercise III.24 What is the matrix for CNOT in the standard basis? Prove your answer.

Exercise III.25 Show that CNOT does not violate the No-cloning Theorem by showing that, in general, CNOT$|\psi\rangle|0\rangle \neq |\psi\rangle|\psi\rangle$. Under what conditions does the equality hold?

Exercise III.26 What quantum state results from

$$\text{CNOT}(H \otimes I)\ \frac{1}{2}(c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle)?$$

Express the result in the computational basis.

Exercise III.27 Compute $(Y \otimes I)\text{CNOT}(H \otimes I)|00\rangle$. Show your work.

Exercise III.28

1. Compute $(H \otimes I \otimes I)(\text{CNOT} \otimes I)[(\frac{4}{5}|0\rangle + \frac{3}{5}|1\rangle) \otimes |\beta_{00}\rangle$.
2. Give the probabilities and resulting states for measuring the first two qubits in the computational basis.

3. Apply Z to the state resulting from measuring $|10\rangle$.

Exercise III.29 What is the matrix for CCNOT in the standard basis? Prove your answer.

Exercise III.30 Use a single Toffoli gate to implement each of NOT, NAND, and XOR.

Exercise III.31 Use Toffoli gates to implement FAN-OUT. FAN-OUT would seem to violate the No-cloning Theorem, but it doesn't. Explain why.

Exercise III.32 Design a quantum circuit to transform $|000\rangle$ into the entangled state $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$.

Exercise III.33 Show that $|+\rangle, |-\rangle$ is an ON basis.

Exercise III.34 Prove:

$$\begin{aligned}|0\rangle &= \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle),\\ |1\rangle &= \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle).\end{aligned}$$

Exercise III.35 What are the possible outcomes (probabilities and resulting states) of measuring $a|+\rangle + b|-\rangle$ in the *computational basis* (of course, $|a|^2 + |b|^2 = 1$)?

Exercise III.36 Prove that $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.

Exercise III.37 Prove:

$$\begin{aligned}H(a|0\rangle + b|1\rangle) &= a|+\rangle + b|-\rangle,\\ H(a|+\rangle + b|-\rangle) &= a|0\rangle + b|1\rangle.\end{aligned}$$

Exercise III.38 Prove $H = (X + Z)/\sqrt{2}$.

Exercise III.39 Prove Eq. III.18 (p. 112).

Exercise III.40 Show that three successive CNOTs, connected as in Fig. III.11 (p. 111), will swap two qubits.

Exercise III.41 Recall the conditional selection between two operators (C.3, p. 112): $|0\rangle\langle 0| \otimes U_0 + |1\rangle\langle 1| \otimes U_1$. Suppose the control bit is a superposition $|\chi\rangle = a|0\rangle + b|1\rangle$. Show that:

$$(|0\rangle\langle 0| \otimes U_0 + |1\rangle\langle 1| \otimes U_1)|\chi, \psi\rangle = a|0, U_0\psi\rangle + b|1, U_1\psi\rangle.$$

Exercise III.42 Show that the 1-bit full adder (Fig. III.15, p. 114) is correct.

Exercise III.43 Show that the operator U_f is unitary:

$$U_f|x, y\rangle \stackrel{\text{def}}{=} |x, y \oplus f(x)\rangle,$$

Exercise III.44 Verify the remaining superdense encoding transformations in Sec. C.6.a (p. 118).

Exercise III.45 Verify the remaining decoding cases for quantum teleportation Sec. C.6.b (p. 122).

Exercise III.46 Confirm the quantum teleportation circuit in Fig. III.21 (p. 123).

Exercise III.47 Complete the following step from the derivation of the Deutsch-Jozsa algorithm (Sec. D.1, p. 130):

$$H|x\rangle = \sum_{z \in \mathbf{2}} \frac{1}{\sqrt{2}}(-1)^{xz}|z\rangle.$$

Exercise III.48 Show that $\text{CNOT}(H \otimes I) = (I \otimes H)C_Z H^{\otimes 2}$, where C_Z is the controlled-Z gate.

Exercise III.49 Show that the Fourier transform matrix (Eq. III.25, p. 138, Sec. D.3.a) is unitary.

Exercise III.50 Prove the claim on page 154 (Sec. D.4.b) that D is unitary.

Exercise III.51 Prove the claim on page 154 (Sec. D.4.b) that

$$WR'W = \begin{pmatrix} \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \end{pmatrix}.$$

Exercise III.52 Show that if there are s solutions x such that $P(x) = 1$, then $\frac{\pi\sqrt{N/s}}{4}$ is the optimal number of iterations in Grover's algorithm.

Exercise III.53 Design a quantum gate array for the following syndrome extraction operator (Sec. D.5.d, p. 164):

$$S|x_3, x_2, x_1, 0, 0, 0\rangle \overset{\text{def}}{=} |x_3, x_2, x_1, x_1 \oplus x_2, x_1 \oplus x_3, x_2 \oplus x_3\rangle.$$

Exercise III.54 Design a quantum gate array to apply the appropriate error correction for the extracted syndrome as given in Sec. D.5.d, p. 164:

bit flipped	syndrome	error correction
none	$\|000\rangle$	$I \otimes I \otimes I$
1	$\|110\rangle$	$I \otimes I \otimes X$
2	$\|101\rangle$	$I \otimes X \otimes I$
3	$\|011\rangle$	$X \otimes I \otimes I$

Exercise III.55 Design encoding, syndrome extraction, and error correction quantum circuits for the code $|0\rangle \mapsto |+++\rangle$, $|1\rangle \mapsto |---\rangle$ to correct single phase flip (Z) errors.

Exercise III.56 Prove that $A_a A_a = \mathbf{1}$ (Sec. F.1.b).

Exercise III.57 Prove that $A_{ab,c} = \mathbf{1} + a^\dagger a b^\dagger b(c + c^\dagger - \mathbf{1}) = \mathbf{1} + N_a N_b (A_c - \mathbf{1})$ is a correct definition of CCNOT by showing how it transforms the quantum register $|a, b, c\rangle$ (Sec. F.1.b).

Exercise III.58 Show that the following definition of Feynman's switch is unitary (Sec. F.1.b):

$$q^\dagger c p + r^\dagger c^\dagger p + p^\dagger c^\dagger q + p^\dagger c r.$$

Chapter IV

Molecular Computation

A Basic concepts

A.1 Introduction to molecular computation

Molecular computation uses molecules to represent information and molecular processes to implement information processing. DNA computation is the most important (and most explored) variety of molecular computation, and it is the principal topic of this chapter. The principal motivation is that molecular computation is massively parallel (*molar* or *Avogadro-scale* parallelism, that is, on the order of 10^{26}). Data representations are also very dense (on the order of nanometers), and molecular computation can be very efficient in terms of energy.

A.2 DNA basics

The DNA molecule is a polymer (*poly* = many, *mer* = part), a sequence of *nucleotides*. A DNA nucleotide has three parts: a deoxyribose sugar, a phosphate group, and a nucleobase (C = cytosine, G = guanine, A = adenine, T= thymine).[1] See Fig. IV.1 for deoxyribose. The *DNA backbone* is a

[1]In RNA, uracil replaces thymine.

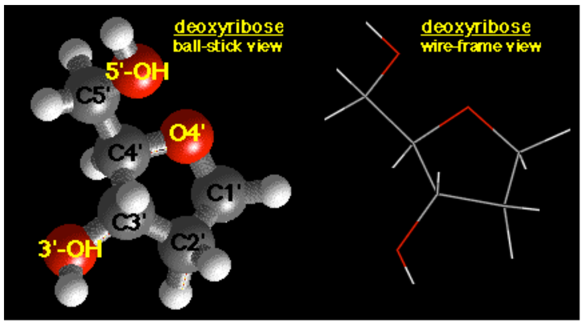


Figure IV.1: Deoxyribose. [source: wikipedia]

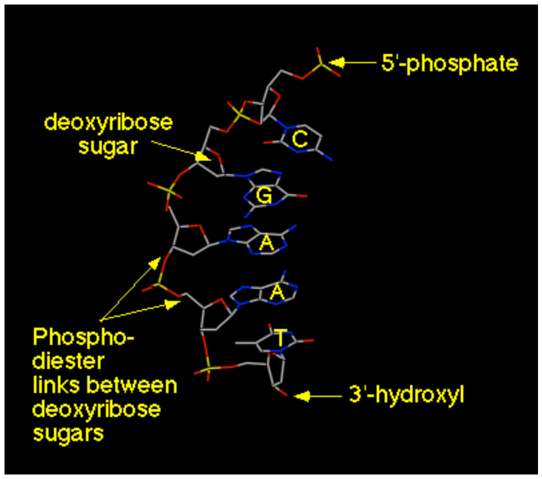


Figure IV.2: DNA backbone. [source: wikipedia]

sequence of deoxyribose sugars alternating with phosphate groups connected by covalent *phosphodiester* bonds (Fig. IV.2). The backbone connects the hydroxyl group on the $3'$ carbon (the "$3'$-hydroxyl group") of one sugar to the $5'$-hydroxyl of the next. In this way we distinguish the $5'$ and $3'$ ends of a polynucleotide: $5'$ has a terminal phosphate group, $3'$ a terminal hydroxl group. We generally write the bases in the $5'$ to $3'$ order.

Adenine and thymine each have two hydrogen-bonding sites and can bind together; similarly, guanine and cytosine each have three hydrogen-bonds and can bind together. This results in *Watson-Crick complementarity*, which means that two complementary polynucleotides can bind together. This can occur only if the two single strands are *antiparallel*, that is, run in opposite directions ($3'$ to $5'$ versus $5'$ to $3'$); see Fig. IV.3. Since we write the *sense-strand* in the $5'$ to $3'$ order, we write the *antisense-strand* in the $3'$ to $5'$ order. Complementary single stranded DNA can hybridize, leaving *sticky ends* (unmatched single strands) at either or both ends. They are called "sticky" because they are available for bonding by complementary sequences.

Fig. IV.4 gives some idea what the actual double helix looks like. It is worth keeping in mind that it is a very complex organic molecule, not a string of the letters A, C, T, G. For example, in addition to the double helix formed by the backbones of the two strands, there are two groves between the strands (Fig. IV.5). They are adjacent to the bases and therefore provide binding sites for proteins. Because the backbones are not symmetrically placed, the grooves have two sizes: *minor* (1.2nm) and *major* (2.2nm). The latter provides the best binding site.

To give some sense of the scale, here are some numbers. The nucleotides are about 3.3nm long. The chain is 2.2 to 2.6nm wide. The radius of the coil is about 1nm. The longest human chromosome (chromosome 1) is about 220×10^6bp (base pairs) long.

DNA data storage: In Aug. 2012 a team at Harvard reported converting a 53,000 word book to DNA and then reading it out by DNA sequencing.[2] This is the densest consolidation of data in any medium (including flash memory, but also compared to experimental media, such as quantum

[2]Church, George M, Gao, Yuan, and Kosuri, Sriram. "Next-generation digital information storage in DNA." *Science* **337** (28 Sept. 2012): 1628. See also *Science* (Aug. 16, 2012), `DOI: 10.1126/science.1226355` (accessed 2012-08-24). See also `http://spectrum.ieee.org/biomedical/imaging/reading-and-writing-a-book-with-dna/` (accessed 2012-08-24).

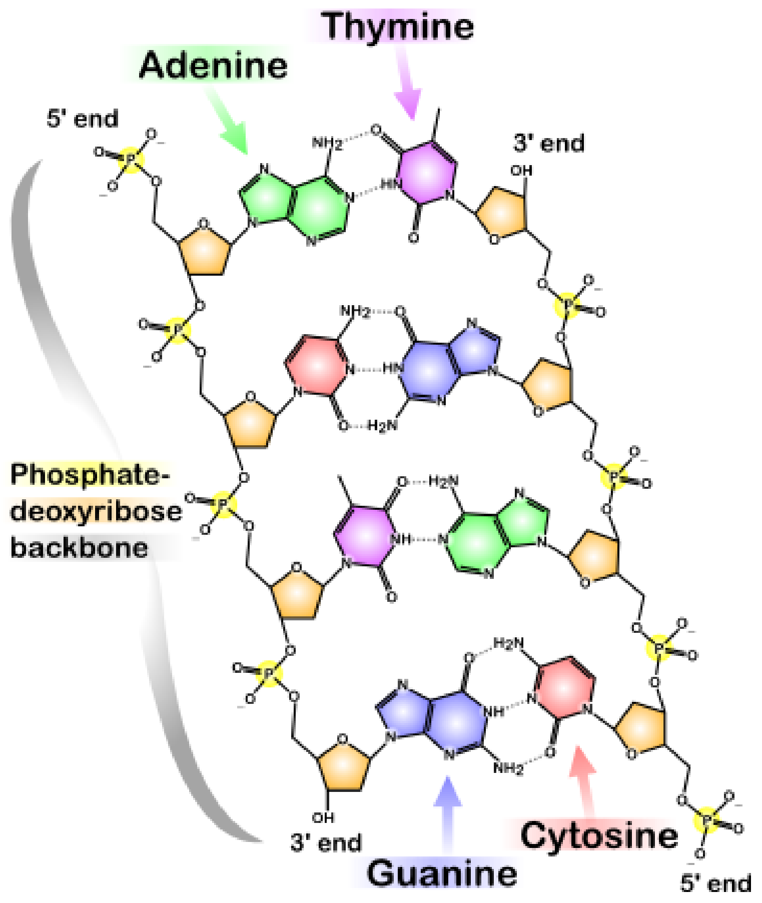


Figure IV.3: Watcon-Crick complementarity. Complementary H-bonds allow guanine to bind to cytosine and adenine to bind to thymine. [source: wikimedia commons]

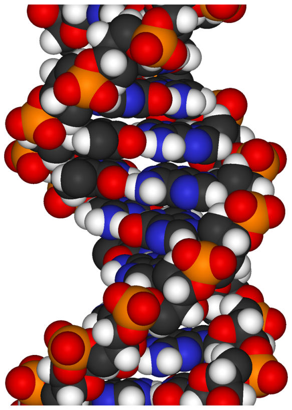


Figure IV.4: [source: wikimedia commons]

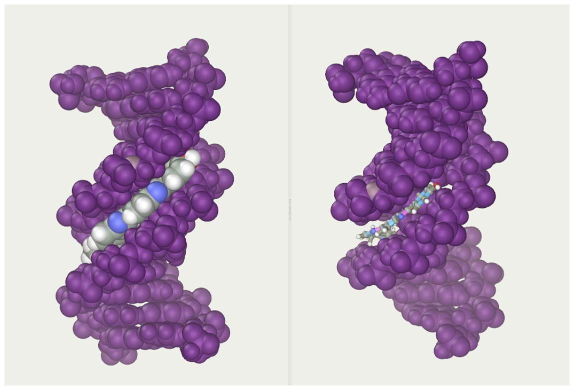


Figure IV.5: Major and minor grooves. [source: wikimedia commons]

holography). The book included 11 jpegs and a javascript program, for a total of 5.27 Mbits, and the decoded version had only 10 incorrect bits. The encoded book could be reproduced by DNA replication. This is a storage density of 5.5 petabits/mm^3 = 5.5×10^{15}bits/mm^3. The authors estimate that DNA can encode 455 exabytes (4.55×10^{20} bytes) per gram of single-stranded DNA. (This is based on 2 bits per nucleotide, and doesn't take redundancy, error correction, etc., into account, which would decrease the density by several orders of magnitude.[3]). In Jan. 2013 *Nature*, it was reported that 5 Mbits had been encoded and perfectly decoded (99.99 to 100% accuracy with quadruple redundancy. It cost $12,400 for the encoding and $220 for retrieval. The cost of DNA synthesis has been decreasing about 5× per year, and the cost of sequencing by about 12× per year, so it could become economical for archival storage. Enclosed in silica glass spheres, DNA storage could last for a million years.

A.3 DNA manipulation

There are several standard procedures for manipulating DNA that have been applied to DNA computing, which has benefitted from well-developed laboratory procedures and apparatus for handling DNA. The following sections review these procedures briefly, omitting details not directly relevant to DNA computation.

A.3.a Denaturation

Denaturation causes DNA double strands to unwind and separate into its two single strands. This occurs by breaking the H-bonds between the bases, which are weak compared to covalent bonds (e.g., those in the backbone). Thermal denaturing (heating to about 95°C) is the most common procedure, but there are also chemical denaturing agents.

A.3.b Hybridization

Hybridization is the process by which two DNA single strands combine in a sequence specific way through the formation of non-covalent H-bonds between the bases. *Annealing* is used to achieve an energetically favorable result (a minimal number mismatches).

[3]Church, *op. cit.*, Supplementary Material

A.3.c Polymerase extension

Enzymes called *polymerases* (polymerizing enzymes) add nucleotides one at a time to the $3'$ end of a DNA molecule, an operation called *polymerase extension.* It can be used to fill in a sticky end, or a short sequence bound to a complementary sequence can operate as a *primer* causing the rest of the sequence to be filled in.

A.3.d Ligation

Hybridization links to strands together by H-bonds, but there is a *nick* in the DNA backbone resulting from a missing phosphodiester bond. Certain enzymes will *ligate* the strands by filling in the missing covalent bonds, making a more stable strand.

A.3.e Nuclease degradation

Nucleases are enzymes that remove nucleotides from a DNA strand. *Exonucleases* remove nucleotides from the ends (either $3'$ or $5'$ depending on the exonuclease), whereas *endonucleases* cut DNA strands (single or double) at specific places. *Restriction enzymes* are the most common endonucleases. They operate at specific *restriction sites*, typically 4 to 8 bases long. The cut can be *blunt* or *staggered* leaving sticky ends.

A.3.f Synthesis

Synthesis is the process of assembling single-stranded *oligonucleotides* ("oligos"), which are short strands with a desired sequence. Assembly occurs in the $3'$ to $5'$ direction.The process is completely automated nowadays, but due to the accumulation of errors, the current limit is about 200 nucleotides.

A.3.g Polymerase chain reaction

The *Polymerase Chain Reaction* (PCR) is a method for exponentially amplifying the concentration of a selected DNA sequence in a sample. It was a breakthrough and remains one of the most important processes in practical DNA technology. Here is the basic procedure.

The sequence to be replicated is identified by a known short sequence (about 20 bases) at the $3'$ ends of both its sense and antisense strands. Short

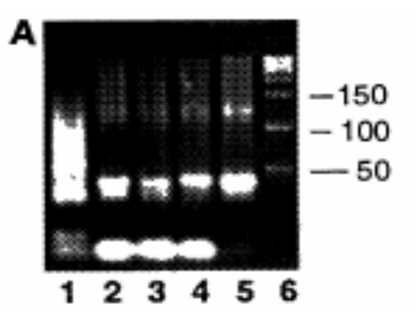


Figure IV.6: Example of result of gel electrophoresis.

DNA strands called *primers*, which are complementary to the identifying sequences, are synthesized. That is, two primers are used to bound or delimit the sequence of interest (at the $3'$ end of the sense strand and the $3'$ end of the antisense strand). (You can also have primers that bind to the $5'$ ends of the target sequence, but $3'$ is typical.)

Step 1 (add primers): The primers and DNA polymerase are added to the DNA sample (possibly containing the desired sequence and many others).

Step 2 (denaturation): The mixture is heated to about 95°C for about 30 seconds, which causes all the double strands to separate.

Step 3 (annealing): The mixture is cooled to about 68°C for a minute, which allows the primers to hybridize with the other strands.

Step 4 (polymerase extension): The mixture is warmed to about 72°C for 30 seconds or more (depending on the length of the sequence to be replicated, which goes at about about 1000 bp/min.). This allows the polymerase to extend the primer on each single strand so that it becomes a double strand. Pairs of long strands can rehybridize, but the lighter oligonucleotides tend to get there faster, so instead we get two double strands. As a result, the concentration of the desired DNA sequence has doubled.

Step 5 (repeat): Repeat Steps 2 to 4 until the desired concentration is reached. The concentration of the desired sequence increases exponentially, 2^n, in the number of cycles. There are "PCR machines" that automate the entire PCR process.

A.3.h Gel electrophoresis

Gel electrophoresis is a method of determining the size of DNA fragments in a sample. The sample is put in a well at one end of a layer of gel, and a positive electric field is applied at the other end of the gel. Since DNA is negatively charged, it migrates through the gel, but smaller fragments migrate faster. Later the DNA can be stained and the gel imaged to determine the relative concentration of fragments of different sizes in the sample. The resulting pattern is called a *DNA ladder*; see Fig. IV.6 for several "lanes" of DNA ladders.

A.3.i Filtering

There are several ways to filter DNA samples for specific sequences. One way to select for an oligonucleotide o in a sample S is to attach the complementary sequence $\overline{o}$ to a filter. Then when S goes through the filter, the o in it will hybridize with the $\overline{o}$ in the filter, and stay there; what passes through is $S - o$. If desired, the $o\overline{o}$ on the filter can be denatured to separate the strands containing o from the $\overline{o}$ on the filter.

A.3.j Modification

Certain enzymes can be used to change subsequences of a DNA sequence.

A.3.k Sequencing

Sequencing is the process of reading out the sequence of a DNA molecule. This is traditionally accomplished by extension of a primed template sequence, but a variety of other techniques have been developed since the 1970s. The latest techniques do massively parallel sequencing of many fragments.

B Filtering models

Filtering models, an important class of DNA algorithms, operate by filtering out of the solution molecules that are not part of the desired result. A chemical solution can be treated mathematically as a finite *bag* or *multi-set* of molecules, and filtering operations can be treated as operations to produce multi-sets from multi-sets. Typically, for a problem of size n, strings of size $\mathcal{O}(n)$ are required. The chemical solution should contain enough strings to include many copies all possible answers. Therefore, for an exponential problem, we will have $\mathcal{O}(k^n)$ strings. Filtering is essentially a brute-force method of solving problems.

B.1 Adleman: HPP

B.1.a Review of HPP

Leonard Adleman's solution of the Hamiltonian Path Problem was the first successful demonstration of DNA computing. The *Hamiltonian Path Problem* (HPP) is to determine, for a given directed graph $G = (V, E)$ and two of its vertices $v_{\text{in}}, v_{\text{out}} \in V$, whether there is a *Hamiltonian path* from v_{in} to v_{out}, that is, a path that goes through each vertex exactly once. HPP is an NP-complete problem, but we will see that for Adleman's algorithm the *number of algorithm steps* is linear in problem size.

Adleman (the "A," by the way, of "RSA.") gave a laboratory demonstration of the procedure in 1994 for $n = 7$, which is a very small instance of HPP. (We will use this instance, shown in Fig. IV.7, as an example.) Later his group applied similar techniques to solving a 20-variable 3-SAT problem, which has more than a million potential solutions (see p. 218).[4]

B.1.b Problem Representation

The heart of Adleman's algorithm is a clever way to encode candidate paths in DNA. Vertices are represented by single-stranded 20mers, that is, sequences of 20nt (nucleotides). They were generated at random and assigned to the vertices, but with the restriction that none of them were too similar or complementary. Each vertex code can be considered a catenation of two 10mers: $v_i = a_i b_i$ (i.e., a_i is the $5'$ 10mer and b_i is the $3'$ 10mer). Edges are also

[4] `https://en.wikipedia.org/wiki/Adleman,_Leonard` (accessed 2012-11-04).

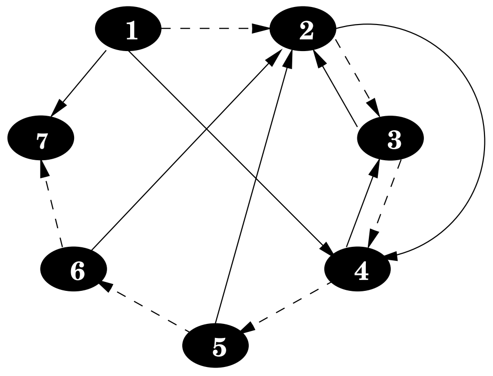


Figure IV.7: HPP solved by Adleman. The HP is indicated by the dotted edges. [source: Amos, Fig. 5.2]

represented by 20mers. The edge from vertex i to vertex j is represented by

$$e_{i\to j} = b_i a_j,\text{ where } v_i = a_i b_i,\text{ and } v_j = a_j b_j.$$

Paths are represented by using complements of the vertex 20mers to stitch together the edge 20mers. (Of course, using the complements of the edges to stitch together the vertices works as well.) For example, a path $2 \to 3 \to 4$ is represented:

$$\begin{array}{l} \overbrace{b_2 \quad a_3}^{e_{2\to3}} \quad \overbrace{b_3 \quad a_4}^{e_{3\to4}} \\ \hline \qquad \underbrace{\overline{a_3} \quad \overline{b_3}}_{\overline{v_3}} \end{array}$$

The edges from v_{in} and to v_{out} have special representations as 30mers:

$$\begin{aligned} e_{\text{in}\to j} &= v_{\text{in}} a_j,\text{ where } v_j = a_j b_j, \\ e_{i\to\text{out}} &= b_i v_{\text{out}},\text{ where } v_i = a_i b_i. \end{aligned}$$

Note that the special representation of the initial and terminal edges results in blunt ends for complete paths.

Therefore, for the $n = 7$ problems, candidate solutions were 140bp in length: There are $n-1$ edges, but the first and last edges are 30mers. Hence $2 \times 30 + (n-3) \times 20 = 140$. Ligation is used to remove the nicks in the backbone.

B.1.c Adleman's Algorithm

algorithm Adlemen:

Step 1 (generation of all paths): Generate multiple representations of all possible paths through the graph. This is done by combining the oligos for the edges with the oligos for the complements of the vertices in a single ligation reaction.

Step 2: Amplify the concentration of paths beginning with v_{in} and ending with v_{out}. This is done by PCR using v_{in} and $\overline{v_{\text{out}}}$ as primers. Remember that denaturation separates the sense and antisense strands. PCR extends the sense strand in the $3'$ direction from v_{in}, and extends the antisense strand in the $3'$ direction from $\overline{v_{\text{out}}}$. At the end of this step we have paths of all sorts from v_{in} to v_{out}.

Step 3: Only paths with the correct length are retained; for $n = 7$ this is 140bp. This operation is accomplished by gel electrophoresis. The band corresponding to 140bp is determined by comparison with a *marker lane*, and the DNA is extracted from this band and amplified by PCR. Gel electrophoresis and PCR are repeated to get a sufficient quantity of the DNA. We now have paths from v_{in} to v_{out}, but they might not be Hamiltonian.

Step 4 (affinity purification): Select for paths that contain all the vertices (and are thus necessarily Hamiltonian). This is done by first selecting all those paths that contain v_1, and then, of those, all that contain v_2, and so forth. To select for v_i, first heat the solution to separate the strands. Then add the $\overline{v_i}$ bound to a magnetic bead. Rehybridize (so the beads are bound to strands containing v_i), and use a magnet to extract all the paths containing v_i. Repeat this process for each vertex.

Step 5 If there any paths left, they are Hamiltonian. Therefore amplify them by PCR and inspect the result by gel electrophoresis to see if there are any

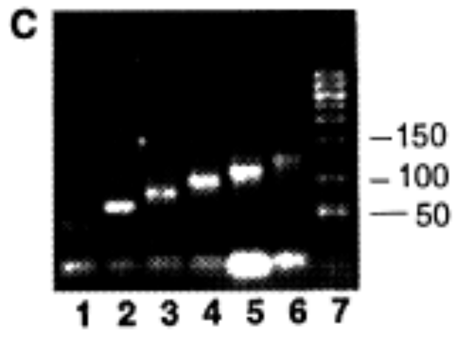


Figure IV.8: Electrophoresis showing solution to HPP problem.

strands of the correct length. If there are, then there is a Hamiltonian path; if there aren't, then there is not. If desired, the precise HP can be determined by a graduated PCR procedure: Run $n - 1$ parallel PCR reactions. In the ith lane, v_{in} is the left primer and v_i is the right primer. This will produce bands with lengths 40, 60, 80, 100, 120, and 140 bp. The lane that has a band at 40 corresponds to the first vertex after v_{in} in the path, the lane with a band at 60 corresponds to the next vertex, etc. This final readout process depends on there being only one Hamiltonian path, and it is error-prone due to its dependence on PCR.
□

B.1.d Discussion

Adleman's algorithm is linear in the number of nodes, since the only iteration is Step 4, which is repeated for each vertex. Step 5 is also linear if the path is read out. Thanks to the massive parallelism of molecular computation, it solves this NP-complete problem in linear time. Adleman's experiment took about a week, but with a more automated approach it could be done in a few hours. On the other hand, the PCR process cannot be significantly shortened.

In addition to time, we need to consider the *molecular resources* required. The number of different oligos required is proportional to n, but the number of strands is much larger, since there must be multiples instances of each possible path. If d is the average degree of the graph, then there are about d^n possible paths (exponential in n). For example, if $d = 10$ and $n = 80$, then the required 10^{80} DNA molecules is more than the estimated number of atoms in the universe. Hartmanis calculated that for $n = 200$ the weight of the DNA would exceed the weight of the earth. So this brute-force approach is still defeated by exponential explosion. Lipton (1995) estimates that Adleman's algorithm is feasible for $n \leq 70$, based on an upper limit of $10^{21} \approx 2^{70}$ DNA strands (Boneh, Dunworth, Lipton & Sgall, 1996), but this is also feasible on conventional computers.

Nevertheless, Adleman's algorithm illustrates the massive parallelism of molecular computation. Step 1 (generation of all possible paths) took about an hour for $n = 7$. Adleman estimates that about 10^{14} ligation operations were performed, and that it could be scaled up to 10^{20} operations. Therefore, speeds of about 10^{15} to 10^{16} ops/sec (1–10 peta-operations/s) should be achievable, which is, digital supercomputer range. Adlemen also estimates that 2×10^{19} ligation operations were performed per joule of energy. Contemporary supercomputers perform only 10^9 operations per joule, so molecular computation is 10^{10} more energy-efficient. It is near the thermodynamic limit of 34×10^{19} operations per joule. Recall (Ch. II, Sec. **??**) $kT \ln 2 \approx 3 \times 10^{-9}$pJ $= 3 \times 10^{-21}$J, so there can be about 3.3×10^{20} bit changes/J.[5]

A more pervasive problem is the inherent error in the filtering processes (due to incorrect hybridization). Some strands we don't want, get through; and some that we do want, don't. With many filtering stages the errors accumulate to the extent that the algorithms fail. There are some approaches to error-resistant DNA computing, but this is an open problem.

[5]DNA is of course space efficient. One bit of information occupies about 1 cubic nm, whereas contemporary disks store a bit in about 10^{10} cubic nm. That is, DNA is a 10^{10} improvement in density.

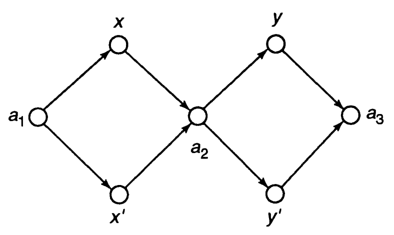


Figure IV.9: Graph G_2 for Lipton's algorithm (with two variables, x and y). [source: Lipton (1995)]

B.2 Lipton: SAT

In this section we will discuss DNA solution of another classic NP-complete problem, *Boolean satisfiability*, in fact the first problem proved to be NP-complete.[6]

B.2.a Review of SAT problem

In the Boolean satisfiability problem (called "SAT"), we are given a Boolean expression of n variables. The problem is to determine if the expression is *satisfiable*, that is, if there is an assignment of Boolean values to the variables that makes the expression true.

Without loss of generality, we can restrict our attention to expressions in *conjunctive normal form*, for every Boolean expression can be put into this form. That is, the expression is a conjunction of *clauses*, each of which is a disjunction of either positive or negated variables, such as this:

$$(x_1 \vee x_2' \vee x_3') \wedge (x_3 \vee x_5' \vee x_6) \wedge (x_3 \vee x_6' \vee x_4) \wedge (x_4 \vee x_5 \vee x_6),$$

For convenience we use primes for negation, for example, $x_2' = \neg x_2$. In the above example, we have $n = 6$ variables $m = 4$ clauses. The (possibly negated) variables are called *literals*.

B.2.b Data representation

To apply DNA computation, we have to find a way to represent potential solutions to the problem as DNA strands. Potential solutions to SAT are

[6]This section is based on Richard J. Lipton (1995), "DNA solution of hard computational problems," *Science* **268**: 542–5.

n-bit binary strings, which can be thought of as paths through a particular graph G_n (see Fig. IV.9). For vertices $a_k, x_k, x'_k, k = 1, \ldots, n$, and a_{n+1}, there are edges from a_k to x_k and x'_k, and from x_k and x'_k to a_{k+1}. Binary strings are represented by paths from a_1 to a_{n+1}. A path that goes through x_k encodes the assignment $x_k = 1$ and a path through x'_k encodes $x_k = 0$. The DNA encoding of these paths is essentially the same as in Adleman's algorithm.

B.2.c Lipton's Algorithm

algorithm Lipton:

Input: Suppose we have an instance (formula) to be solved: $I = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. The algorithm will use a series of "test tubes" (reaction vessels) $T_0, T_1, \ldots, T_m$ and $T^i_1, \overline{T}^i_1, \ldots, T^i_m, \overline{T}^i_m$, for $i = 0, \ldots, n$.

Step 1 (initialization): Create in a test tube T_0 a *library* of all possible n-bit binary strings, encoded as above as paths through the graph.

Step 2 (clause satisfaction): For each clause C_k, $k = 1, \ldots, m$: we will extract from T_{k-1} only those strings that satisfy C_k, and put them in T_k. (These successive filtrations in effect do an AND operation.) The goal is that the DNA in T_k satisfies the first k clauses of the formula. That is, $\forall x \in T_k \ \forall\, 1 \leq j \leq k : C_j(x) = 1$. Here are the details.

For $k = 0, \ldots, m-1$ do the following steps:

Precondition: The strings in T_k satisfy clauses $C_1, \ldots, C_k$.

Let $\ell = |C_{k+1}|$ (the number of literals in C_{k+1}), and suppose C_{k+1} has the form $v_1 \vee \cdots \vee v_\ell$, where the v_i are literals (positive or negative variables). Our goal is to find all strings that satisfy at least one of these literals. To accomplish this we will use an *extraction operation* $E(T, i, a)$ that extracts from test tube T all (or most) of the strings whose ith bit is a.

Let $\overline{T}_k^0 = T_k$. Do the following for literals $i = 1, \ldots, \ell$.

Positive literal: Suppose $v_i = x_j$ (some positive literal). Let $T_k^i = E(\overline{T}_k^{i-1}, j, 1)$ and let $a = 1$ (used below). These are the paths that satisfy this positive literal, since they have 1 in position j.

Negative literal: Suppose $v_i = x'_j$ (some negative literal). Let $T_k^i = E(\overline{T}_k^{i-1}, j, 0)$ and let $a = 0$. These are the paths that satisfy this negative literal, since they have 0 in position j.

In either case, T_k^i are the strings that satisfy literal i of the clause. Let $\overline{T}_k^i = E(\overline{T}_k^{i-1}, j, \neg a)$ be the remaining strings (which do not satisfy this literal). Continue the process above until all the literals in the clause are processed. At the end, for each $i = 1, \ldots, \ell$, T_k^i will contain the strings that satisfy literal i of clause k.

Combine $T_k^1, \ldots, T_k^\ell$ into T_{k+1}. (Combining the test tubes effectively does OR.) These will be the strings that satisfy at least one of the literals in clause $k + 1$.

Postcondition: The strings in T_{k+1} satisfy clauses $C_1, \ldots, C_{k+1}$.

Continue the above for $k = 1, \ldots, m$.

Step 3 (detection): At this point, the strings in T_m (if any) are those that satisfy $C_1, \ldots, C_m$, so do a *detect* operation (for example, with PCR and gel electrophoresis) to see if there are any strings left. If there are, the formula is satisfiable; if there aren't, then it is not.
□

If the number of literals per clause is fixed (as in the 3-SAT problem), then performance is linear in m. The main problem with this algorithm is the effect of errors, but imperfections in extraction are not fatal, so long as there are enough copies of the desired sequence. In 2002, Adelman's group solved a 20-variable 3-SAT problem with 24 clauses, finding the unique satisfying string.[7] In this case the number of possible solutions is $2^{20} \approx 10^6$. Since the degree of the specialized graph used for this problem is small, the number of possible paths is not excessive (as it might be in the Hamiltonian Path Problem). They stated, "This computational problem may be the largest yet solved by nonelectronic means," and they conjectured that their method might be extended to 30 variables.

[7] Ravinderjit S. Braich, Nickolas Chelyapov, Cliff Johnson, Paul W. K. Rothemund, Leonard Adleman, "Solution of a 20-Variable 3-SAT Problem on a DNA Computer," *Science* 296 (19 Apr. 2002), 499–502.

B.3 Test tube programming language

Filtering algorithms use a small set of basic DNA operations, which can be extended to a *Test Tube Programming Language (TTPL)*, such as was developed in the mid 90s by Lipton and Adleman (Adleman, 1995).

B.3.a Basic Operations

DNA algorithms operate on "test tubes," which are multi-sets of strings over $\Sigma = \{\mathrm{A, C, T, G}\}$. There are four basic operations (all implementable):

Extract (or separate): There are two complementary *extraction* (or *separation*) operations. Given a test tube t and a string w, $+(t, w)$ returns all strings in t that have w as a subsequence:

$$+(t, w) \stackrel{\text{def}}{=} \{s \in t \mid \exists u, v \in \Sigma^* : s = uwv\}.$$

Likewise, $-(t, w)$ returns a test tube of all the remaining strings:

$$-(t, w) \stackrel{\text{def}}{=} t \; - \; +(t, w) \quad \text{(multi-set difference).}$$

Merge: The *merge* operation combines several test tubes into one test tube:

$$\cup(t_1, t_2, \ldots, t_n) \stackrel{\text{def}}{=} t_1 \cup t_2 \cup \cdots \cup t_n.$$

Detect: The detect operation determines if any DNA strings remain in a test tube:

$$\text{detect}(t) \stackrel{\text{def}}{=} \begin{cases} \textbf{true}, & \text{if } t \neq \emptyset \\ \textbf{false}, & \text{otherwise} \end{cases}.$$

Amplify: Given a test tube t, the *amplify* operation produces two copies of it: $t', t'' \leftarrow \text{amplify}(t)$. Amplification is a problematic operation, which depends on the special properties of DNA and RNA, and it may be error prone. Therefore it is useful to consider a *restricted model* of DNA computing that avoids or minimizes the use of amplification.

The following additional operations have been proposed:

Length-separate: This operation produces a test tube containing all the strands less than a specified length:

$$(t, \leq n) \stackrel{\text{def}}{=} \{s \in t \mid |s| \leq n\}.$$

Position-separate: There are two *position-separation* operations, one that selects for strings that *begin* with a given sequence, and one for sequences that end with it:

$$\begin{aligned} B(t,w) &\overset{\text{def}}{=} \{s \in t \mid \exists v \in \Sigma^* : s = wv\}, \\ E(t,w) &\overset{\text{def}}{=} \{s \in t \mid \exists u \in \Sigma^* : s = uw\}. \end{aligned}$$

B.3.b Examples

AllC: The following example algorithm detects if there are any sequences that contain only C:

```
**procedure** [out] = AllC(t, A, T, G)
  t ← −(t, A)
  t ← −(t, T)
  t ← −(t, G)
  out ← detect (t)
**end procedure**
```

HPP: Adelman's solution of the HPP can be expressed in TTPL:

```
**procedure** [out] = HPP(t, vin, vout)
  t ← B(t, vin)                    //begin with vin
  t ← E(t, vout)                   //end with vout
  t ← (t, ≤ 140)                   //correct length
  **for** i=1 **to** 5 **do**                 //except vin and vout
    t ← +(t, s[i])                 //contains vertex i
  **end for**
  out ← detect(t)                  //any HP left?
**end procedure**
```

SAT: Programming Lipton's solution to Sat requires another primitive operation, which extracts all sequences for which the jth bit is $a \in \mathbf{2}$: $E(t,j,a)$. Recall that these are represented by the sequences containing x_j and x'_j. Therefore:

$$\begin{aligned} E(t,j,1) &= +(t,x_j), \\ E(t,j,0) &= +(t,x'_j). \end{aligned}$$

```
procedure [out] = SAT(t)
  for k = 1 to m do                          // for each clause
    for i = 1 to n do                        // for each literal
      if C[k][i] = $x_j$                     // i-th literal in clause k
        then t[i] ← E(t,j,1)
        else t[i] ← E(t,j,0)
      end if
    end for
    t ← merge(t[1], t[2], . . . , t[n]) // solutions for clauses 1,...,k
  end for
  out ← detect(t)
end procedure
```

B.4 Parallel filtering model

The *parallel filtering model* (PFM) was developed in the mid 90s by Martyn Amos and colleagues to be a means of describing DNA algorithms for any NP problem (as opposed to Ableson's and Lipton's algorthms, which are specialized to particular problems). "Our choice is determined by what we know can be effectively implemented by very precise and complete chemical reactions within the DNA implementation."[8] All PFM algorithms begin with a multi-set of all candidate solutions. The PFM differs from other DNA computation models in that removed strings are discarded and cannot be used in further operations. Therefore this is a "mark and destroy" approach to DNA computation.

B.4.a Basic operations

The basic operations are *remove*, *union*, *copy*, and *select*.

Remove: The operation remove$(U, \{S_1, \ldots, S_n\})$ removes from U any strings that contain any of the substrings S_i. Remove is implemented by two primitive operations, *mark* and *destroy*:

Mark: mark(U, S) marks all strands that have S as a substring. This is done by adding $\overline{S}$ as a primer with polymerase to make it double-stranded.

Destroy: destroy(U) removes all the marked sequences from U. This is done by adding a restriction enzyme that cuts up the double-stranded part. These fragments can be removed by gel electrophoresis, or left in the solution (since they won't affect it). Restriction enzymes are much more reliable than other DNA operations, which is one advantage of the PFM approach.

Union: The operation union$(\{U_1, \ldots, U_n\}, U)$ combines *in parallel* the multi-sets $U_1, \ldots, U_n$ into U.

Copy: The operation copy$(U, \{U_1, \ldots, U_n\})$ divides multi-set U into n equal multi-sets $U_1, \ldots, U_n$.

Select: The operation select(U) returns a random element of U. If $U = \emptyset$, then it returns $\emptyset$.

Homogeneous DNA can be detected and sequenced by PCR, and nested PCR can be used in non-homogeneous cases (multiple solutions). All of these operations are assumed to be constant-time. Periodic amplification (especially after copy operations) may be necessary to ensure an adequate

[8]Amos, p. 50.

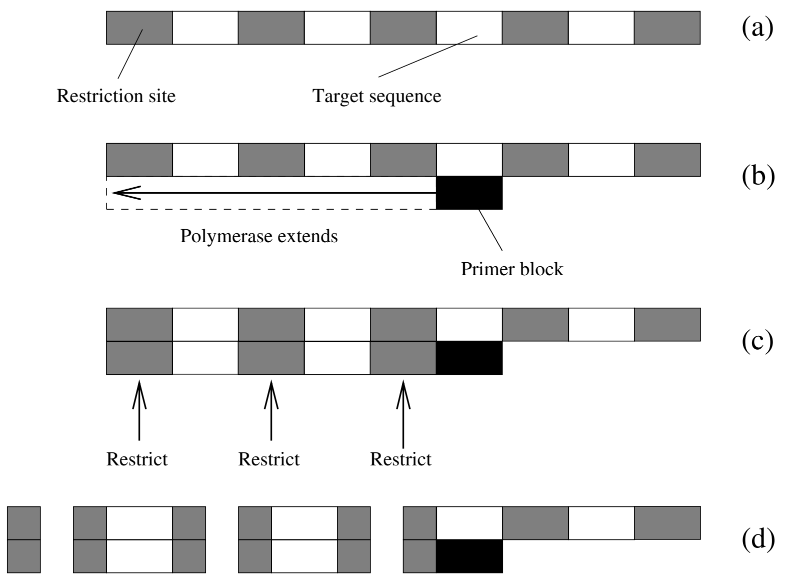


Figure IV.10: *Remove* operation implemented by *mark* and *destroy.* [source: Amos]

number of instances. Amos et al. have done a number of experiments to determine optimum reactions parameters and procedures.

B.4.b Permutations

Amos et al. describe a PFM algorithm for generating all possible permutations of a set of integers.

algorithm Permutations:

Input: "The input set U consists of all strings of the form $p_1 i_1 p_2 i_2 \cdots p_n i_n$ where, for all j, p_j uniquely encodes 'position j' and each i_j is in $\{1, 2, \ldots, n\}$. Thus each string consists of n integers with (possibly) many occurrences of the same integer."[9]

[9]Amos, p. 51.

Iteration:

```
for $j = 1$ to $n - 1$ do
  copy($U, \{U_1, U_2, \ldots, U_n\}$)
  for $i = 1, 2, \ldots, n$ and all $k > j$
    in parallel do remove($U_i, \{p_j i_j \neq p_j i, p_k i\}$)
  // $U_i$ contains $i$ in $j$th position and no other $i$s
  union($\{U_1, U_2, \ldots, U_n\}, U$)
end for
$P_n \leftarrow U$
```

In the preceding, remove$(U_i, \{p_j i_j \neq p_j i, p_k i\})$ means to remove from U_i all strings that have a p_j value not equal to i and all strings containing $p_k i$ for any $k > j$. For example, if $i = 2$ and $j = n - 1$, this remove operation translates to remove$(U_2, \{p_{n-1}1, p_{n-1}3, p_{n-1}4, \ldots, p_{n-1}n, p_n 2\})$. That is, it eliminates all strings except those with 2 in the $n - 1$ position, and eliminates those with 2 in the n position. At the end of iteration j we have:

$$\overbrace{p_1 i_1 p_2 i_2 \cdots p_j i_j}^{\alpha} \underbrace{p_{j+1} i_{j+1} \cdots \ p_n i_n}_{\beta}$$

where α represents a permutation of j integers from $1, \ldots, n$, and none of these integers $i_1, \ldots, i_j$ are in β.

Amos shows how to do a number of NP-complete problems, including 3-vertex-colorability, HPP, subgraph isomorphism, and maximum clique.

C Formal models

C.1 Sticker systems

C.1.a Basic operations

The *sticker model* was developed by Rosweis et al. in the mid-1990s. It depends primarily on separation by means of hybridization and makes no use of strand extension and enzymes. It implements a sort of random-access binary memory. Each bit position is represented by a substrand of length m. A *memory strand* comprises k contiguous substrands, and so has length $n = km$ and can store k bits. *Sticker strands* or *stickers* are strands that are complementary to substrands representing bits. When a sticker is bound to a bit, it represents 1, and if no sticker is bound, the bit is 0. Such a strand, which is partly double and partly single, is called a *complex* strand.

Computations begin with a prepared *library* of strings. A (k, l) library uses the first $l \leq k$ bits as inputs to the algorithm, and the remaining $k-l$ for output and working storage. Therefore, the last $k - l$ are initially 0. There are four basic operations, which act on multi-sets of binary strings:

Merge: Creates the union of two *tubes* (multi-sets).

Separate: The operation separate(N, i) separates a tube N into two tubes: $+(N, i)$ contains all strings in which bit i is 1, and $-(N, i)$ contains all strings in which bit i is 0.

Set: The operation set(N, i) produces a tube in which every string from N has had its ith bit set to 1.

Clear: The operation clear(N, i) produces a tube in which every string from N has had its ith bit cleared to 0.

C.1.b Set cover problem

The *set cover problem* is a classic NP-complete problem. Given a finite set of p objects S, and a finite collection of subsets of S $(C_1, \ldots, C_q \subset S)$ whose union is S, find the *smallest* collection of these subsets whose union is S. For an example, consider $S = \{1, 2, 3, 4, 5\}$ and $C_1 = \{3, 4, 5\}, C_2 = \{1, 3, 4\}, C_3 = \{1, 2, 5\}, C_4 = \{3, 4\}$. In this case there are three minimal solutions: $\{C_1, C_3\}, \{C_3, C_4\}, \{C_2, C_3\}$.

algorithm Minimum Set Cover:

Data representation: The memory strands are of size $k = p + q$. Each strand represents a collection of subsets, and the first q bits encode which subsets are in the collection; call them *subset bits*. For example 1011 represents $\{C_1, C_3, C_4\}$ and 0010 represents $\{C_3\}$. Eventually, the last p bits will represent the union of the collection, that is, the elements of S that are contained in at lease one subset in the collection; call them *element bits*. For example, 0101 10110 represents $\{C_2, C_4\}$ $\{1, 3, 4\}$.

Library: The algorithm begins with the $(p + q, q)$ library, which must be initialized to reflect the subsets' members.

Step 1 (initialization): For all strands, if the i subset bit is set, then set the bits for all the elements of that subset. Call the result tube N_0. This is accomplished by the following code:

```
Initialize $(p + q, q)$ library in $N_0$
for $i = 1$ to $q$ do
  separate$(+(N_0, i), -(N_0, i))$        //separate those with subset $i$
  for $j = 1$ to $|C_i|$ do
    set$(+(N_0, i), q + c_i^j)$             //set bit for $j$th element of set $i$
  end for
  $N_0 \leftarrow$ merge$(+(N_0, i), -(N_0, i))$ //recombine
end for
```

Step 2 (retain covers): Retain only the strands that represent collections that cover the set. To do this, retain in N_0 only the strands whose last p bits are set.

```
for $i = q + 1$ to $q + p$ do
  $N_0 \leftarrow +(N_0, i)$                    //retain those with element $i$
end for
```

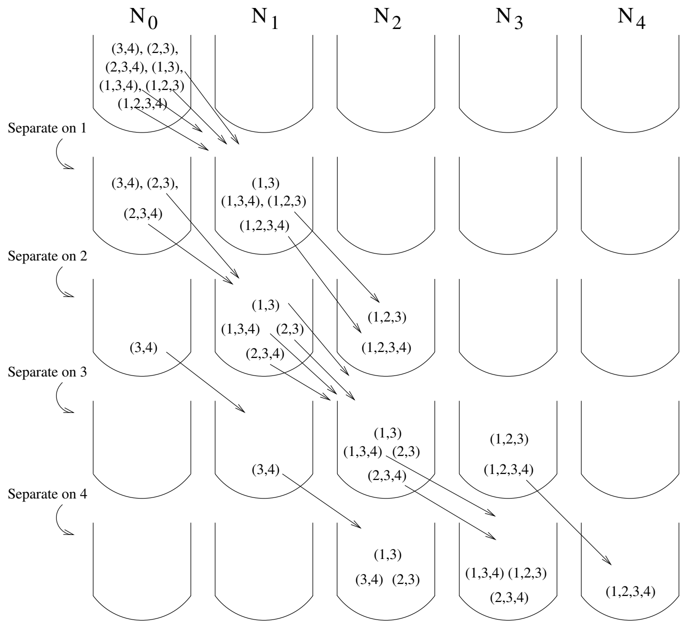


Figure IV.11: Sorting of covers by repeated separations. [source: Amos, Fig. 3.4]

Step 3 (isolate minimum covers): Tube N_0 now holds all covers, so we have to somehow sort its contents to find the minimum cover(s). Set up a row of tubes $N_0, N_1, \ldots, N_q$. We will arrange things so that N_i contains the covers of size i; then we just have to find the first tube with some DNA in it.

Sorting: For $i = 1, \ldots, q$, "drag" to the right all collections containing C_i, that is, for which bit i is set (see Fig. IV.11). This is accomplished by the following code:[10]

```
for $i = 0$ to $q - 1$ do
  for $j = i$ down to $0$ do
    separate($+(N_j, i + 1), -(N_j, i + 1)$) //those that do & don't have $i$
```

[10]Corrected from Amos p. 60.

```
    $N_{j+1} \leftarrow \text{merge}(+(N_j, i+1), N_{j+1})$  //move those that do to $N_{j+1}$
    $N_j \leftarrow -(N_j, i+1)$                              //leave those that don't in $N_j$
  **end for**
**end for**
```

Detection: Find the minimum i such that N_i contains DNA; N_i contains the minimum covers.
□

The algorithm is $\mathcal{O}(pq)$.

C.2 Splicing systems

It has been argued that the full power of a TM requires some sort of string editing operation. Therefore, beginning with Tom Head (1987), a number of *splcing systems* have been defined. The splicing operations takes two strings $S = S_1S_2$ and $T = T_1T_2$ and performs a "crossover" at a specified location, yielding S_1T_2 and T_1S_2. *Finite extended splicing systems* have been shown to be computationally universal (1996).

The *Parallel Associative Memory (PAM) Model* was defined by Reif in 1995. It is based on a restricted splicing operation called *parallel associative matching* (PA-Match) operation, which is named *Rsplice*. Suppose $S = S_1S_2$ and $T = T_1T_2$. Then,

$$\text{Rsplice}(S, T) = S_1T_2, \quad \text{if } S_2 = T_1,$$

and is undefined otherwise. The PAM model can simulate nondeterministic TMs and parallel random access machines.

Figure IV.12: The *fok*I restriction enzyme bound to DNA. [source: wikipedia]

D Enzymatic computation

The molecular computation processes that we have seen so far are externally controlled by a person or conventional automatic controller sequencing the chemical operations.[11] In *autonomous molecular computation* the chemical processes sequence themselves so that they do not require external control. This is also called "one-pot" molecular computation; that is, you put all the reactants in one pot, and the molecular processes do the rest. Autonomous molecular computation is essential for, example, controlling drug delivery in the body.

Shapiro and his colleagues have demonstrated how to implement finite state machines (FSMs) by autonomous molecular computation. In addition to DNA, it uses a restriction enzyme, ligase, and ATP (for fuel).

The implementation is based on the *fok*I restriction enzyme. "Once the

[11]This section is based primarily on: (1) Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, and Ehud Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.
(2) Yaakov Benenson, Rivka Adam, Tamar Paz-Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proceedings of the National Academies of Science*, 100(5):2191–2196, 2003.

protein is bound to duplex DNA via its DNA-binding domain at the 5′-GGATG-3′ : 5′-CATCC-3′ recognition site, the DNA cleavage domain is activated and cleaves, without further sequence specificity, the first strand 9 nucleotides downstream and the second strand 13 nucleotides upstream of the nearest nucleotide of the recognition site."[12] It leaves 4-nucleotide sticky ends. That is, the restriction enzyme cuts the DNA as follows:

```
GGATGNNNNNNNNN NNNNNNNNNN
CCTACNNNNNNNNNNNNN NNNNNN
```

The 'N's can be any nucleotides (respecting Watson-Crick complementarity, of course).

Both the current state of the FSM and the input string are represented by a double DNA strand. *fok*I operates at the beginning of this string and leaves a sticky end that encodes both the current state and the next input symbol (see p. 231 below). The state transitions of the FSM are encoded in *transition molecules*, which have sticky ends complementary to the state-symbol code at the beginning of the string. The rest of a transition molecule ensures that the string properly encodes the new state, including adding a new recognition site for the enzyme. A matching transition molecule binds to the string's sticky end, providing a new opportunity for *fok*I to operate, and so the process continues.

A state transition $(q, s_1) \rightarrow q'$ can be represented:

$$[q, s_1]s_2s_3 \cdots s_n t \Longrightarrow [q', s_2]s_3 \cdots s_n t$$

where $[q, s]$ represents a DNA sequence encoding both state q and symbol s, and t is a *terminator* for the string. The *fok*I enzyme cleaves off $[q, s_1]$ in such a way that a transition molecule can bind to the sticky end in a way that encodes $[q', s_2]$. A special *terminator* symbol marks the end of the input string.

As an example we will consider a two-state FSM on {a, b} that accepts strings with an even number of 'b's. Ignoring the terminator, DNA codes are assigned to the two symbols 'a' and 'b' as follows:

$$\begin{array}{rcl} \text{a} & \mapsto & AA\alpha\alpha aa \\ \text{b} & \mapsto & BB\beta\beta bb \end{array}$$

[12]wikipedia, s.v. fokI.

where $A, \alpha, a, B, \beta, b$ are unspecified (by me) bases.[13] The bases are selected in such a way that either the first four bases ($AA\alpha\alpha$, $BB\beta\beta$) or the last four bases ($\alpha\alpha aa$, $\beta\beta bb$) encode the symbol. These alternatives represent the two machine states.

The transition molecules are constructed so that the distance between the recognition site (for *fok*I) and the next symbol depends on new state. As a consequence, when *fok*I operates it cleaves the next symbol code at a place that depends on the state. Therefore the sticky end encodes the state in the way that it represents the next symbol:

$$\begin{array}{rcl} [q_0, \mathrm{a}] & \mapsto & \alpha\alpha aa \\ [q_1, \mathrm{a}] & \mapsto & AA\alpha\alpha \\ [q_0, \mathrm{b}] & \mapsto & \beta\beta bb \\ [q_1, \mathrm{b}] & \mapsto & BB\beta\beta \end{array}$$

The transition molecules are:

$$\begin{array}{ll} (q_0, \mathrm{a}) \to q_0 & \texttt{GGATG}NNN \\ & \texttt{CCTAC}\overline{NNN\alpha\alpha aa} \\ (q_1, \mathrm{a}) \to q_1 & \texttt{GGATG}NNN \\ & \texttt{CCTAC}\overline{NNNAA\alpha\alpha} \\ (q_0, \mathrm{b}) \to q_1 & \texttt{GGATG}NNNNN \\ & \texttt{CCTAC}\overline{NNNNN\beta\beta bb} \\ (q_1, \mathrm{b}) \to q_0 & \texttt{GGATG}N \\ & \texttt{CCTAC}\overline{NBB\beta\beta} \end{array}$$

The Ns represent any bases as before. They are used as *spacers* to adjust the restriction site to represent the new state.

After transition to the new state the sense strand will look like this (for convenience assuming the next symbol is ‘a’):

$$\begin{array}{lr} q_0 & \mathrm{GGATG}NXXYYyyAA.\alpha\alpha aa \\ q_1 & \mathrm{GGATG}NNNXXYYyy.AA\alpha\alpha aa \end{array}$$

This is *after* attachment of the transition molecule but before restriction. Here XX represents either spacers or the first two bases of the previous first symbol, and $YYyy$ represents the last four bases of this symbol. The cleavage site is indicated by a period.

[13]Note that repeated letters might refer to different bases.

The longest strings processed in the PNAS experiments were 12.[14] Operation required about 20 seconds per step. However, the parallel speed was about 6.6×10^{10} ops/s/μl. Energy consumption was about $34kT$ per transition, which is only about $50\times$ the von Neumann-Landauer limit ($kT \ln 2$). The authors note, "Reaction rates were surprisingly insensitive to temperature and remained similar over the range of 2–20°C." This implementation also handles nondeterministic FSMs (just put in all the transition molecules), but the yield decreases exponentially (due to following out all the nondeterministic paths, breadth-first). Therefore it doesn't seem to be practical for nondeterministic machines.

[14]Benenson et al., PNAS **100** (5), March 4, 2003.

Chapter V

Analog Computation

[1]

A Definition

Although analog computation was eclipsed by digital computation in the second half of the twentieth century, it is returning as an important alternative computing technology. Indeed, as explained in this chapter, theoretical results imply that analog computation can escape from the limitations of digital computation. Furthermore, analog computation has emerged as an important theoretical framework for discussing computation in the brain and other natural systems.

Analog computation gets its name from an *analogy*, or systematic relationship, between the physical processes in the computer and those in the system it is intended to model or simulate (the *primary system*). For example, the electrical quantities voltage, current, and conductance might be used as analogs of the fluid pressure, flow rate, and pipe diameter of a hydrolic system. More specifically, in traditional analog computation, physical quantities in the computation obey the same mathematical laws as physical quantities in the primary system. Thus the computational quantities are proportional

[1]This chapter is based on an unedited draft for an article that appeared in the *Encyclopedia of Complexity and System Science* (Springer, 2008).

to the modeled quantities. This is in contrast to *digital computation*, in which quantities are represented by strings of symbols (e.g., binary digits) that have no direct physical relationship to the modeled quantities. According to the *Oxford English Dictionary* (2nd ed., s.vv. analogue, digital), these usages emerged in the 1940s.

However, in a fundamental sense all computing is based on an analogy, that is, on a systematic relationship between the states and processes in the computer and those in the primary system. In a digital computer, the relationship is more abstract and complex than simple proportionality, but even so simple an analog computer as a slide rule goes beyond strict proportion (i.e., distance on the rule is proportional to the logarithm of the number). In both analog and digital computation—indeed in all computation—the relevant abstract mathematical structure of the problem is realized in the physical states and processes of the computer, but the realization may be more or less direct (MacLennan, 1994a,c, 2004).

Therefore, despite the etymologies of the terms "analog" and "digital," in modern usage the principal distinction between digital and analog computation is that the former operates on discrete representations in discrete steps, while the later operated on continuous representations by means of continuous processes (e.g., MacLennan 2004, Siegelmann 1999, p. 147, Small 2001, p. 30, Weyrick 1969, p. 3). That is, the primary distinction resides in the *topologies* of the states and processes, and it would be more accurate to refer to *discrete* and *continuous computation* (Goldstine, 1972, p. 39). (Consider so-called analog and digital clocks. The principal difference resides in the continuity or discreteness of the representation of time; the motion of the two (or three) hands of an "analog" clock do not mimic the motion of the rotating earth or the position of the sun relative to it.)

B Introduction

B.1 History

B.1.a Pre-electronic analog computation

Just like digital calculation, analog computation was originally performed by hand. Thus we find several analog computational procedures in the "constructions" of Euclidean geometry (Euclid, fl. 300 BCE), which derive from techniques used in ancient surveying and architecture. For example, Problem

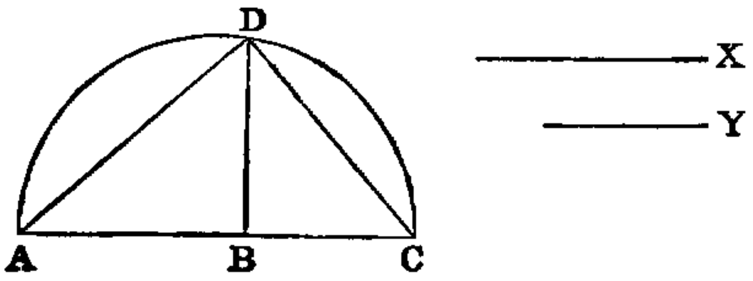
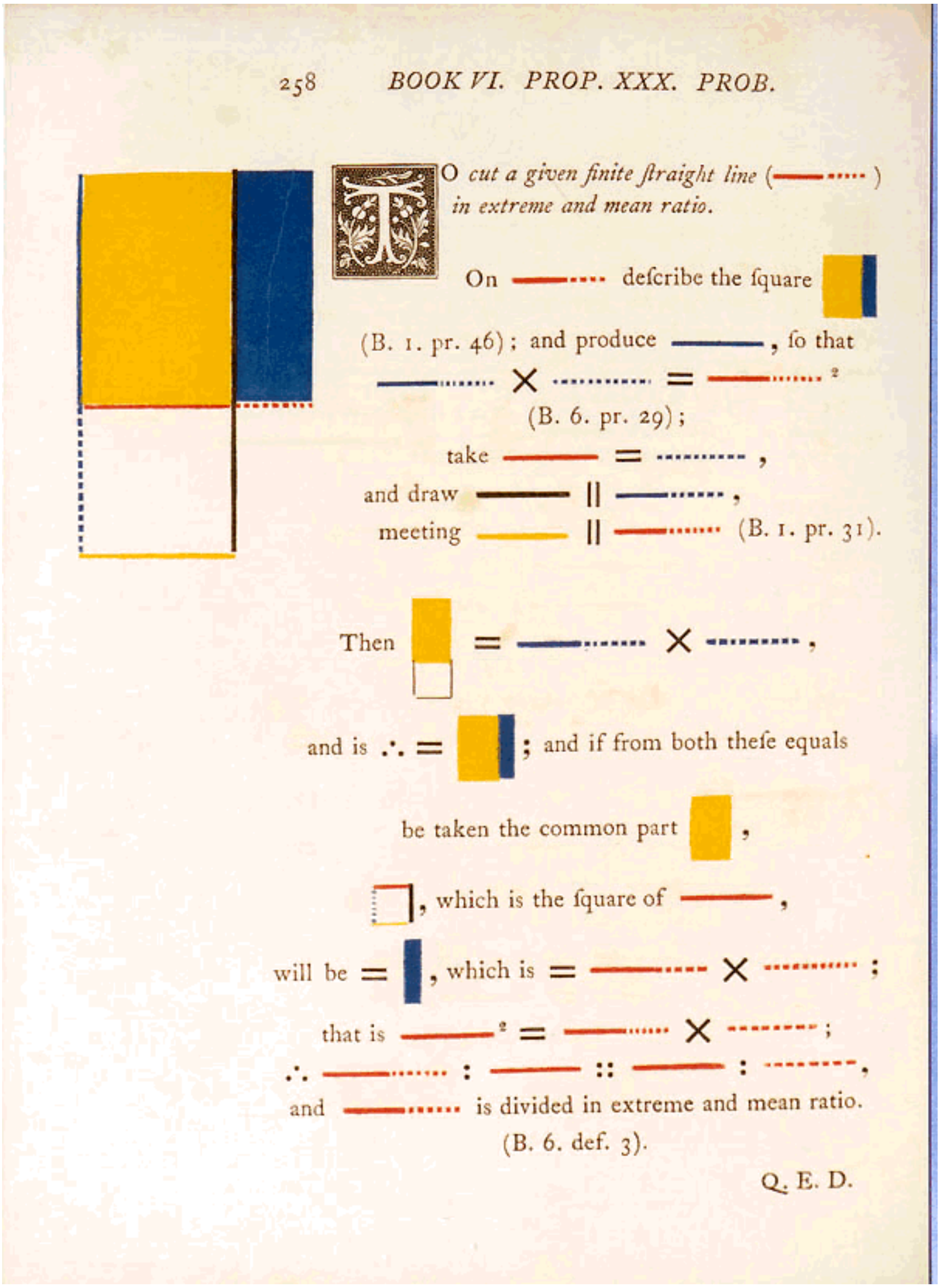


Figure V.1: Euclid Problem VI.13: To find a mean proportional between two given straight lines. Solution: Take on any line AC parts AB, BC respectively equal to X, Y. On AC describe a semicircle ADC. Erect BD at right angles to AC, meeting the semicircle in D. BD will be the mean proportional required.

II.51 is "to divide a given straight line into two parts, so that the rectangle contained by the whole and one of the parts shall be equal to the square of the other part." Also, Problem VI.13 is "to find a mean proportional between two given straight lines" (Fig. V.1), and VI.30 is "to cut a given straight line in extreme and mean ratio." These procedures do not make use of measurements in terms of any fixed unit or of digital calculation; the lengths and other continuous quantities are manipulated directly (via compass and straightedge). On the other hand, the techniques involve discrete, precise operational steps, and so they can be considered *algorithms*, but over continuous magnitudes rather than discrete numbers.

It is interesting to note that the ancient Greeks distinguished continuous *magnitudes* (Grk., *megethoi*), which have physical dimensions (e.g., length, area, rate), from discrete *numbers* (Grk., *arithmoi*), which do not (Maziarz & Greenwood, 1968). Euclid axiomatizes them separately (magnitudes in Book V, numbers in Book VII), and a mathematical system comprising both discrete and continuous quantities was not achieved until the nineteenth century in the work of Weierstrass and Dedekind.

The earliest known mechanical analog computer is the "Antikythera mechanism," which was found in 1900 in a shipwreck under the sea near the Greek island of Antikythera (between Kythera and Crete) (Figs. V.3, V.4). It dates to the second century BCE and appears to be intended for astronomical calculations. The device is sophisticated (at least 70 gears) and well engineered, suggesting that it was not the first of its type, and therefore that other analog



258 *BOOK VI. PROP. XXX. PROB.*

TO *cut a given finite ſtraight line* ( ) *in extreme and mean ratio.*

On deſcribe the ſquare (B. 1. pr. 46); and produce , ſo that × = ^{2} (B. 6. pr. 29);

take = ,

and draw || ,

meeting || (B. 1. pr. 31).

Then = × ,

and is ∴ = ; and if from both theſe equals

be taken the common part ,

, which is the ſquare of ,

will be = , which is = × ;

that is ^{2} = × ;

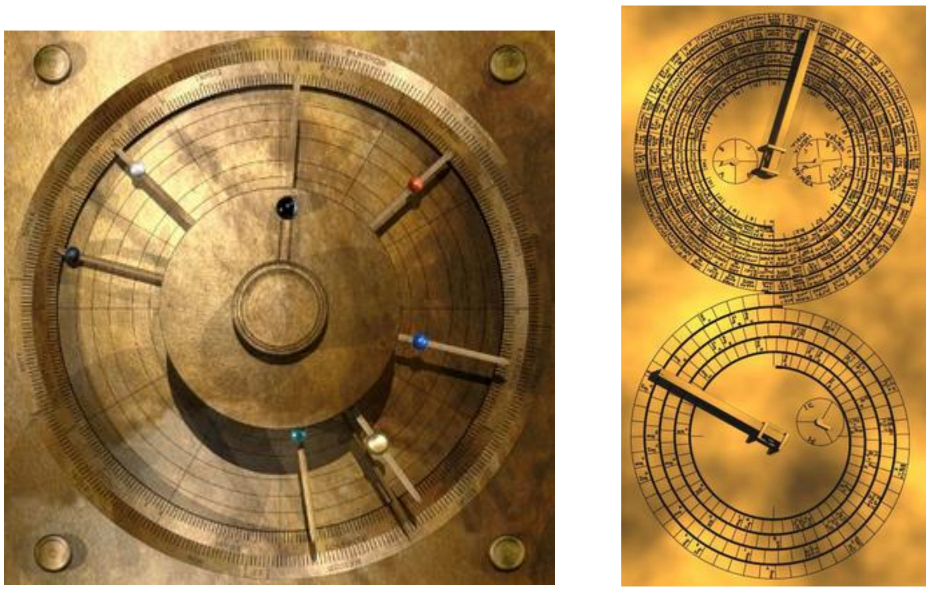
∴ : :: : ,

and is divided in extreme and mean ratio. (B. 6. def. 3).

Q. E. D.

Figure V.2: Euclid Problem VI.30 from Byrne's *First Six Books of Euclid's Elements* (1847). [source: http://www.math.ubc.ca/~cass/Euclid/byrne.html]

Figure V.3: The Antikythera Mechanism. An ancient analog computer (2nd century BCE) for astronomical calculations including eclipses. [source: wikipedia]



(a) front (b) back

Figure V.4: Computer reconstruction of Antikythera Mechanism. [source: wikipedia]

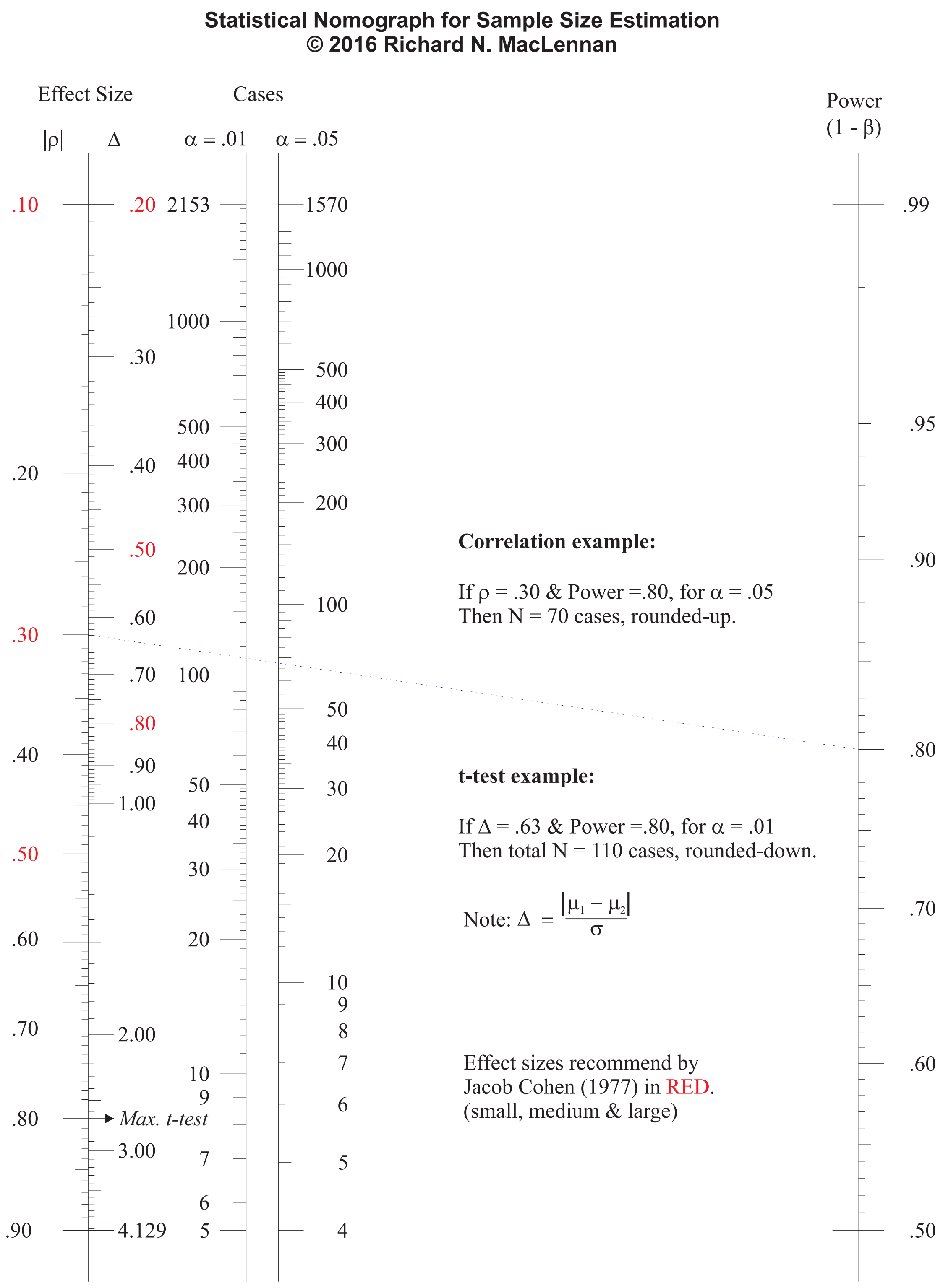


Figure V.5: Example nomograph.

computing devices may have been used in the ancient Mediterranean world (Freeth et al., 2006). Indeed, according to Cicero (*Rep.* 22) and other authors, Archimedes (c. 287–c. 212 BCE) and other ancient scientists also built analog computers, such as armillary spheres, for astronomical simulation and computation. Other antique mechanical analog computers include the astrolabe, which is used for the determination of longitude and a variety of other astronomical purposes, and the *torquetum*, which converts astronomical measurements between equatorial, ecliptic, and horizontal coordinates.

A class of special-purpose analog computer, which is simple in conception but may be used for a wide range of purposes, is the *nomograph* (also, *nomogram*, *alignment chart*) (Fig. V.5). In its most common form, it permits the solution of quite arbitrary equations in three real variables, $f(u, v, w) = 0$. The nomograph is a chart or graph with scales for each of the variables; typically these scales are curved and have non-uniform numerical markings. Given values for any two of the variables, a straightedge is laid across their positions on their scales, and the value of the third variable is read off where the straightedge crosses the third scale. Nomographs were used to solve many problems in engineering and applied mathematics. They improve intuitive understanding by allowing the relationships among the variables to be visualized, and facilitate exploring their variation by moving the straightedge. Lipka (1918) is an example of a course in graphical and mechanical methods of analog computation, including nomographs and slide rules.

Until the introduction of portable electronic calculators in the early 1970s, the *slide rule* was the most familiar analog computing device. Slide rules use logarithms for multiplication and division, and they were invented in the early seventeenth century shortly after John Napier's description of logarithms.

The mid-nineteenth century saw the development of the *field analogy method* by G. Kirchhoff (1824–87) and others (Kirchhoff, 1845). In this approach an electrical field in an electrolytic tank or conductive paper was used to solve two-dimensional boundary problems for temperature distributions and magnetic fields (Small, 2001, p. 34). It is an early example of *analog field computation*, which operates on continuous spatial distributions of quantity (i.e., fields).

In the nineteenth century a number of mechanical analog computers were developed for integration and differentiation (e.g., Lipka 1918, pp. 246–56, Clymer 1993). For example, the *planimeter* measures the area under a curve or within a closed boundary (Fig. V.6). While the operator moves a pointer along the curve, a rotating wheel accumulates the area. Similarly, the *inte-*

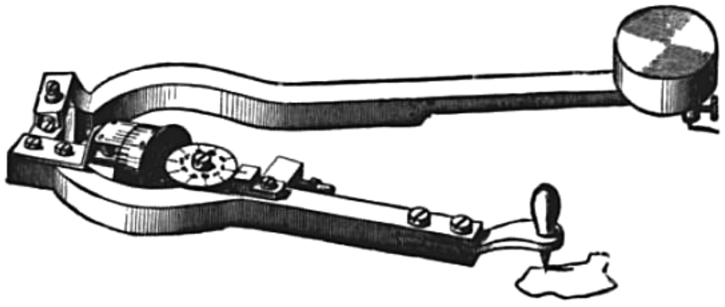


Figure V.6: Planimeter for measuring the area inside an arbitrary curve (1908). [source: wikipedia]

graph is able to draw the integral of a given function as its shape is traced (Fig. V.7). Other mechanical devices can draw the derivative of a curve or compute a tangent line at a given point.

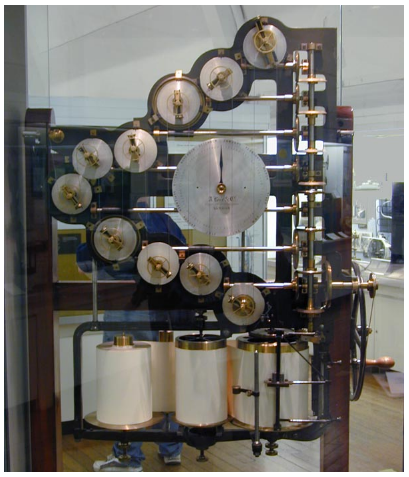


Figure V.8: Lord Kelvin's analog tide computer. [source: wikipedia]

In the late nineteenth century William Thomson, Lord Kelvin, constructed several analog computers, including a "tide predictor" and a "harmonic analyzer," which computed the Fourier coefficients of a tidal curve (Thomson, 1878, 1938) (Fig. V.8). In 1876 he described how the mechanical integrators invented by his brother could be connected together in a feedback loop in order to solve second and higher order differential equations (Small 2001, pp. 34–5, 42, Thomson 1876). He was unable to construct this *differential analyzer*, which had to await the invention of the torque amplifier in 1927.

The torque amplifier and other technical advancements permitted Vannevar Bush at MIT to construct the first practical differential analyzer in 1930 (Small, 2001, pp. 42–5) (Fig. V.9). It had six integrators and could also do addition, subtraction, multiplication, and division. Input

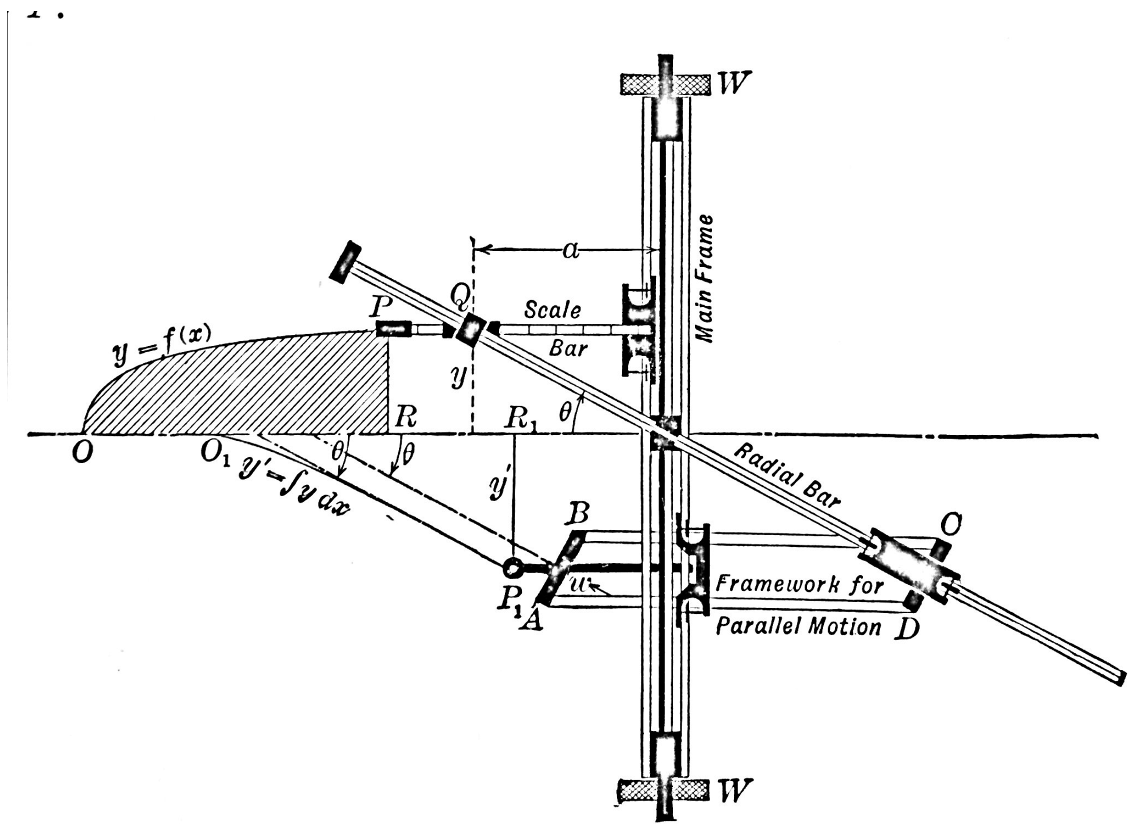


Figure V.7: Integraph for drawing the integral of an arbitrary curve (Lipka, 1918).

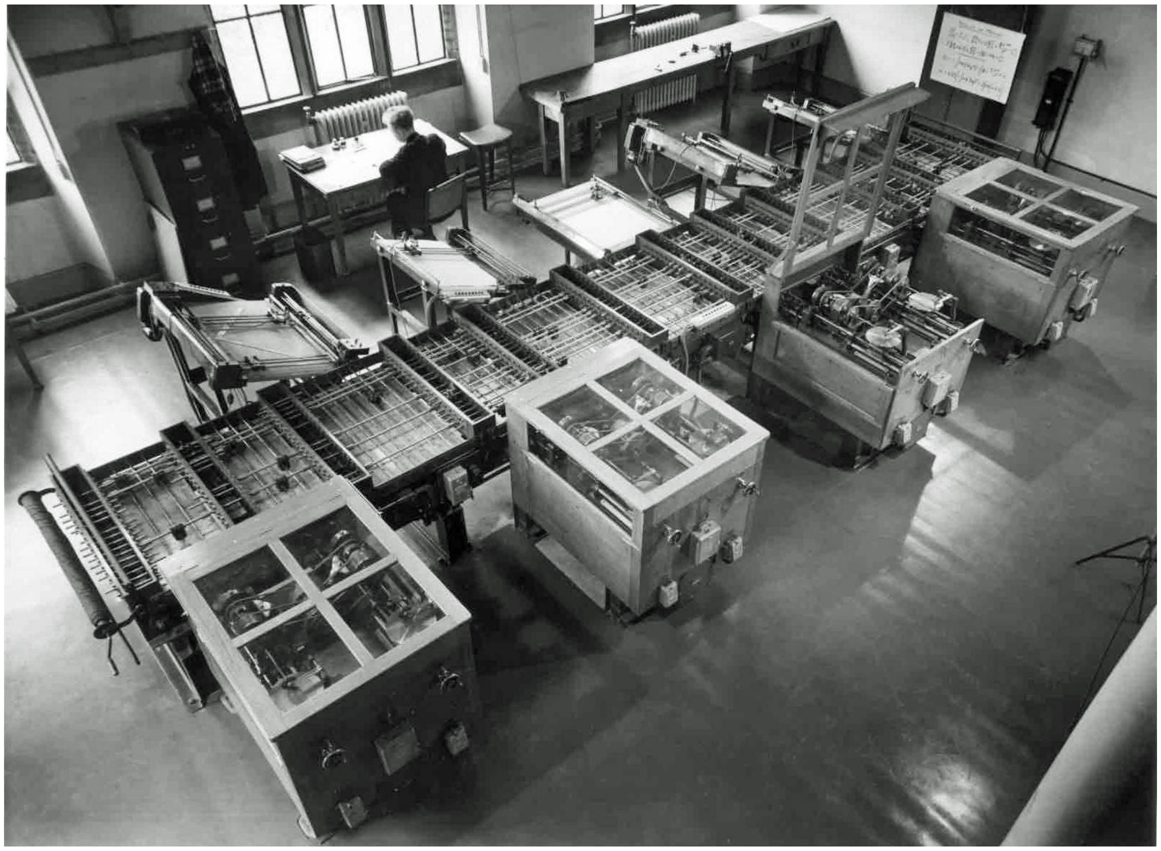


Figure V.9: Meccano differential analyzer at Cambridge University, 1938. The computer was constructed by Douglas Hartree based on Vannevar Bush's design. [source: wikipedia]

data were entered in the form of continuous curves, and the machine automatically plotted the output curves continuously as the equations were integrated. Similar differential analyzers were constructed at other laboratories in the US and the UK.

Setting up a problem on the MIT differential analyzer took a long time; gears and rods had to be arranged to define the required dependencies among the variables. Bush later designed a much more sophisticated machine, the Rockefeller Differential Analyzer, which became operational in 1947. With 18 integrators (out of a planned 30), it provided programmatic control of machine setup, and permitted several jobs to be run simultaneously. Mechanical differential analyzers were rapidly supplanted by electronic analog computers in the mid-1950s, and most were disassembled in the 1960s (Bowles 1996, Owens 1986, Small 2001, pp. 50–5).

During World War II, and even later wars, an important application of optical and mechanical analog computation was in "gun directors" and "bomb sights," which performed ballistic computations to accurately target artillery and dropped ordnance.

B.1.b Electronic analog computation in the 20th century

It is commonly supposed that electronic analog computers were superior to mechanical analog computers, and they were in many respects, including speed, cost, ease of construction, size, and portability (Small, 2001, pp. 54–6). On the other hand, mechanical integrators produced higher precision results (0.1%, vs. 1% for early electronic devices) and had greater mathematical flexibility (they were able to integrate with respect to any variable, not just time). However, many important applications did not require high precision and focused on dynamic systems for which time integration was sufficient; for these, electronic analog computers were superior.

Analog computers (non-electronic as well as electronic) can be divided into *active-element* and *passive-element* computers; the former involve some kind of amplification, the latter do not (Truitt & Rogers, 1960, pp. 2-1–4). Passive-element computers included the *network analyzers* that were developed in the 1920s to analyze electric power distribution networks, and which continued in use through the 1950s (Small, 2001, pp. 35–40). They were also applied to problems in thermodynamics, aircraft design, and mechanical engineering. In these systems networks or grids of resistive elements or reactive elements (i.e., involving capacitance and inductance as well as resistance) were used to model the spatial distribution of physical quantities such as voltage, current, and power (in electric distribution networks), electrical potential in space, stress in solid materials, temperature (in heat diffusion problems), pressure, fluid flow rate, and wave amplitude (Truitt & Rogers, 1960, p. 2-2). That is, network analyzers dealt with partial differential equations (PDEs), whereas active-element computers, such as the differential analyzer and its electronic successors, were restricted to ordinary differential equations (ODEs) in which time was the independent variable. Large network analyzers are early examples of *analog field computers*.

Electronic analog computers became feasible after the invention of the *DC operational amplifier* ("op amp") c. 1940 (Small, 2001, pp. 64, 67–72). Already in the 1930s scientists at Bell Telephone Laboratories (BTL) had developed the DC-coupled feedback-stabilized amplifier, which is the basis of the op amp. In 1940, as the USA prepared to enter World War II, D. L. Parkinson at BTL had a dream in which he saw DC amplifiers being used to control an anti-aircraft gun. As a consequence, with his colleagues C. A. Lovell and B. T. Weber, he wrote a series of papers on "electrical mathematics," which described electrical circuits to "operationalize" addition, subtraction, integration, differentiation, etc. The project to produce an electronic gun-director led to the development and refinement of DC op amps suitable for analog computation.

The war-time work at BTL was focused primarily on control applications of analog devices, such as the gun-director. Other researchers, such as E. Lakatos at BTL, were more interested in applying them to general-purpose analog computation for science and engineering, which resulted in the design of the General Purpose Analog Computer (GPAC), also called "Gypsy," completed in 1949 (Small, 2001, pp. 69–71). Building on the BTL op amp design, fundamental work on electronic analog computation was conducted at Columbia University in the 1940s. In particular, this research showed how analog computation could be applied to the simulation of dynamic systems and to the solution of nonlinear equations.

Commercial general-purpose analog computers (GPACs) emerged in the late 1940s and early 1950s (Small, 2001, pp. 72–3) (Fig. V.10). Typically they provided several dozen integrators, but several GPACs could be connected together to solve larger problems. Later, large-scale GPACs might have up to 500 amplifiers and compute with 0.01%–0.1% precision (Truitt & Rogers, 1960, pp. 2–33).

Besides integrators, typical GPACs provided adders, subtracters, multi-

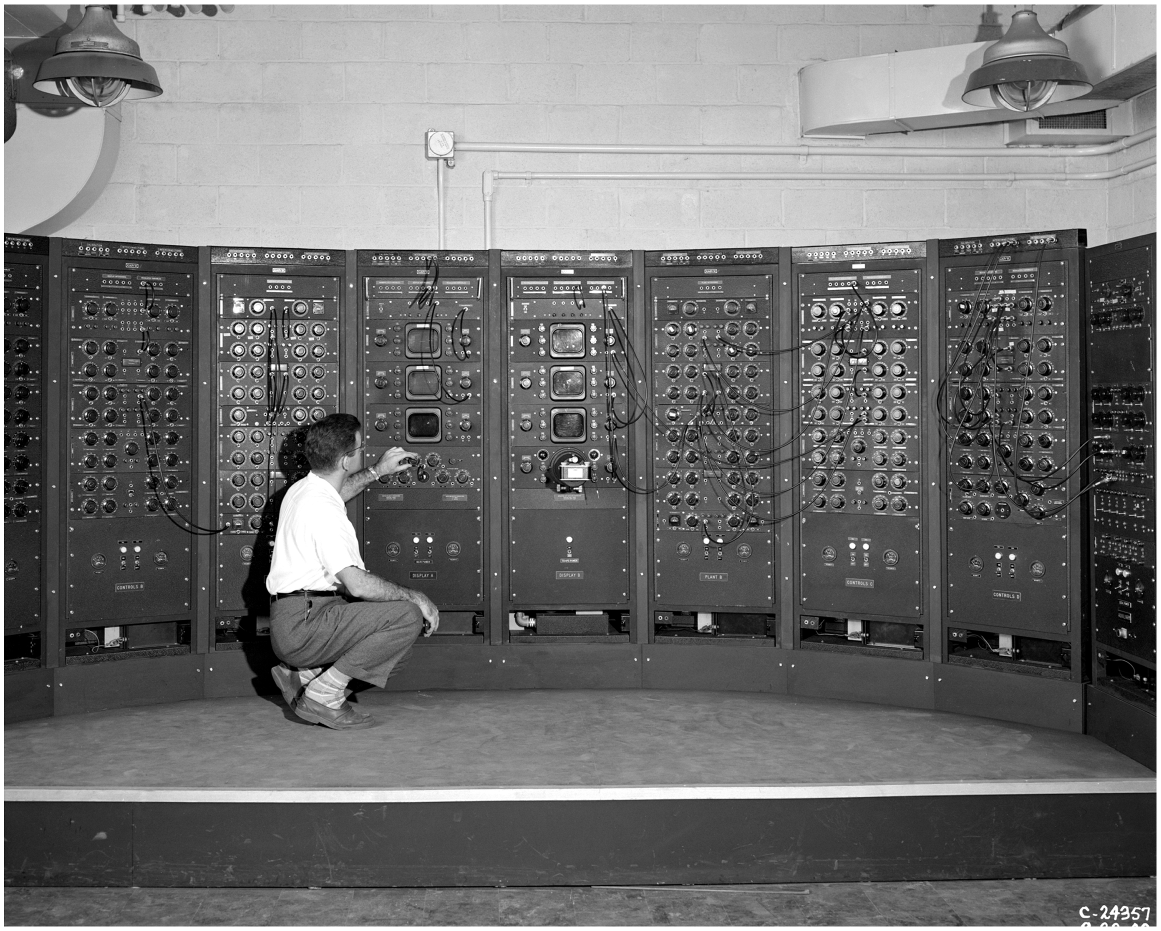


Figure V.10: Analog computer at Lewis Flight Propulsion Laboratory circa 1949.

pliers, fixed function generators (e.g., logarithms, exponentials, trigonometric functions), and variable function generators (for user-defined functions) (Truitt & Rogers, 1960, chs. 1.3, 2.4). A GPAC was programmed by connecting these components together, often by means of a patch panel. In addition, parameters could be set by adjusting potentiometers (attenuators), and arbitrary functions could be entered in the form of graphs (Truitt & Rogers, 1960, pp. 1-72–81, 2-154–156). Output devices plotted data continuously or displayed it numerically (Truitt & Rogers, 1960, pp. 3-1–30).

The most basic way of using a GPAC was in *single-shot mode* (Weyrick, 1969, pp. 168–70). First, parameters and initial values were entered into the potentiometers. Next, putting a master switch in "reset" mode controlled relays to apply the initial values to the integrators. Turning the switch to "operate" or "compute" mode allowed the computation to take place (i.e., the integrators to integrate). Finally, placing the switch in "hold" mode stopped the computation and stabilized the values, allowing them to be read from the computer (e.g., on voltmeters). Although single-shot operation was also called "slow operation" (in comparison to "repetitive operation," discussed next), it was in practice quite fast. Because all of the devices computed in parallel and at electronic speeds, analog computers usually solved problems in real-time but often much faster (Truitt & Rogers 1960, pp. 1-30–32, Small 2001, p. 72).

One common application of GPACs was to explore the effect of one or more parameters on the behavior of a system. To facilitate this exploration of the parameter space, some GPACs provided a *repetitive operation mode*, which worked as follows (Weyrick 1969, p. 170, Small 2001, p. 72). An electronic clock switched the computer between reset and compute modes at an adjustable rate (e.g., 10–1000 cycles per second) (Ashley, 1963, p. 280, n. 1). In effect the simulation was rerun at the clock rate, but if any parameters were adjusted, the simulation results would vary along with them. Therefore, within a few seconds, an entire family of related simulations could be run. More importantly, the operator could acquire an intuitive understanding of the system's dependence on its parameters.

B.1.c The eclipse of analog computing

A common view is that electronic analog computers were a primitive predecessor of the digital computer, and that their use was just a historical episode, or even a digression, in the inevitable triumph of digital technology. It is supposed that the current digital hegemony is a simple matter of technological superiority. However, the history is much more complicated, and involves a number of social, economic, historical, pedagogical, and also technical factors, which are outside the scope of this book (see Small 1993 and Small 2001, especially ch. 8, for more information). In any case, beginning after World War II and continuing for twenty-five years, there was lively debate about the relative merits of analog and digital computation.

Speed was an oft-cited advantage of analog computers (Small, 2001, ch. 8). While early digital computers were much faster than mechanical differential analyzers, they were slower (often by several orders of magnitude) than electronic analog computers. Furthermore, although digital computers could perform individual arithmetic operations rapidly, complete problems were solved sequentially, one operation at a time, whereas analog computers operated in parallel. Thus it was argued that increasingly large problems required more time to solve on a digital computer, whereas on an analog computer they might require more hardware but not more time. Even as digital computing speed was improved, analog computing retained its advantage for several decades, but this advantage eroded steadily.

Another important issue was the comparative *precision* of digital and analog computation (Small, 2001, ch. 8). Analog computers typically computed with three or four digits of precision, and it was very expensive to do much better, due to the difficulty of manufacturing the parts and other factors. In contrast, digital computers could perform arithmetic operations with many digits of precision, and the hardware cost was approximately proportional to the number of digits. Against this, analog computing advocates argued that many problems did not require such high precision, because the measurements were known to only a few significant figures and the mathematical models were approximations. Further, they distinguished between precision and *accuracy*, which refers to the conformity of the computation to physical reality, and they argued that digital computation was often less accurate than analog, due to numerical limitations (e.g., truncation, cumulative error in numerical integration). Nevertheless, some important applications, such as the calculation of missile trajectories, required greater precision, and for these, digital computation had the advantage. Indeed, to some extent precision was viewed as inherently desirable, even in applications where it was unimportant, and it was easily mistaken for accuracy. (See Sec. C.4.a for more on precision and accuracy.)

There was even a social factor involved, in that the written programs, precision, and exactness of digital computation were associated with mathematics and science, but the hands-on operation, parameter variation, and approximate solutions of analog computation were associated with engineering, and so analog computing inherited "the lower status of engineering *vis-à-vis* science" (Small, 2001, p. 251). Thus the status of digital computing was further enhanced as engineering became more mathematical and scientific after World War II (Small, 2001, pp. 247–51).

Already by the mid-1950s the competition between analog and digital had evolved into the idea that they were complementary technologies. This resulted in the development of a variety of *hybrid* analog/digital computing systems (Small, 2001, pp. 251–3, 263–6). In some cases this involved using a digital computer to control an analog computer by using digital logic to connect the analog computing elements, to set parameters, and to gather data. This improved the accessibility and usability of analog computers, but had the disadvantage of distancing the user from the physical analog system. The intercontinental ballistic missile program in the USA stimulated the further development of hybrid computers in the late 1950s and 1960s (Small, 1993). These applications required the speed of analog computation to simulate the closed-loop control systems and the precision of digital computation for accurate computation of trajectories. However, by the early 1970s hybrids were being displaced by all-digital systems. Certainly part of the reason was the steady improvement in digital technology, driven by a vibrant digital computer industry, but contemporaries also pointed to an inaccurate perception that analog computing was obsolete and to a lack of education about the advantages and techniques of analog computing.

Another argument made in favor of digital computers was that they were general-purpose, since they could be used in business data processing and other application domains, whereas analog computers were essentially special-purpose, since they were limited to scientific computation (Small, 2001, pp. 248–50). Against this it was argued that *all* computing is essentially computing by analogy, and therefore analog computation was general-purpose because the class of analog computers included digital computers! (See also Sec. A on computing by analogy.) Be that as it may, analog computation, as normally understood, is restricted to continuous variables, and so it was not immediately applicable to discrete data, such as that manipulated in business computing and other nonscientific applications. Therefore business (and eventually consumer) applications motivated the computer industry's investment in digital computer technology at the expense of analog technology.

Although it is commonly believed that analog computers quickly disappeared after digital computers became available, this is inaccurate, for both general-purpose and special-purpose analog computers have continued to be used in specialized applications to the present time. For example, a general-purpose *electrical* (vs. electronic) analog computer, the Anacom, was still in use in 1991. This is not technological atavism, for "there is no doubt considerable truth in the fact that Anacom continued to be used because it effectively met a need in a historically neglected but nevertheless important computer application area" (Aspray, 1993). As mentioned, the reasons for the eclipse of analog computing were not simply the technological superiority of digital computation; the conditions were much more complex. Therefore a change in conditions has necessitated a reevaluation of analog technology.

B.1.d Analog VLSI

In the mid-1980s, Carver Mead, who already had made important contributions to digital VLSI technology, began to advocate for the development of analog VLSI (Mead, 1987, 1989). His motivation was that "the nervous system of even a very simple animal contains computing paradigms that are orders of magnitude more effective than are those found in systems made by humans" and that they "can be realized in our most commonly available technology—silicon integrated circuits" (Mead, 1989, p. xi). However, he argued, since these natural computation systems are analog and highly nonlinear, progress would require understanding neural information processing in animals and applying it in a new analog VLSI technology.

Because analog computation is closer to the physical laws by which all computation is realized (which are continuous), analog circuits often use fewer devices than corresponding digital circuits. For example, a four-quadrant adder (capable of adding two signed numbers) can be fabricated from four transistors (Mead, 1989, pp. 87–8), and a four-quadrant multiplier from nine to seventeen, depending on the required range of operation (Mead, 1989, pp. 90–6). Intuitions derived from digital logic about what is simple or complex to compute are often misleading when applied to analog computation. For example, two transistors are sufficient to compute the logarithm or exponential, five for the hyperbolic tangent (which is very useful in neural computation), and three for the square root (Mead, 1989, pp. 70–1, 97–9). Thus analog VLSI is an attractive approach to "post-Moore's Law computing" (see Sec.

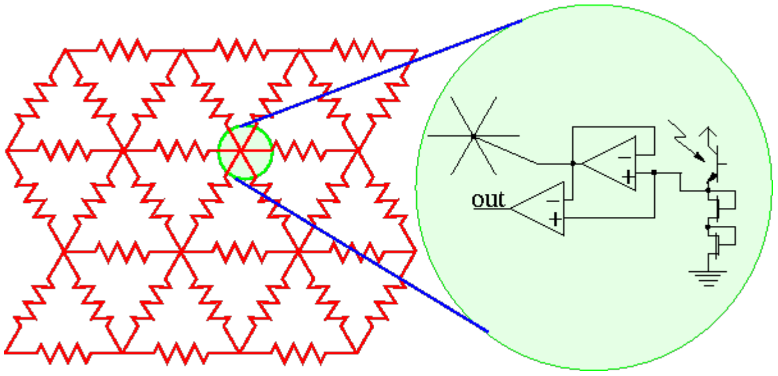


Figure V.11: Mahowald/Mead silicon retina.

H, p. 286 below). Mead and his colleagues demonstrated a number of analog VLSI devices inspired by the nervous system, including a "silicon retina" and an "electronic cochlea" (Fig. V.11) (Mead, 1989, chs. 15–16), research that has lead to a renaissance of interest in electronic analog computing.

B.1.e Field-programmable analog arrays

Field Programmable Analog Arrays (FPAAs) permit the programming of analog VLSI systems comparable to Field Programmable Gate Arrays (FPGAs) for digital systems (Fig. V.12). An FPAA comprises a number of identical Computational Analog Blocks (CABs), each of which contains a small number of analog computing elements. Programmable switching matrices control the interconnections among the elements of a CAB and the interconnections between the CABs. Contemporary FPAAs make use of floating-gate transistors, in which the gate has no DC connection to other circuit elements and thus is able to hold a charge indefinitely $\boxed{\textbf{cite}}$. Therefore the floating gate can be used to store a continuous value that governs the impedance of the transistor by several orders of magnitude. The gate charge can be changed by processes such as electron tunneling, which increases the charge, and hot-electron injection, which decreases it. Digital decoders allow individual floating-gate transistors in the switching matrices to be addressed and programmed. At the extremes of zero and infinite impedance the transistors operate as perfect switches, connecting or disconnecting circuit elements. Programming the connections to these extreme values is time consuming,

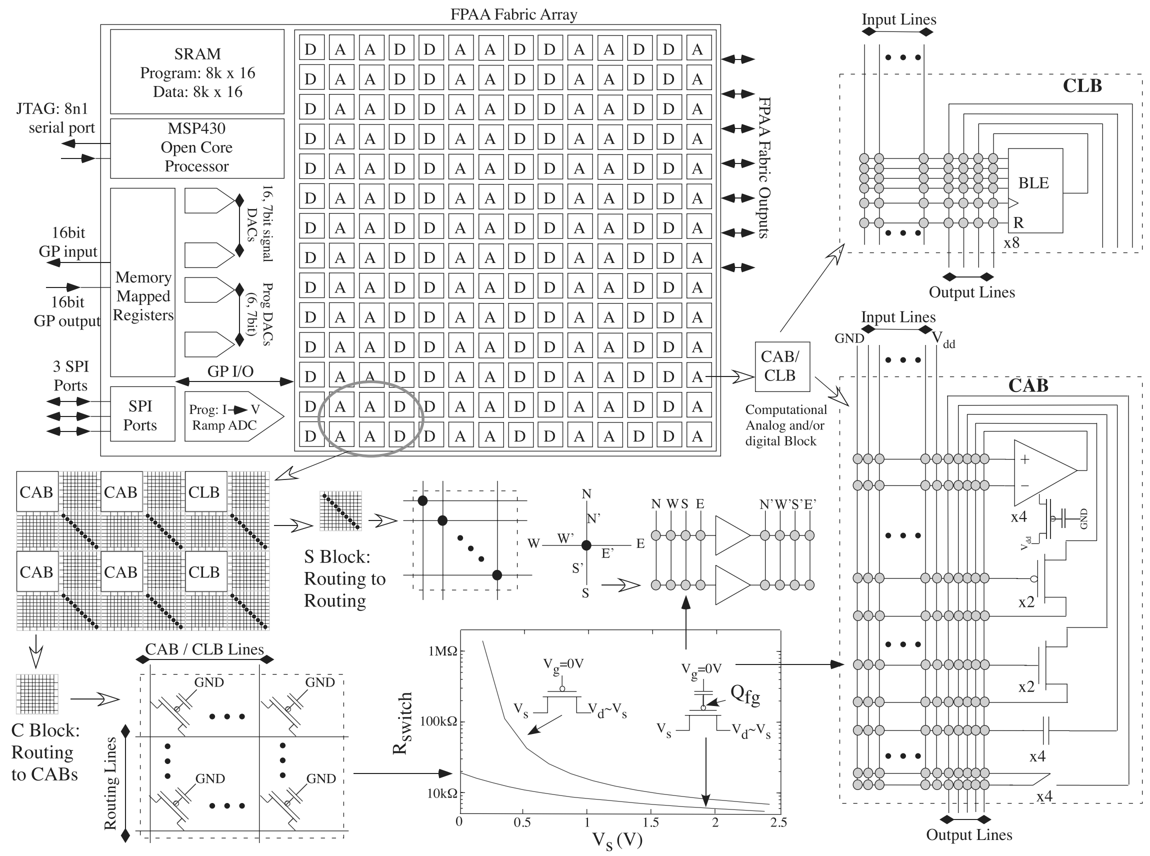


Figure V.12: RASP 3.0 FPAA. "RASP 3.0 integrates divergent concepts from multiple previous FPAA designs … along with low-power digital computation, including a 16-bit microprocessor (μP), interface circuitry, and DACs + ADCs. The FPAA SoC die photo measures 12 mm × 7 mm, fabricated in a 350-nm standard CMOS process. The die photo identifies μP, SRAM memory, DACs, and programming (DACs + ADC) infrastructure; the mixed array of the FPAA fabric is composed of interdigitated analog (A) and digital (D) configurable blocks on a single routing grid. DACs and programming infrastructure are accessed through memory-mapped registers." (George et al., 2016)

however, and so in practice some tradeoff is made between programming time and switch impedance. Each CAB contains several Operational Transconductance Amplifiers (OTAs), which are op-amps whose gain is controlled by a bias current. They are the principal analog computing elements, since they can be used for operations such as integration, differentiation, and gain amplification. Other computing elements may include tunable band-pass filters, which can be used for Fourier signal processing, and small matrix-vector multipliers, which can be used to implement linear operators. Current FPAAs can compute with a resolution of 10 bits (precision of 10^{-3}).

B.1.f Non-electronic analog computation

As will be explained later in this chapter, analog computation suggests many opportunities for future computing technologies. Many physical phenomena are potential media for analog computation provided they have useful mathematical structure (i.e., the mathematical laws describing them are mathematical functions useful for general- or special-purpose computation), and they are sufficiently controllable for practical use.

B.2 Chapter roadmap

The remainder of this chapter will begin by summarizing the fundamentals of analog computing, starting with the continuous state space and the various processes by which analog computation can be organized in time. Next it will discuss analog computation in nature, which provides models and inspiration for many contemporary uses of analog computation, such as neural networks. Then we consider general-purpose analog computing, both from a theoretical perspective and in terms of practical general-purpose analog computers. This leads to a discussion of the theoretical power of analog computation and in particular to the issue of whether analog computing is in some sense more powerful than digital computing. We briefly consider the cognitive aspects of analog computing, and whether it leads to a different approach to computation than does digital computing. Finally, we conclude with some observations on the role of analog computation in "post-Moore's Law computing."

C Fundamentals of analog computing

C.1 Continuous state space

As discussed in Sec. B, the fundamental characteristic that distinguishes analog from digital computation is that the state space is continuous in analog computation and discrete in digital computation. Therefore it might be more accurate to call analog and digital computation *continuous* and *discrete computation*, respectively. Furthermore, since the earliest days there have been *hybrid computers* that combine continuous and discrete state spaces and processes. Thus, there are several respects in which the state space may be continuous.

In the simplest case the state space comprises a finite (generally modest) number of variables, each holding a continuous quantity (e.g., voltage, current, charge). In a traditional GPAC they correspond to the variables in the ODEs defining the computational process, each typically having some independent meaning in the analysis of the problem. Mathematically, the variables are taken to contain bounded real numbers, although complex-valued variables are also possible (e.g., in AC electronic analog computers). In a practical sense, however, their precision is limited by noise, stability, device tolerance, and other factors (discussed below, Sec. C.4).

In typical analog neural networks the state space is larger in dimension but more structured than in traditional analog computers. The artificial neurons are organized into one or more layers, each composed of a (possibly large) number of artificial neurons. Commonly each layer of neurons is densely connected to the next layer (i.e., each neuron in one layer is connected to every neuron in the next). In general the layers each have some meaning in the problem domain, but the individual neurons constituting them do not (and so, in mathematical descriptions, the neurons are typically numbered rather than named).

The individual artificial neurons usually perform a simple computation such as this:

$$y = \sigma(s), \text{ where } s = b + \sum_{i=1}^{n} w_i x_i,$$

and where y is the activity of the neuron, $x_1, \ldots, x_n$ are the activities of the neurons that provide its inputs, b is a bias term, and $w_1, \ldots, w_n$ are the *weights* or strengths of the connections. Often the *activation function* σ is a real-valued *sigmoid* ("S-shaped") function, such as the *logistic sigmoid*,

$$\sigma(s) = \frac{1}{1 + e^{-s}},$$

in which case the neuron activity y is a real number, but some applications use a discontinuous *threshold function*, such as the *Heaviside function*,

$$U(s) = \begin{cases} +1 & , \quad \text{if } s \geq 0 \\ 0 & , \quad \text{if } s < 0 \end{cases},$$

in which case the activity is a discrete quantity. The *saturated-linear* or *piecewise-linear* sigmoid is also used occasionally:

$$\sigma(s) = \begin{cases} +1 & , \quad \text{if } s > 1 \\ s & , \quad \text{if } 0 \leq s \leq 1 \\ 0 & , \quad \text{if } s < 0 \end{cases}.$$

Regardless of whether the activation function is continuous or discrete, the bias b and connection weights $w_1, \ldots, w_n$ are real numbers, as is the "net input" $s = b + \sum_i w_i x_i$ to the activation function. Analog computation may be used to evaluate the linear combination s and the activation function $\sigma(s)$, if it is real-valued. If it is discrete, analog computation can approximate it with a sufficiently sharp sigmoid. The biases and weights are normally determined by a learning algorithm (e.g., back-propagation), which is also a good candidate for analog implementation.

In summary, the continuous state space of a neural network includes the bias values and net inputs of the neurons and the interconnection strengths between the neurons. It also includes the activity values of the neurons, if the activation function is a real-valued sigmoid function, as is often the case. Often large groups ("layers") of neurons (and the connections between these groups) have some intuitive meaning in the problem domain, but typically the individual neuron activities, bias values, and interconnection weights do not (they are "sub-symbolic").

If we extrapolate the number of neurons in a layer to the continuum limit, we get a *field*, which may be defined as a spatially continuous distribution of continuous quantity. Treating a group of artificial or biological neurons as a continuous mass is a reasonable mathematical approximation if their number is sufficiently large and if their spatial arrangement is significant (as it generally is in the brain). Fields are especially useful in modeling *cortical* *maps*, in which information is represented by the pattern of activity over a region of neural cortex.

In field computation the state space in continuous in two ways: it is continuous in variation but also in space. Therefore, field computation is especially applicable to solving PDEs and to processing spatially extended information such as visual images. Some early analog computing devices were capable of field computation (Truitt & Rogers, 1960, pp. 1-14–17, 2-2–16). For example, as previously mentioned (Sec. B), large resistor and capacitor networks could be used for solving PDEs such as diffusion problems. In these cases a discrete ensemble of resistors and capacitors was used to approximate a continuous field, while in other cases the computing medium was spatially continuous. The latter made use of conductive sheets (for two-dimensional fields) or electrolytic tanks (for two- or three-dimensional fields). When they were applied to steady-state spatial problems, these analog computers were called *field plotters* or *potential analyzers*.

The ability to fabricate very large arrays of analog computing devices, combined with the need to exploit massive parallelism in realtime computation and control applications, creates new opportunities for field computation (MacLennan, 1987, 1990, 1999). There is also renewed interest in using physical fields in analog computation. For example, Rubel (1993) defined an abstract *extended analog computer* (EAC), which augments Shannon's (1941) general purpose analog computer with (unspecified) facilities for field computation, such as PDE solvers (see Secs. E.3–E.4 below). J. W. Mills has explored the practical application of these ideas in his *artificial neural field networks* and VLSI EACs, which use the diffusion of electrons in bulk silicon or conductive gels and plastics for 2D and 3D field computation (Mills, 1996; Mills et al., 2006).

C.2 Computational process

We have considered the continuous state space, which is the basis for analog computing, but there are a variety of ways in which analog computers can operate on the state. In particular, the state can change continuously in time or be updated at distinct instants (as in digital computation).

C.2.a Continuous time

Since the laws of physics on which analog computing is based are differential equations, many analog computations proceed in continuous real time. Also, as we have seen, an important application of analog computers in the late 19th and early 20th centuries was the integration of ODEs in which time is the independent variable. A common technique in analog simulation of physical systems is *time scaling*, in which the differential equations are altered systematically so the simulation proceeds either more slowly or more quickly than the primary system (see Sec. C.4 for more on time scaling). On the other hand, because analog computations are close to the physical processes that realize them, analog computing is rapid, which makes it very suitable for real-time control applications.

In principle, any mathematically describable physical process operating on time-varying physical quantities can be used for analog computation. In practice, however, analog computers typically provide familiar operations that scientists and engineers use in differential equations (Rogers & Connolly, 1960; Truitt & Rogers, 1960). These include basic arithmetic operations, such as algebraic sum and difference ($u(t) = v(t) \pm w(t)$), constant multiplication or scaling ($u(t) = cv(t)$), variable multiplication and division ($u(t) = v(t)w(t)$, $u(t) = v(t)/w(t)$), and inversion ($u(t) = -v(t)$). Transcendental functions may be provided, such as the exponential ($u(t) = \exp v(t)$), logarithm ($u(t) = \ln v(t)$), trigonometric functions ($u(t) = \sin v(t)$, etc.), and *resolvers* for converting between polar and rectangular coordinates. Most important, of course, is definite integration ($u(t) = v_0 + \int_0^t v(\tau)\mathrm{d}\tau$), but differentiation may also be provided ($u(t) = \dot{v}(t)$). Generally, however, direct differentiation is avoided, since noise tends to have a higher frequency than the signal, and therefore differentiation amplifies noise; typically problems are reformulated to avoid direct differentiation (Weyrick, 1969, pp. 26–7). As previously mentioned, many GPACs include *(arbitrary) function generators*, which allow the use of functions defined only by a graph and for which no mathematical definition might be available; in this way empirically defined functions can be used (Rogers & Connolly, 1960, pp. 32–42). Thus, given a graph $(x, f(x))$, or a sufficient set of samples, $(x_k, f(x_k))$, the function generator approximates $u(t) = f(v(t))$. Rather less common are generators for arbitrary functions of two variables, $u(t) = f(v(t), w(t))$, in which the function may be defined by a surface, $(x, y, f(x, y))$, or by sufficient samples from it.

Although analog computing is primarily continuous, there are situations in which discontinuous behavior is required. Therefore some analog computers provide *comparators*, which produce a discontinuous result depending on the relative value of two input values. For example,

$$u = \begin{cases} k & , \quad \text{if } v \geq w, \\ 0 & , \quad \text{if } v < w. \end{cases}$$

Typically, this would be implemented as a Heaviside (unit step) function applied to the difference of the inputs, $u = kU(v - w)$. In addition to allowing the definition of discontinuous functions, comparators provide a primitive decision making ability, and may be used, for example to terminate a computation (switching the computer from "operate" to "hold" mode).

Other operations that have proved useful in analog computation are time delays and noise generators (Howe, 1961, ch. 7). The function of a *time delay* is simply to retard the signal by an adjustable delay $T > 0$: $u(t + T) = v(t)$. One common application is to model delays in the primary system (e.g., human response time).

Typically a *noise generator* produces time-invariant Gaussian-distributed noise with zero mean and a flat power spectrum (over a band compatible with the analog computing process). The standard deviation can be adjusted by scaling, the mean can be shifted by addition, and the spectrum altered by filtering, as required by the application. Historically noise generators were used to model noise and other random effects in the primary system, to determine, for example, its sensitivity to effects such as turbulence. However, noise can make a positive contribution in some analog computing algorithms (e.g., for symmetry breaking and in simulated annealing, weight perturbation learning, and stochastic resonance).

As already mentioned, some analog computing devices for the direct solution of PDEs have been developed. In general a PDE solver depends on an analogous physical process, that is, on a process obeying the same class of PDEs that it is intended to solve. For example, in Mills' EAC, diffusion of electrons in conductive sheets or solids is used to solve diffusion equations (Mills, 1996; Mills et al., 2006). Historically, PDEs were solved on electronic GPACs by discretizing all but one of the independent variables, thus replacing the differential equations by difference equations (Rogers & Connolly, 1960, pp. 173–93). That is, computation over a field was approximated by computation over a finite real array.

Reaction-diffusion computation is an important example of continuous-time analog computing. The state is represented by a set of time-varying chemical concentration fields, $c_1, \ldots, c_n$. These fields are distributed across a one-, two-, or three-dimensional space Ω, so that, for $\mathbf{x} \in \Omega$, $c_k(\mathbf{x}, t)$ represents the concentration of chemical k at location $\mathbf{x}$ and time t. Computation proceeds in continuous time according to reaction-diffusion equations, which have the form:

$$\partial \mathbf{c} / \partial t = \mathrm{D} \nabla^2 \mathbf{c} + \mathbf{F}(\mathbf{c}),$$

where $\mathbf{c} = (c_1, \ldots, c_n)^{\mathrm{T}}$ is the vector of concentrations, $\mathrm{D} = \mathrm{diag}(d_1, \ldots, d_n)$ is a diagonal matrix of positive diffusion rates, and $\mathbf{F}$ is nonlinear vector function that describes how the chemical reactions affect the concentrations.

Some neural net models operate in continuous time and thus are examples of continuous-time analog computation. For example, Grossberg (Grossberg, 1967, 1973, 1976) defines the activity of a neuron by differential equations such as this:

$$\dot{x}_i = -a_i x_i + \sum_{j=1}^{n} b_{ij} w_{ij}^{(+)} f_j(x_j) - \sum_{j=1}^{n} c_{ij} w_{ij}^{(-)} g_j(x_j) + I_i.$$

This describes the continuous change in the activity of neuron i resulting from passive decay (first term), positive feedback from other neurons (second term), negative feedback (third term), and input (last term). The f_j and g_j are nonlinear activation functions, and the $w_{ij}^{(+)}$ and $w_{ij}^{(-)}$ are adaptable excitatory and inhibitory connection strengths, respectively.

The continuous Hopfield network is another example of continuous-time analog computation (Hopfield, 1984). The output y_i of a neuron is a nonlinear function of its internal state x_i, $y_i = \sigma(x_i)$, where the hyperbolic tangent is usually used as the activation function, $\sigma(x) = \tanh x$, because its range is $[-1, 1]$. The internal state is defined by a differential equation,

$$\tau_i \dot{x}_i = -a_i x_i + b_i + \sum_{j=1}^{n} w_{ij} y_j,$$

where τ_i is a time constant, a_i is the decay rate, b_i is the bias, and w_{ij} is the connection weight to neuron i from neuron j. In a Hopfield network every neuron is symmetrically connected to every other ($w_{ij} = w_{ji}$) but not to itself ($w_{ii} = 0$).

Of course analog VLSI implementations of neural networks also operate in continuous time (e.g., Mead, 1989; Fakhraie & Smith, 1997)

Concurrent with the resurgence of interest in analog computation have been innovative reconceptualizations of continuous-time computation. For example, Brockett (1988) has shown that dynamical systems can perform a number of problems normally considered to be intrinsically sequential. In particular, a certain system of ODEs (a *nonperiodic finite Toda lattice*) can sort a list of numbers by continuous-time analog computation. The system is started with the vector $\mathbf{x}$ equal to the values to be sorted and a vector $\mathbf{y}$ initialized to small nonzero values; the $\mathbf{y}$ vector converges to a sorted permutation of $\mathbf{x}$.

C.2.b Sequential time

Sequential-time computation refers to computation in which discrete computational operations take place in succession but at no definite interval (van Gelder, 1997). Ordinary digital computer programs take place in sequential time, for the operations occur one after another, but the individual operations are not required to have any specific duration, so long as they take finite time.

One of the oldest examples of sequential analog computation is provided by the compass-and-straightedge constructions of traditional Euclidean geometry (Sec. B). These computations proceed by a sequence of discrete operations, but the individual operations involve continuous representations (e.g., compass settings, straightedge positions) and operate on a continuous state (the figure under construction). Slide rule calculation might seem to be an example of sequential analog computation, but if we look at it, we see that although the operations are performed by an analog device, the intermediate results are recorded digitally (and so this part of the state space is discrete). Thus it is a kind of hybrid computation.

The familiar digital computer automates sequential digital computations that once were performed manually by human "computers." Sequential analog computation can be similarly automated. That is, just as the control unit of an ordinary digital computer sequences digital computations, so a digital control unit can sequence analog computations. In addition to the analog computation devices (adders, multipliers, etc.), such a computer must provide variables and registers capable of holding continuous quantities between the sequential steps of the computation (see also Sec. C.2.c below).

The primitive operations of sequential-time analog computation are typically similar to those in continuous-time computation (e.g., addition, multiplication, transcendental functions), but integration and differentiation with respect to sequential time do not make sense. However, continuous-time integration within a single step, and space-domain integration, as in PDE solvers or field computation devices, are compatible with sequential analog computation.

In general, any model of digital computation can be converted to a similar model of sequential analog computation by changing the discrete state space to a continuum, and making appropriate changes to the rest of the model. For example, we can make an analog Turing machine by allowing it to write a bounded real number (rather than a symbol from a finite alphabet) onto a tape cell. The Turing machine's finite control can be altered to test for tape markings in some specified range.

Similarly, in a series of publications Blum, Shub, and Smale developed a theory of computation over the reals, which is an abstract model of sequential-time analog computation (Blum et al., 1998, 1988). In this "BSS model" programs are represented as flowcharts, but they are able to operate on real-valued variables. Using this model they were able to prove a number of theorems about the complexity of sequential analog algorithms.

The BSS model, and some other sequential analog computation models, assume that it is possible to make exact comparisons between real numbers (analogous to exact comparisons between integers or discrete symbols in digital computation) and to use the result of the comparison to control the path of execution. Comparisons of this kind are problematic because they imply infinite precision in the comparator (which may be defensible in a mathematical model but is impossible in physical analog devices), and because they make the execution path a discontinuous function of the state (whereas analog computation is usually continuous). Indeed, it has been argued that this is not "true" analog computation (Siegelmann, 1999, p. 148).

Many artificial neural network models are examples of sequential-time analog computation. In a simple feed-forward neural network, an input vector is processed by the layers in order, as in a pipeline. That is, the output of layer n becomes the input of layer $n + 1$. Since the model does not make any assumptions about the amount of time it takes a vector to be processed by each layer and to propagate to the next, execution takes place in sequential time. Most *recurrent* neural networks, which have feedback, also operate in sequential time, since the activities of all the neurons are updated synchronously (that is, signals propagate through the layers, or back to earlier layers, in lockstep).

Many artificial neural-net learning algorithms are also sequential-time analog computations. For example, the back-propagation algorithm updates a network's weights, moving sequentially backward through the layers.

In summary, the correctness of sequential time computation (analog or digital) depends on the *order* of operations, not on their *duration*, and similarly the efficiency of sequential computations is evaluated in terms of the *number* of operations, not on their *total duration*.

C.2.c Discrete time

Discrete-time analog computation has similarities to both continuous-time and sequential-time analog computation. Like the latter, it proceeds by a sequence of discrete (analog) computation steps; like the former, these steps occur at a constant rate in real time (e.g., some "frame rate"). If the real-time rate is sufficient for the application, then discrete-time computation can approximate continuous-time computation (including integration and differentiation).

Some electronic GPACs implemented discrete-time analog computation by a modification of repetitive operation mode, called *iterative analog computation* (Ashley, 1963, ch. 9). Recall (Sec. B.1.b) that in repetitive operation mode a clock rapidly switched the computer between reset and compute modes, thus repeating the same analog computation, but with different parameters (set by the operator). However, each repetition was independent of the others. Iterative operation was different in that analog values computed by one iteration could be used as initial values in the next. This was accomplished by means of an analog memory circuit (based on an op amp) that sampled an analog value at the end of one compute cycle (effectively during hold mode) and used it to initialize an integrator during the following reset cycle. (A modified version of the memory circuit could be used to retain a value over several iterations.) Iterative computation was used for problems such as determining, by iterative search or refinement, the initial conditions that would lead to a desired state at a future time. Since the analog computations were iterated at a fixed clock rate, iterative operation is an example of discrete-time analog computation. However, the clock rate is not directly relevant in some applications (such as the iterative solution of boundary value problems), in which case iterative operation is better characterized as sequential analog computation.

The principal contemporary examples of discrete-time analog computing are in neural network applications to time-series analysis and (discrete-time) control. In each of these cases the input to the neural net is a sequence of discrete-time samples, which propagate through the net and generate discrete-time output signals. Many of these neural nets are recurrent, that is, values from later layers are fed back into earlier layers, which allows the net to remember information from one sample to the next.

C.3 Analog computer programs

The concept of a *program* is central to digital computing, both practically, for it is the means for programming general-purpose digital computers, and theoretically, for it defines the limits of what can be computed by a universal machine, such as a universal Turing machine. Therefore it is important to discuss means for describing or specifying analog computations.

Traditionally, analog computers were used to solve ODEs (and sometimes PDEs), and so in one sense a mathematical differential equation is one way to represent an analog computation. However, since the equations were usually not suitable for direct solution on an analog computer, the process of *programming* involved the translation of the equations into a schematic diagram showing how the analog computing devices (integrators etc.) should be connected to solve the problem. These diagrams are the closest analogies to digital computer programs and may be compared to flowcharts, which were once popular in digital computer programming. It is worth noting, however, that flowcharts (and ordinary computer programs) represent sequences among operations, whereas analog computing diagrams represent functional relationships among variables, and therefore a kind of parallel data flow.

Differential equations and schematic diagrams are suitable for continuous-time computation, but for sequential analog computation something more akin to a conventional digital program can be used. Thus, as previously discussed (Sec. C.2.b), the BSS system uses flowcharts to describe sequential computations over the reals. Similarly, Moore (1996) defines recursive functions over the reals by means of a notation similar to a programming language.

In principle any sort of analog computation might involve constants that are arbitrary real numbers, which therefore might not be expressible in finite form (e.g., as a finite string of digits). Although this is of theoretical interest (see Sec. F.3 below), from a practical standpoint these constants could be set with about at most four digits of precision (Rogers & Connolly, 1960, p. 11). Indeed, automatic potentiometer-setting devices were constructed that read a series of decimal numerals from punched paper tape and used them to set the potentiometers for the constants (Truitt & Rogers, 1960, pp. 3-58–60). Nevertheless it is worth observing that analog computers do allow continuous inputs that need not be expressed in digital notation, for example, when the parameters of a simulation are continuously varied by the operator. In principle, therefore, an analog program can incorporate constants that are represented by a real-valued physical quantity (e.g., an angle or a distance), which need not be expressed digitally. Further, as we have seen (Sec. B.1.b), some electronic analog computers could compute a function by means of an arbitrarily drawn curve, that is, not represented by an equation or a finite set of digitized points. Therefore, in the context of analog computing it is natural to expand the concept of a program beyond discrete symbols to include continuous representations (scalar magnitudes, vectors, curves, shapes, surfaces, etc.).

Typically such continuous representations would be used as adjuncts to conventional discrete representations of the analog computational process, such as equations or diagrams. However, in some cases the most natural static representation of the process is itself continuous, in which case it is more like a "guiding image" than a textual prescription (MacLennan, 1995). A simple example is a potential surface, which defines a continuum of trajectories from initial states (possible inputs) to fixed-point attractors (the results of the computations). Such a "program" may define a deterministic computation (e.g., if the computation proceeds by gradient descent), or it may constrain a nondeterministic computation (e.g., if the computation may proceed by any potential-decreasing trajectory). Thus analog computation suggests a broadened notion of programs and programming.

C.4 Characteristics of analog computation

C.4.a Precision

Analog computation is evaluated in terms of both accuracy and precision, but the two must be distinguished carefully (Ashley 1963, pp. 25–8, Weyrick 1969, pp. 12–13, Small 2001, pp. 257–61). *Accuracy* refers primarily to the relationship between a simulation and the primary system it is simulating or, more generally, to the relationship between the results of a computation and the mathematically correct result. Accuracy is a result of many factors, including the mathematical model chosen, the way it is set up on a computer, and the precision of the analog computing devices. *Precision*, therefore, is a narrower notion, which refers to the quality of a representation or computing device. In analog computing, precision depends on *resolution* (fineness of operation) and *stability* (absence of drift), and may be measured as a fraction of the represented value. Thus a precision of 0.01% means that the representation will stay within 0.01% of the represented value for a reasonable period of time. For purposes of comparing analog devices, the precision is usually expressed as a fraction of *full-scale variation* (i.e., the difference between the maximum and minimum representable values).

It is apparent that the precision of analog computing devices depends on many factors. One is the choice of physical process and the way it is utilized in the device. For example a linear mathematical operation can be realized by using a linear region of a nonlinear physical process, but the realization will be approximate and have some inherent imprecision. Also, associated, unavoidable physical effects (e.g., loading, and leakage and other losses) may prevent precise implementation of an intended mathematical function. Further, there are fundamental physical limitations to resolution (e.g., quantum effects, diffraction). Noise is inevitable, both intrinsic (e.g., thermal noise) and extrinsic (e.g., ambient radiation). Changes in ambient physical conditions, such as temperature, can affect the physical processes and decrease precision. At slower time scales, materials and components age and their physical characteristics change. In addition, there are always technical and economic limits to the control of components, materials, and processes in analog device fabrication.

The precision of analog and digital computing devices depend on very different factors. The precision of a (binary) digital device depends on the number of bits, which influences the amount of hardware, but not its quality. For example, a 64-bit adder is about twice the size of a 32-bit adder, but can made out of the same components. At worst, the size of a digital device might increase with the square of the number of bits of precision. This is because binary digital devices only need to represent two states, and therefore they can operate in saturation. The fabrication standards sufficient for the first bit of precision are also sufficient for the 64th bit. Analog devices, in contrast, need to be able to represent a continuum of states precisely. Therefore, the fabrication of high-precision analog devices is much more expensive than low-precision devices, since the quality of components, materials, and processes must be much more carefully controlled. Doubling the precision of an analog device may be expensive, whereas the cost of each additional bit of digital precision is incremental; that is, the cost is proportional to the logarithm of the precision expressed as a fraction of full range.

The forgoing considerations might seem to be a convincing argument for the superiority of digital to analog technology, and indeed they were an important factor in the competition between analog and digital computers in the middle of the twentieth century (Small, 2001, pp. 257–61). However, as was argued at that time, many computer applications do not require high precision. Indeed, in many engineering applications, the input data are known to only a few digits, and the equations may be approximate or derived from experiments. In these cases the very high precision of digital computation is unnecessary and may in fact be misleading (e.g., if one displays all 14 digits of a result that is accurate to only three). Furthermore, many applications in image processing and control do not require high precision. More recently, research in artificial neural networks (ANNs) has shown that low-precision analog computation is sufficient for almost all ANN applications. Indeed, neural information processing in the brain seems to operate with very low precision — perhaps less than 10% (McClelland et al., 1986, p. 378) — for which it compensates with massive parallelism. For example, by *coarse coding* a population of low-precision devices can represent information with relatively high precision (Rumelhart et al. 1986, pp. 91–6, Sanger 1996).

C.4.b Scaling

An important aspect of analog computing is *scaling*, which is used to adjust a problem to an analog computer. First is *time scaling*, which adjusts a problem to the characteristic time scale at which a computer operates, which is a consequence of its design and the physical processes by which it is realized (Peterson 1967, pp. 37–44, Rogers & Connolly 1960, pp. 262–3, Weyrick 1969, pp. 241–3). For example, we might want a simulation to proceed on a very different time scale from the primary system. Thus a weather or economic simulation should proceed faster than real time in order to get useful predictions. Conversely, we might want to slow down a simulation of protein folding so that we can observe the stages in the process. Also, for accurate results it is necessary to avoid exceeding the maximum response rate of the analog devices, which might dictate a slower simulation speed. On the other hand, too slow a computation might be inaccurate as a consequence of instability (e.g., drift and leakage in the integrators).

Time scaling affects only time-dependant operations such as integration. For example, suppose t, time in the primary system or "problem time," is related to τ, time in the computer, by $\tau = \beta t$. Therefore, an integration $u(t) = \int_0^t v(t')\mathrm{d}t'$ in the primary system is replaced by the integration $u(\tau) = \beta^{-1} \int_0^\tau v(\tau')\mathrm{d}\tau'$ on the computer. Thus time scaling may be accomplished simply by decreasing the input gain to the integrator by a factor of β.

Fundamental to analog computation is the representation of a continuous quantity in the primary system by a continuous quantity in the computer. For example, a displacement x in meters might be represented by a potential V in volts. The two are related by an *amplitude* or *magnitude scale factor*, $V = \alpha x$, (with units volts/meter), chosen to meet two criteria (Ashley 1963, pp. 103–6, Peterson 1967, ch. 4, Rogers & Connolly 1960, pp. 127–8, Weyrick 1969, pp. 233–40). On the one hand, α must be sufficiently small so that the range of the problem variable is accommodated within the range of values supported by the computing device. Exceeding the device's intended operating range may lead to inaccurate results (e.g., forcing a linear device into nonlinear behavior). On the other hand, the scale factor should not be too small, or relevant variation in the problem variable will be less than the resolution of the device, also leading to inaccuracy. (Recall that precision is specified as a fraction of full-range variation.)

In addition to the explicit variables of the primary system, there are implicit variables, such as the time derivatives of the explicit variables, and scale factors must be chosen for them too. For example, in addition to displacement x, a problem might include velocity $\dot{x}$ and acceleration $\ddot{x}$. Therefore, scale factors α, α', and α'' must be chosen so that αx, $\alpha' \dot{x}$, and $\alpha'' \ddot{x}$ have an appropriate range of variation (neither too large nor too small).

Once a scale factor has been chosen, the primary system equations are adjusted to obtain the analog computing equations. For example, if we have scaled $u = \alpha x$ and $v = \alpha' \dot{x}$, then the integration $x(t) = \int_0^t \dot{x}(t')\mathrm{d}t'$ would be computed by scaled equation:

$$u(t) = \frac{\alpha}{\alpha'} \int_0^t v(t')\mathrm{d}t'.$$

This is accomplished by simply setting the input gain of the integrator to α/α'.

In practice, time scaling and magnitude scaling are not independent (Rogers & Connolly, 1960, p. 262). For example, if the derivatives of a variable can be large, then the variable can change rapidly, and so it may be necessary to slow down the computation to avoid exceeding the high-frequency response of the computer. Conversely, small derivatives might require the computation to be run faster to avoid integrator leakage etc. Appropriate scale factors are determined by considering both the physics and the mathematics of the problem (Peterson, 1967, pp. 40–4). That is, first, the physics of the primary system may limit the ranges of the variables and their derivatives. Second, analysis of the mathematical equations describing the system can give additional information on the ranges of the variables. For example, in some cases the natural frequency of a system can be estimated from the coefficients of the differential equations; the maximum of the nth derivative is then estimated as the n power of this frequency (Peterson 1967, p. 42, Weyrick 1969, pp. 238–40). In any case, it is not necessary to have accurate values for the ranges; rough estimates giving orders of magnitude are adequate.

It is tempting to think of magnitude scaling as a problem unique to analog computing, but before the invention of floating-point numbers it was also necessary in digital computer programming. In any case it is an essential aspect of analog computing, in which physical processes are more directly used for computation than they are in digital computing. Although the necessity of scaling has been a source of criticism, advocates for analog computing have argued that it is a blessing in disguise, because it leads to improved understanding of the primary system, which was often the goal of the computation in the first place (Bissell 2004, Small 2001, ch. 8). Practitioners of analog computing are more likely to have an intuitive understanding of both the primary system and its mathematical description (see Sec. G).

D Analog Computation in Nature

Computational processes—that is to say, information processing and control—occur in many living systems, most obviously in nervous systems, but also in the self-organized behavior of groups of organisms. In most cases natural computation is analog, either because it makes use of continuous natural processes, or because it makes use of discrete but stochastic processes. Several examples will be considered briefly.

D.1 Neural computation

In the past neurons were thought of binary computing devices, something like digital logic gates. This was a consequence of the "all or nothing" response of a neuron, which refers to the fact that it does or does not generate an *action potential* (voltage spike) depending, respectively, on whether its total input exceeds a threshold or not (more accurately, it generates an action potential if the membrane depolarization at the axon hillock exceeds the threshold and the neuron is not in its refractory period). Certainly some neurons (e.g., so-called "command neurons") do act something like logic gates. However, most neurons are analyzed better as analog devices, because the *rate* of impulse generation represents significant information. In particular, an *amplitude code*, the membrane potential near the axon hillock (which is a summation of the electrical influences on the neuron), is translated into a *rate code* for more reliable long-distance transmission along the axons. Nevertheless, the code is low precision (about one digit), since information theory shows that it takes at least N milliseconds (and probably more like $5N$ msec.) to discriminate N values (MacLennan, 1991). The rate code is translated back to an amplitude code by the synapses, since successive impulses release neurotransmitter from the axon terminal, which diffuses across the synaptic cleft to receptors. Thus a synapse acts as a leaky integrator to time-average the impulses.

As previously discussed (Sec. C.1), many artificial neural net models have real-valued neural activities, which correspond to rate-encoded axonal signals of biological neurons. On the other hand, these models typically treat the input connections as simple real-valued weights, which ignores the analog signal processing that takes place in the dendritic trees of biological neurons. The dendritic trees of many neurons are complex structures, which often have tens of thousands of synaptic inputs. The binding of neurotransmitters to receptors causes minute voltage fluctuations, which propagate along the membrane, and ultimately cause voltage fluctuations at the axon hillock, which influence the impulse rate. Since the dendrites have both resistance and capacitance, to a first approximation the signal propagation is described by the "cable equations," which describe passive signal propagation in cables of specified diameter, capacitance, and resistance (Anderson, 1995, ch. 1). Therefore, to a first approximation, a neuron's dendritic net operates as an adaptive linear analog filter with thousands of inputs, and so it is capable of quite complex signal processing. More accurately, however, it must be treated as a *nonlinear* analog filter, since voltage-gated ion channels introduce nonlinear effects. The extent of analog signal processing in dendritic trees is still poorly understood.

In most cases, then, neural information processing is treated best as low-precision analog computation. Although individual neurons have quite broadly tuned responses, accuracy in perception and sensorimotor control is achieved through coarse coding, as already discussed (Sec. C.4). Further, one widely used neural representation is the *cortical map*, in which neurons are systematically arranged in accord with one or more dimensions of their stimulus space, so that stimuli are represented by patterns of activity over the map. (Examples are *tonotopic maps*, in which pitch is mapped to cortical location, and *retinotopic maps*, in which cortical location represents retinal location.) Since neural density in the cortex is at least 146 000 neurons per square millimeter (Changeux, 1985, p. 51), even relatively small cortical maps can be treated as fields and information processing in them as analog field computation. Overall, the brain demonstrates what can be accomplished by massively parallel analog computation, even if the individual devices are comparatively slow and of low precision.

D.2 Adaptive self-organization in social insects

Another example of analog computation in nature is provided by the self-organizing behavior of social insects, microorganisms, and other populations (Camazine et al., 2001). Often such organisms respond to concentrations, or gradients in the concentrations, of chemicals produced by other members of the population. These chemicals may be deposited and diffuse through the environment. In other cases, insects and other organisms communicate by contact, but may maintain estimates of the relative proportions of different kinds of contacts. Because the quantities are effectively continuous, all these are examples of analog control and computation.

Self-organizing populations provide many informative examples of the use of natural processes for analog information processing and control. For example, diffusion of pheromones is a common means of self-organizzation in insect colonies, facilitating the creation of paths to resources, the construction of nests, and many other functions (Camazine et al., 2001). Real diffusion (as opposed to sequential simulations of it) executes, in effect, a massively parallel search of paths from the chemical's source to its recipients and allows the identification of near-optimal paths. Furthermore, if the chemical degrades, as is generally the case, then the system will be adaptive, in effect continually searching out the shortest paths, so long as source continues to function (Camazine et al., 2001). Simulated diffusion has been applied to robot path planning (Khatib, 1986; Rimon & Koditschek, 1989).

D.3 Genetic circuits

Another example of natural analog computing is provided by the *genetic regulatory networks* that control the behavior of cells, in multicellular organisms as well as single-celled ones (Davidson, 2006). These networks are defined by the mutually interdependent regulatory genes, promoters, and repressors that control the internal and external behavior of a cell. The interdependencies are mediated by proteins, the synthesis of which is governed by genes, and which in turn regulate the synthesis of other gene products (or themselves). Since it is the quantities of these substances that is relevant, many of the regulatory motifs can be described in computational terms as adders, subtracters, integrators, etc. Thus the genetic regulatory network implements an analog control system for the cell (Reiner, 1968).

It might be argued that the number of intracellular molecules of a particular protein is a (relatively small) discrete number, and therefore that it is inaccurate to treat it as a continuous quantity. However, the molecular processes in the cell are stochastic, and so the relevant quantity is the *probability* that a regulatory protein will bind to a regulatory site. Further, the processes take place in continuous real time, and so the rates are generally the significant quantities. Finally, although in some cases gene activity is either on or off (more accurately: very low), in other cases it varies continuously between these extremes (Hartl, 1994, pp. 388–90).

Embryological development combines the analog control of individual cells with the sort of self-organization of populations seen in social insects and other colonial organisms. Locomotion of the cells and the expression of specific genes is controlled by chemical signals, among other mechanisms (Davidson, 2006; Davies, 2005). Thus PDEs have proved useful in explaining some aspects of development; for example *reaction-diffusion equations* have been used to describe the formation of hair-coat patterns and related phenomena (Camazine et al., 2001; Maini & Othmer, 2001; Murray, 1977). Therefore the developmental process is governed by naturally occurring analog computation.

D.4 Is everything a computer?

It might seem that any continuous physical process could be viewed as analog computation, which would make the term almost meaningless. As the question has been put, is it meaningful (or useful) to say that the solar system is *computing* Kepler's laws? In fact, it is possible and worthwhile to make a distinction between computation and other physical processes that happen to be described by mathematical laws (MacLennan, 1994a,c, 2001, 2004).

If we recall the original meaning of analog computation (Sec. A), we see that the computational system is used to solve some mathematical problem with respect to a primary system. What makes this possible is that the computational system and the primary system have the same, or systematically related, abstract (mathematical) structures. Thus the computational system can inform us about the primary system, or be used to control it, etc. Although from a practical standpoint some analogs are better than others, in principle any physical system can be used that obeys the same equations as the primary system.

Based on these considerations we may define computation as a physical process the purpose of which is the abstract manipulation of abstract objects (i.e., information processing); this definition applies to analog, digital, and hybrid computation (MacLennan, 1994a,c, 2001, 2004). Therefore, to determine if a natural system is computational we need to look to its purpose or function within the context of the living system of which it is a part. One test of whether its function is the abstract manipulation of abstract objects is to ask whether it could still fulfill its function if realized by different physical processes, a property called *multiple realizability*. (Similarly, in artificial systems, a simulation of the economy might be realized equally accurately by a hydraulic analog computer or an electronic analog computer (Bissell, 2004).) By this standard, the majority of the nervous system is purely computational; in principle it could be replaced by electronic devices obeying the same differential equations. In the other cases we have considered (self-organization of living populations, genetic circuits) there are instances of both pure computation and computation mixed with other functions (for example, where the specific substances used have other—e.g. metabolic—roles in the living system).

E General-purpose analog computation

E.1 The importance of general-purpose computers

Although special-purpose analog and digital computers have been developed, and continue to be developed, for many purposes, the importance of general-purpose computers, which can be adapted easily for a wide variety of purposes, has been recognized since at least the nineteenth century. Babbage's plans for a general-purpose digital computer, his *analytical engine* (1835), are well known, but a general-purpose differential analyzer was advocated by Kelvin (Thomson, 1876). Practical general-purpose analog and digital computers were first developed at about the same time: from the early 1930s through the war years. General-purpose computers of both kinds permit the prototyping of special-purpose computers and, more importantly, permit the flexible reuse of computer hardware for different or evolving purposes.

The concept of a general-purpose computer is useful also for determining the limits of a computing paradigm. If one can design—theoretically or practically—a *universal computer*, that is, a general-purpose computer capable of simulating any computer in a relevant class, then anything uncomputable by the universal computer will also be uncomputable by any computer in that class. This is, of course, the approach used to show that certain functions are uncomputable by any Turing machine because they are uncomputable by a universal Turing machine. For the same reason, the concept of general-purpose analog computers, and in particular of *universal analog computers* are theoretically important for establishing limits to analog computation.

E.2 General-purpose electronic analog computers

Before taking up these theoretical issues, it is worth recalling that a typical electronic GPAC would include linear elements, such as adders, subtracters, constant multipliers, integrators, and differentiators; nonlinear elements, such as variable multipliers and function generators; other computational elements, such as comparators, noise generators, and delay elements (Sec. B.1.b). These are, of course, in addition to input/output devices, which would not affect its computational abilities.

E.3 Shannon's analysis

Claude Shannon did an important analysis of the computational capabilities of the differential analyzer, which applies to many GPACs (Shannon, 1941, 1993). He considered an abstract differential analyzer equipped with an unlimited number of integrators, adders, constant multipliers, and function generators (for functions with only a finite number of finite discontinuities), with at most one source of drive (which limits possible interconnections between units). This was based on prior work that had shown that almost all the generally used elementary functions could be generated with addition and integration. We will summarize informally a few of Shannon's results; for details, please consult the original paper.

First Shannon offers proofs that, by setting up the correct ODEs, a GPAC with the mentioned facilities can generate any function if and only if is not hypertranscendental (Theorem II); thus the GPAC can generate any function that is algebraic transcendental (a very large class), but not, for example, Euler's gamma function and Riemann's zeta function. He also shows that the GPAC can generate functions derived from generable functions, such as the integrals, derivatives, inverses, and compositions of generable functions (Thms. III, IV). These results can be generalized to functions of any number of variables, and to their compositions, partial derivatives, and inverses with respect to any one variable (Thms. VI, VII, IX, X).

Next Shannon shows that a function of any number of variables that is continuous over a closed region of space can be approximated arbitrarily closely over that region with a finite number of adders and integrators (Thms. V, VIII).

Shannon then turns from the generation of functions to the solution of ODEs and shows that the GPAC can solve any system of ODEs defined in terms of non-hypertranscendental functions (Thm. XI).

Finally, Shannon addresses a question that might seem of limited interest, but turns out to be relevant to the computational power of analog computers (see Sec. F below). To understand it we must recall that he was investigating the differential analyzer—a mechanical analog computer—but similar issues arise in other analog computing technologies. The question is whether it is possible to perform an arbitrary constant multiplication, $u = kv$, by means of gear ratios. He show that if we have just two gear ratios a and b ($a, b \neq 0, 1$), such that b is not a rational power of a, then by combinations of these gears we can approximate k arbitrarily closely (Thm. XII). That is, to approximate multiplication by arbitrary real numbers, it is sufficient to be able to multiply by a, b, and their inverses, provided a and b are not related by a rational power.

Shannon mentions an alternative method of constant multiplication, which uses integration, $kv = \int_0^v k\mathrm{d}v$, but this requires setting the integrand to the constant function k. Therefore, multiplying by an arbitrary real number requires the ability to input an arbitrary real as the integrand. The issue of real-valued inputs and outputs to analog computers is relevant both to their theoretical power and to practical matters of their application (see Sec. F.3).

Shannon's proofs, which were incomplete, were eventually refined by Pour-El (1974a) and finally corrected by Lipshitz & Rubel (1987). Rubel (1988) proved that Shannon's GPAC cannot solve the Dirichlet problem for Laplace's equation on the disk; indeed, it is limited to initial-value problems for algebraic ODEs. Specifically, *the Shannon–Pour-El Thesis* is that the outputs of the GPAC are exactly the solutions of the *algebraic differential equations*, that is, equations of the form

$$P[x, y(x), y'(x), y''(x), \ldots, y^{(n)}(x)] = 0,$$

where P is a polynomial that is not identically vanishing in any of its variables (these are the *differentially algebraic* functions) (Rubel, 1985). (For details please consult the cited papers.) The limitations of Shannon's GPAC motivated Rubel's definition of the Extended Analog Computer.

E.4 Rubel's Extended Analog Computer

The combination of Rubel's (1985) conviction that the brain is an analog computer together with the limitations of Shannon's GPAC led him to propose the *Extended Analog Computer* (EAC) (Rubel, 1993).

Like Shannon's GPAC (and the Turing machine), the EAC is a conceptual computer intended to facilitate theoretical investigation of the limits of a class of computers. The EAC extends the GPAC in a number of respects. For example, whereas the GPAC solves equations defined over a single variable (time), the EAC can generate functions over any finite number of real variables. Further, whereas the GPAC is restricted to initial-value problems for ODEs, the EAC solves both initial- and boundary-value problems for a variety of PDEs.

The EAC is structured into a series of levels, each more powerful than the ones below it, from which it accepts inputs. The inputs to the lowest level are a finite number of real variables ("settings"). At this level it operates on real polynomials, from which it is able to generate the differentially algebraic functions. The computation on each level is accomplished by conceptual analog devices, which include constant real-number generators, adders, multipliers, differentiators, "substituters" (for function composition), devices for analytic continuation, and inverters, which solve systems of equations defined over functions generated by the lower levels. Most characteristic of the EAC is the "boundary-value-problem box," which solves systems of PDEs and ODEs subject to boundary conditions and other constraints. The PDEs are defined in terms of functions generated by the lower levels. Such PDE solvers may seem implausible, and so it is important to recall field-computing devices for this purpose were implemented in some practical analog computers (see Sec. B.1) and more recently in Mills' EAC (Mills et al., 2006). As Rubel observed, PDE solvers could be implemented by physical processes that obey the same PDEs (heat equation, wave equation, etc.). (See also Sec. H.1 below.)

Finally, the EAC is required to be "extremely well-posed," which means that each level is relatively insensitive to perturbations in its inputs; thus "all the outputs depend in a strongly deterministic and stable way on the initial settings of the machine" (Rubel, 1993).

Rubel (1993) proves that the EAC can compute everything that the GPAC can compute, but also such functions as the gamma and zeta, and that it can solve the Dirichlet problem for Laplace's equation on the disk, all of which are beyond the GPAC's capabilities. Further, whereas the GPAC can compute differentially algebraic functions of time, the EAC can compute differentially algebraic functions of any finite number of real variables. In fact, Rubel did not find any real-analytic (C^∞) function that is *not* computable on the EAC, but he observes that if the EAC can indeed generate every real-analytic function, it would be too broad to be useful as a model of analog computation.

F Analog computation and the Turing limit

F.1 Introduction

The Church-Turing Thesis asserts that anything that is effectively computable is computable by a Turing machine, but the Turing machine (and equivalent models, such as the lambda calculus) are models of discrete computation, and so it is natural to wonder how analog computing compares in power, and in particular whether it can compute beyond the "Turing limit." Superficial answers are easy to obtain, but the issue is subtle because it depends upon choices among definitions, none of which is obviously correct, it involves the foundations of mathematics and its philosophy, and it raises epistemological issues about the role of models in scientific theories. This is an active research area, but many of the results are apparently inconsistent due to the differing assumptions on which they are based. Therefore this section will be limited to a mention of a few of the interesting results, but without attempting a comprehensive, systematic, or detailed survey; Siegelmann (1999) can serve as an introduction to the literature.

F.2 A sampling of theoretical results

F.2.a Continuous-time models

Orponen's (1997) survey of continuous-time computation theory is a good introduction to the literature as of that time; here we give a sample of these and more recent results.

There are several results showing that—under various assumptions—analog computers have at least the power of Turing machines (TMs). For example, Branicky (1994) showed that a TM could be simulated by ODEs, but he used non-differentiable functions; Bournez et al. (2006) provide an alternative construction using only analytic functions. They also prove that the GPAC computability coincides with (Turing-)computable analysis, which is surprising, since the gamma function is Turing-computable but, as we have seen, the GPAC cannot generate it. The paradox is resolved by a distinction between *generating* a function and *computing* it, with the latter, broader notion permitting convergent computation of the function (that is, as $t \to \infty$). However, the computational power of general ODEs has not been determined in general (Siegelmann, 1999, p. 149). M. B. Pour-El and I. Richards exhibit a Turing-computable ODE that does not have a Turing-computable solution (Pour-El & Richards, 1979, 1982). Stannett (1990) also defined a continuous-time analog computer that could solve the halting problem.

Moore (1996) defines a class of continuous-time recursive functions over the reals, which includes a zero-finding operator μ. Functions can be classified into a hierarchy depending on the number of uses of μ, with the lowest level (no μs) corresponding approximately to Shannon's GPAC. Higher levels can compute non-Turing-computable functions, such as the decision procedure for the halting problem, but he questions whether this result is relevant in the physical world, which is constrained by "noise, quantum effects, finite accuracy, and limited resources." Bournez & Cosnard (1996) have extended these results and shown that many dynamical systems have super-Turing power.

Omohundro (1984) showed that a system of ten coupled nonlinear PDEs could simulate an arbitrary cellular automaton, which implies that PDEs have at least Turing power. Further, D. Wolpert and B. J. MacLennan (Wolpert, 1991; Wolpert & MacLennan, 1993) showed that any TM can be simulated by a field computer with linear dynamics, but the construction uses Dirac delta functions. Pour-El and Richards exhibit a wave equation in three-dimensional space with Turing-computable initial conditions, but for which the unique solution is Turing-uncomputable (Pour-El & Richards, 1981, 1982).

F.2.b Sequential-time models

We will mention a few of the results that have been obtained concerning the power of sequential-time analog computation.

Although the BSS model has been investigated extensively, its power has not been completely determined (Blum et al., 1998, 1988). It is known to depend on whether just rational numbers or arbitrary real numbers are allowed in its programs (Siegelmann, 1999, p. 148).

A *coupled map lattice* (CML) is a cellular automaton with real-valued states; it is a sequential-time analog computer, which can be considered a discrete-space approximation to a simple sequential-time field computer. Orponen & Matamala (1996) showed that a finite CML can simulate a universal Turing machine. However, since a CML can simulate a BSS program or a recurrent neural network (see Sec. F.2.c below), it actually has super-Turing power (Siegelmann, 1999, p. 149).

Recurrent neural networks are some of the most important examples of sequential analog computers, and so the following section is devoted to them.

F.2.c Recurrent neural networks

With the renewed interest in neural networks in the mid-1980s, many investigators wondered if recurrent neural nets have super-Turing power. M. Garzon and S. Franklin showed that a sequential-time net with a countable infinity of neurons could exceed Turing power (Franklin & Garzon, 1990; Garzon & Franklin, 1989, 1990). Indeed, Siegelmann & Sontag (1994b) showed that finite neural nets with real-valued weights have super-Turing power, but Maass & Sontag (1999b) showed that recurrent nets with Gaussian or similar noise had *sub*-Turing power, illustrating again the dependence on these results on assumptions about what is a reasonable mathematical model of analog computing.

For recent results on recurrent neural networks, we will restrict our attention of the work of Siegelmann (1999), who addresses the computational power of these network in terms of the classes of languages they can recognize. Without loss of generality the languages are restricted to sets of binary strings. A string to be tested is fed to the network one bit at a time, along with an input that indicates when the end of the input string has been reached. The network is said to *decide* whether the string is in the language if it correctly indicates whether it is in the set or not, after some finite number of sequential steps since input began.

Siegelmann shows that, if exponential time is allowed for recognition, finite recurrent neural networks with real-valued weights (and saturated-linear activation functions) can compute *all* languages, and thus they are more powerful than Turing machines. Similarly, stochastic networks with rational weights also have super-Turing power, although less power than the deterministic nets with real weights. (Specifically, they compute P/POLY and BPP/log* respectively; see Siegelmann 1999, chs. 4, 9 for details.) She further argues that these neural networks serve as a "standard model" of (sequential) analog computation (comparable to Turing machines in Church-Turing computation), and therefore that the limits and capabilities of these nets apply to sequential analog computation generally.

Siegelmann (1999, p 156) observes that the super-Turing power of recurrent neural networks is a consequence of their use of non-rational real-valued weights. In effect, a real number can contain an infinite number of bits of information. This raises the question of how the non-rational weights of a network can ever be set, since it is not possible to define a physical quantity with infinite precision. However, although non-rational weights may not be able to be set from outside the network, they can be computed within the network by learning algorithms, which are analog computations. Thus, Siegelmann suggests, the fundamental distinction may be between *static computational models*, such as the Turing machine and its equivalents, and *dynamically evolving computational models*, which can tune continuously variable parameters and thereby achieve super-Turing power.

F.2.d Dissipative models

Beyond the issue of the power of analog computing relative to the Turing limit, there are also questions of its relative efficiency. For example, could analog computing solve NP-hard problems in polynomial or even linear time? In traditional computational complexity theory, efficiency issues are addressed in terms of the asymptotic number of computation steps to compute a function as the size of the function's input increases. One way to address corresponding issues in an analog context is by treating an analog computation as a *dissipative system*, which in this context means a system that decreases some quantity (analogous to energy) so that the system state converges to an *point attractor*. From this perspective, the initial state of the system incorporates the input to the computation, and the attractor represents its output. Therefore, H. T. Sieglemann, S. Fishman, and A. Ben-Hur have developed a complexity theory for dissipative systems, in both sequential and continuous time, which addresses the rate of convergence in terms of the underlying rates of the system (Ben-Hur et al., 2002; Siegelmann et al., 1999). The relation between dissipative complexity classes (e.g., P_d, NP_d) and corresponding classical complexity classes (P, NP) remains unclear (Siegelmann, 1999, p. 151).

F.3 Real-valued inputs, output, and constants

A common argument, with relevance to the theoretical power of analog computation, is that an input to an analog computer must be determined by setting a dial to a number or by typing a number into digital-to-analog conversion device, and therefore that the input will be a rational number. The same argument applies to any internal constants in the analog computation. Similarly, it is argued, any output from an analog computer must be measured, and the accuracy of measurement is limited, so that the result will be a rational number. Therefore, it is claimed, real numbers are irrelevant to analog computing, since any practical analog computer computes a function from the rationals to the rationals, and can therefore be simulated by a Turing machine.[2]

There are a number of interrelated issues here, which may be considered briefly. First, the argument is couched in terms of the input or output of *digital representations*, and the numbers so represented are necessarily rational (more generally, computable). This seems natural enough when we think of an analog computer as a calculating device, and in fact many historical analog computers were used in this way and had digital inputs and outputs (since this is our most reliable way of recording and reproducing quantities).

However, in many analog *control systems*, the inputs and outputs are continuous physical quantities that vary continuously in time (also a continuous physical quantity); that is, according to current physical theory, these quantities are real numbers, which vary according to differential equations. It is worth recalling that physical quantities are neither rational nor irrational; they can be so classified only in comparison with each other or with respect to a unit, that is, only if they are measured and digitally represented. Furthermore, physical quantities are neither computable nor uncomputable (in a Church-Turing sense); these terms apply only to discrete representations of these quantities (i.e., to numerals or other digital representations).

Therefore, in accord with ordinary mathematical descriptions of physical processes, analog computations can can be treated as having arbitrary real numbers (in some range) as inputs, outputs, or internal states; like other continuous processes, continuous-time analog computations pass through all the reals in some range, including non-Turing-computable reals. Paradoxically, however, these same physical processes can be simulated on digital computers.

F.4 The issue of simulation by Turing machines and digital computers

Theoretical results about the computational power, relative to Turing machines, of neural networks and other analog models of computation raise difficult issues, some of which are epistemological rather than strictly technical. On the one hand, we have a series of theoretical results proving the super-Turing power of analog computation models of various kinds. On the

[2]See related arguments by Martin Davis (2004, 2006).

other hand, we have the obvious fact that neural nets are routinely simulated on ordinary digital computers, which have at most the power of Turing machines. Furthermore, it is reasonable to suppose that any physical process that might be used to realize analog computation—and certainly the known processes—could be simulated on a digital computer, as is done routinely in computational science. This would seem to be incontrovertible proof that analog computation is no more powerful than Turing machines. The crux of the paradox lies, of course, in the non-Turing-computable reals. These numbers are a familiar, accepted, and necessary part of standard mathematics, in which physical theory is formulated, but from the standpoint of Church-Turing (CT) computation they do not exist. This suggests that the the paradox is not a contradiction, but reflects a divergence between the goals and assumptions of the two models of computation.

F.5 The problem of models of computation

These issues may be put in context by recalling that the Church-Turing (CT) model of computation is in fact a *model*, and therefore that it has the limitations of all models. A model is a cognitive tool that improves our ability to understand some class of phenomena by preserving relevant characteristics of the phenomena while altering other, irrelevant (or less relevant) characteristics. For example, a *scale model* alters the size (taken to be irrelevant) while preserving shape and other characteristics. Often a model achieves its purposes by making *simplifying* or *idealizing assumptions*, which facilitate analysis or simulation of the system. For example, we may use a linear mathematical model of a physical process that is only approximately linear. For a model to be effective it must preserve characteristics and make simplifying assumptions that are appropriate to the domain of questions it is intended to answer, its *frame of relevance* (MacLennan, 2004). If a model is applied to problems outside of its frame of relevance, then it may give answers that are misleading or incorrect, because they depend more on the simplifying assumptions than on the phenomena being modeled. Therefore we must be especially cautious applying a model outside of its frame of relevance, or even at the limits of its frame, where the simplifying assumptions become progressively less appropriate. The problem is aggravated by the fact that often the frame of relevance is not explicitly defined, but resides in a tacit background of practices and skills within some discipline.

Therefore, to determine the applicability of the CT model of computation to analog computing, we must consider the frame of relevance of the CT model. This is easiest if we recall the domain of issues and questions it was originally developed to address: issues of effective calculability and derivability in formalized mathematics. This frame of relevance determines many of the assumptions of the CT model, for example, that information is represented by finite discrete structures of symbols from a finite alphabet, that information processing proceeds by the application of definite formal rules at discrete instants of time, and that a computational or derivational process must be completed in a finite number of these steps.[3] Many of these assumptions are incompatible with analog computing and with the frames of relevance of many models of analog computation.

F.6 Relevant issues for analog computation

Analog computation is often used for control. Historically, analog computers were used in control systems and to simulate control systems, but contemporary analog VLSI is also frequently applied in control. Natural analog computation also frequently serves a control function, for example, sensorimotor control by the nervous system, genetic regulation in cells, and self-organized cooperation in insect colonies. Therefore, control systems delimit one frame of relevance for models of analog computation.

In this frame of relevance real-time response is a critical issue, which models of analog computation, therefore, ought to be able to address. Thus it is necessary to be able to relate the speed and frequency response of analog computation to the rates of the physical processes by which the computation is realized. Traditional methods of algorithm analysis, which are based on sequential time and asymptotic behavior, are inadequate in this frame of relevance. On the one hand, the constants (time scale factors), which reflect the underlying rate of computation are absolutely critical (but ignored in asymptotic analysis); on the other hand, in control applications the asymptotic behavior of algorithm is generally irrelevant, since the inputs are typically fixed in size or of a limited range of sizes.

The CT model of computation is oriented around the idea that the purpose of a computation is to evaluate a mathematical function. Therefore the basic criterion of adequacy for a computation is *correctness*, that is, that

[3]See MacLennan (2003, 2004) for a more detailed discussion of the frame of relevance of the CT model.

given a precise representation of an input to the function, it will produce (after finitely many steps) a precise representation of the corresponding output of the function. In the context of natural computation and control, however, other criteria may be equally or even more relevant. For example, *robustness* is important: how well does the system respond in the presence of noise, uncertainty, imprecision, and error, which are unavoidable in physical natural and artificial control systems, and how well does it respond to defects and damage, which arise in many natural and artificial contexts. Since the real world is unpredictable, *flexibility* is also important: how well does an artificial system respond to inputs for which it was not designed, and how well does a natural system behave in situations outside the range of those to which it is evolutionarily adapted. Therefore, *adaptability* (through learning and other means) is another important issue in this frame of relevance.[4]

F.7 Transcending Turing computability

Thus we see that many applications of analog computation raise different questions from those addressed by the CT model of computation; the most useful models of analog computing will have a different frame of relevance. In order to address traditional questions such as whether analog computers can compute "beyond the Turing limit," or whether they can solve NP-hard problems in polynomial time, it is necessary to construct models of analog computation within the CT frame of relevance. Unfortunately, constructing such models requires making commitments about many issues (such as the representation of reals and the discretization of time), that may affect the answers to these questions, but are fundamentally unimportant in the frame of relevance of the most useful applications of the concept of analog computation. Therefore, being overly focused on traditional problems in the theory of computation (which was formulated for a different frame of relevance) may distract us from formulating models of analog computation that can address important issues in its own frame of relevance.

[4]See MacLennan (2003, 2004) for a more detailed discussion of the frames of relevance of natural computation and control.

G Analog thinking

It will be worthwhile to say a few words about the *cognitive implications* of analog computing, which are a largely forgotten aspect of analog vs. digital debates of the late 20th century. For example, it was argued that analog computing provides a deeper intuitive understanding of a system than the alternatives do (Bissell 2004, Small 2001, ch. 8). On the one hand, analog computers afforded a means of understanding analytically intractable systems by means of "dynamic models." By setting up an analog simulation, it was possible to vary the parameters and explore interactively the behavior of a dynamical system that could not be analyzed mathematically. Digital simulations, in contrast, were orders of magnitude slower and did not permit this kind of interactive investigation. (Performance has improved sufficiently in contemporary digital computers so that in many cases digital simulations can be used as dynamic models, sometimes with an interface that mimics an analog computer; see Bissell 2004.)

Analog computing is also relevant to the cognitive distinction between *knowing how* (*procedural knowledge*) and *knowing that* (*declarative knowledge*) (Small, 2001, ch. 8). The latter ("know-that") is more characteristic of scientific culture, which strives for generality and exactness, often by designing experiments that allow phenomena to be studied in isolation, whereas the former ("know-how") is more characteristic of engineering culture; at least it was so through the first half of the twentieth century, before the development of "engineering science" and the widespread use of analytic techniques in engineering education and practice. Engineers were faced with analytically intractable systems, with inexact measurements, and with empirical relationships (characteristic curves, etc.), all of which made analog computers attractive for solving engineering problems. Furthermore, because analog computing made use of physical phenomena that were mathematically analogous to those in the primary system, the engineer's intuition and understanding of one system could be transferred to the other. Some commentators have mourned the loss of hands-on intuitive understanding resulting from the increasingly scientific orientation of engineering education and the disappearance of analog computers (Bissell, 2004; Lang, 2000; Owens, 1986; Puchta, 1996).

I will mention one last cognitive issue relevant to the differences between analog and digital computing. As already discussed Sec. C.4, it is generally agreed that it is less expensive to achieve high precision with digital technology than with analog technology. Of course, high precision may not be important, for example when the available data are inexact or in natural computation. Further, some advocates of analog computing argue that high precision digital results are often misleading (Small, 2001, p. 261). Precision does not imply accuracy, and the fact that an answer is displayed with 10 digits does not guarantee that it is accurate to 10 digits; in particular, engineering data may be known to only a few significant figures, and the accuracy of digital calculation may be limited by numerical problems. Therefore, on the one hand, users of digital computers might fall into the trap of trusting their apparently exact results, but users of modest-precision analog computers were more inclined to healthy skepticism about their computations. Or so it was claimed.

H Future directions

H.1 Post-Moore's Law computing

Certainly there are many purposes that are best served by digital technology; indeed there is a tendency nowadays to think that everything is done better digitally. Therefore it will be worthwhile to consider whether analog computation should have a role in future computing technologies. I will argue that the approaching end of *Moore's Law* (Moore, 1965), which has predicted exponential growth in digital logic densities, will encourage the development of new analog computing technologies.

Two avenues present themselves as ways toward greater computing power: faster individual computing elements and greater densities of computing elements. Greater density increases power by facilitating parallel computing, and by enabling greater computing power to be put into smaller packages. Other things being equal, the fewer the layers of implementation between the computational operations and the physical processes that realize them, that is to say, the more directly the physical processes implement the computations, the more quickly they will be able to proceed. Since most physical processes are continuous (defined by differential equations), analog computation is generally faster than digital. For example, we may compare analog addition, implemented directly by the additive combination of physical quantities, with the sequential process of digital addition. Similarly, other things being equal, the fewer physical devices required to implement a computational element, the greater will be the density of these elements. Therefore, in general, the closer the computational process is to the physical processes that realize it, the fewer devices will be required, and so the continuity of physical law suggests that analog computation has the potential for greater density than digital. For example, four transistors can realize analog addition, whereas many more are required for digital addition. Both considerations argue for an increasing role of analog computation in post-Moore's Law computing.

From this broad perspective, there are many physical phenomena that are potentially usable for future analog computing technologies. We seek phenomena that can be described by well-known and useful mathematical functions (e.g., addition, multiplication, exponential, logarithm, convolution). These descriptions do not need to be exact for the phenomena to be useful in many applications, for which limited range and precision are adequate. Furthermore, in some applications speed is not an important criterion; for example, in some control applications, small size, low power, robustness, etc. may be more important than speed, so long as the computer responds quickly enough to accomplish the control task. Of course there are many other considerations in determining whether given physical phenomena can be used for practical analog computation in a given application (MacLennan, 2009b). These include stability, controllability, manufacturability, and the ease of interfacing with input and output transducers and other devices. Nevertheless, in the post-Moore's Law world, we will have to be willing to consider all physical phenomena as potential computing technologies, and in many cases we will find that analog computing is the most effective way to utilize them.

Natural computation provides many examples of effective analog computation realized by relatively slow, low-precision operations, often through massive parallelism. Therefore, post-Moore's Law computing has much to learn from the natural world.

Bibliography

Adamatzky, A. (2001). *Computing in Nonlinear Media and Automata Collectives*. Bristol: Institute of Physics Publishing.

Adamatzky, A., De Lacy Costello, B., & Asai, T. (2005). *Reaction-Diffusion Computers*. Amsterdam: Elsevier.

Adleman, L. M. (1995). On constructing a molecular computer. In R. J. Lipton & E. B. Baum (Eds.), *DNA Based Computers*, Proc. of a DIMACS Workshop (pp. 1–22).

Anderson, J. (1995). *An Introduction to Neural Networks*. Cambridge, MA: MIT Press.

Ashley, J. R. (1963). *Introduction to Analog Computing*. New York: John Wiley & Sons.

Aspray, W. (1993). Edwin l. harder and the anacom: Analog computing at westinghouse. *IEEE Annals of the History of Computing*, 15(2), 35–52.

Ben-Hur, A., Siegelmann, H., & Fishman, S. (2002). A theory of complexity for continuous time systems. *Journal of Complexity*, 18, 51–86.

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5), 563–591.

Benioff, P. (1982). Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics*, 29(3), 515–546.

Bennett, C. H. (1973). Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6), 525–532.

Bennett, C. H. (1982). The thermodynamics of computation — a review. *Int. J. Theo. Phys.*, 21(12), 905–940.

Bennett, C. H. (2003). Notes on Landauer's principle, reversible computation, and Maxwell's Demon. *Studies in History and Philosophy of Modern Physics*, 34, 501–510.

Berut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., & Lutz, E. (2012). Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483, 187–189.

Bissell, C. C. (2004). A great disappearing act: The electronic analogue computer. In *IEEE Conference on the History of Electronics* Bletchley, UK.

Blum, L., Cucker, F., Shub, M., & Smale, S. (1998). *Complexity and Real Computation*. Berlin: Springer-Verlag.

Blum, L., Shub, M., & Smale, S. (1988). On a theory of computation and complexity over the real numbers: Np completeness, recursive functions and universal machines. *The Bulletin of the American Mathematical Society*, 21, 1–46.

Bournez, O., Campagnolo, M., Graça, D., & Hainry, E. (2006). The General Purpose Analog Computer and computable analysis are two equivalent paradigms of analog computation. In *Theory and Applications of Models of Computation (TAMC 2006)*, volume 3959 of *Lectures Notes in Computer Science* (pp. 631–43). Berlin: Springer-Verlag.

Bournez, O. & Cosnard, M. (1996). On the computational power of dynamical systems and hybrid systems. *Theoretical Computer Science*, 168(2), 417–59.

Bowles, M. D. (1996). U.S. technological enthusiasm and British technological skepticism in the age of the analog brain. *Annals of the History of Computing*, 18(4), 5–15.

Branicky, M. (1994). Analog computation with continuous ODEs. In *Proceedings IEEE Workshop on Physics and Computation* (pp. 265–74). Dallas, TX.

Brockett, R. (1988). Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In *Proc. 27th IEEE Conf. Decision and Control* (pp. 799–803). Austin, TX.

Calude, C., Casti, J., & Dinneen, M., Eds. (1998). *Unconventional Models of Computation*. Singapore & New York: Springer.

Calude, C. & Paun, G. (2001). *Computing with Cells and Atoms*. London & New York: Taylor & Francis.

Camazine, S., Deneubourg, J., Franks, N. R., Sneyd, J. Theraulaz, G., & Bonabeau, E. (2001). *Self-organization in Biological Systems*. Princeton.

Changeux, J. (1985). *Neuronal Man: The Biology of Mind*. Oxford: Oxford University Press. tr. by L. Garey.

Clymer, A. B. (1993). The mechanical analog computers of hannibal ford and william newell. *IEEE Annals of the History of Computing*, 15(2), 19–34.

Daugman, J. (1993). An information-theoretic view of analog representation in striate cortex. In E. Schwartz (Ed.), *Computational Neuroscience* (pp. 403–423). Cambridge: MIT Press.

Davidson, E. H. (2006). *The Regulatory Genome: Gene Regulatory Networks in Development and Evolution*. Amsterdam: Academic Press.

Davies, J. A. (2005). *Mechanisms of Morphogensis*. Amsterdam: Elsevier.

Davis, M. (2004). The myth of hypercomputation. In C. Teuscher (Ed.), *Alan Turing: Life and Legacy of a Great Thinker* (pp. 195–212). Berlin: Springer-Verlag.

Davis, M. (2006). Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation*, 178, 4–7.

Dennard, R. H., Gaensslen, F., Yu, H.-N., Rideout, L., Bassous, E., & LeBlanc, A. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5), 256–268. Reprinted, Proc. IEEE, vol. 87, no. 4, APRIL 1999, pp. 668–678.

Deutsch, D. (1985). Quantum theory, the Church-Turing principle, and the universal quantum computer. *Proceedings of the Royal Society London A*, 400, 97–119.

Eberbach, E., Goldin, D., & Wegner, P. (2003). Turing's ideas and models of computation. In C. Teuscher (Ed.), *Alan Turing: Life and Legacy of a Great Thinker*. Berlin, Heidelberg & New York: Springer-Verlag.

Fakhraie, S. M. & Smith, K. C. (1997). *VLSI-Compatible Implementation for Artificial Neural Networks*. Boston: Kluwer Academic Publishers.

Feynman, R. (1986). Quantum mechanical computers. *Foundations of Physics*, 16(6), 507–531.

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, 21, 467–488.

Frank, M. P. (2005a). The indefinite logarithm, logarithmic units, and the nature of entropy. Dept. of Electrical & Computer Engineering, FAMU-FSU College of Engineering.

Frank, M. P. (2005b). Introduction to reversible computing: Motivation, progress, and challenges. In *CF '05, May 4–6, 2005, Ischia, Italy.*

Franklin, S. & Garzon, M. (1990). Neural computability. In O. M. Omidvar (Ed.), *Progress in Neural Networks*, volume 1 (pp. 127–145). Norwood, NJ: Ablex.

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

Freeth, T., Bitsakis, Y., Moussas, X., Seiradakis, J., Tselikas, A., Mangou, H., Zafeiropoulou, M., Hadland, R., Bate, D., Ramsey, A., Allen, M., Crawley, A., Hockley, P., Malzbender, T., Gelb, D., Ambrisco, W., & Edmunds, M. (2006). Decoding the ancient Greek astronomical calculator known as the Antikythera mechanism. *Nature*, 444, 587–91.

Garzon, M. & Franklin, S. (1989). Neural computability ii (extended abstract). In *Proceedings, IJCNN International Joint Conference on Neural Networks*, volume 1 (pp. 631–637). New York, NJ: Institute of Electrical and Electronic Engineers.

Garzon, M. & Franklin, S. (1990). Computation on graphs. In O. M. Omidvar (Ed.), *Progress in Neural Networks*, volume 2 chapter 13. Norwood, NJ: Ablex.

George, S., Kim, S., Shah, S., Hasler, J., Collins, M., Adil, F., Wunderlich, R., Nease, S., & Ramakrishnan, S. (2016). A programmable and configurable mixed-mode FPAA SoC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6), 2253–2261.

Goldstine, H. (1972). *The Computer from Pascal to von Neumann.* Princeton, NJ: Princeton.

Grossberg, S. (1967). Nonlinear difference-differential equations in prediction and learning theory. *Proceedings of the National Academy of Sciences, USA*, 58(4), 1329–1334.

Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, LII, 213–257.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23, 121–134.

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42, 335–346.

Harnad, S. (1993). Grounding symbols in the analog world. *Think*, 2, 12–78.

Hartl, D. L. (1994). *Genetics.* Boston: Jones & Bartlett, 3rd edition.

Haykin, S. (2008). *Neural Networks and Learning Machines.* New York: Pearson Education, third edition.

Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, 81, 3088–92.

Howe, R. M. (1961). *Design Fundamentals of Analog Computer Components.* Princeton, NJ: Van Nostrand.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5, 90–9.

Kirchhoff, G. (1845). Ueber den durchgang eines elektrischen stromes durch eine ebene, insbesondere durch eine kreisförmige. *Annalen der Physik und Chemie*, 140/64(4), 497–514.

Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3), 183–191. Reprinted, Vol. 44 No. 1/2, Jan./March 2000, pp. 261–269.

Lang, G. F. (2000). *Analog* was not a computer trademark! Why would anyone write about analog computers in year 2000? *Sound and Vibration*, (pp. 16–24).

Leff, H. S. & Rex, A. F. (1990). *Maxwell's Demon: Entropy, Information, Computing*. Princeton, NJ: Princeton University Press.

Leff, H. S. & Rex, A. F. (2003). *Maxwell's Demon 2: Entropy, Classical and Quantum Information, Computing*. Bristol and Philadelphia: Institute of Physics Publishing.

Lipka, J. (1918). *Graphical and Mechanical Computation*. New York: Wiley.

Lipshitz, L. & Rubel, L. A. (1987). A differentially algebraic replacment theorem. *Proceedings of the American Mathematical Society*, 99(2), 367–72.

Maass, W. & Sontag, E. (1999a). Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. *Neural Computation*, 11, 771–782.

Maass, W. & Sontag, E. (1999b). Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. *Neural Computation*, 11(3), 771–782.

MacLennan, B. J. (1987). Technology-independent design of neurocomputers: The universal field computer. In M. Caudill & C. Butler (Eds.), *Proceedings of the IEEE First International Conference on Neural Networks*, volume 3 (pp. 39–49).: IEEE Press.

MacLennan, B. J. (1990). *Field Computation: A Theoretical Framework for Massively Parallel Analog Computation, Parts I–IV*. Technical Report CS-90-100, Department of Computer Science, University of Tennessee, Knoxville. Also available from `web.eecs.utk.edu/~mclennan`.

MacLennan, B. J. (1991). *Gabor Representations of Spatiotemporal Visual Images*. Technical Report CS-91-144, Department of Computer Science, University of Tennessee, Knoxville. Also available from `web.eecs.utk.edu/~mclennan`.

MacLennan, B. J. (1993). Grounding analog computers. *Think*, 2, 48–51. Also available from `web.eecs.utk.edu/~mclennan` and at `cogprints.org/542/`.

MacLennan, B. J. (1994a). Continuous computation and the emergence of the discrete. In K. H. Pribram (Ed.), *Origins: Brain & Self-Organization* (pp. 121–151). Hillsdale, NJ: Lawrence Erlbaum. Also available from `web.eecs.utk.edu/~mclennan` and at `cogprints.org/540/`.

MacLennan, B. J. (1994b). Continuous symbol systems: The logic of connectionism. In D. S. Levine & M. Aparicio IV (Eds.), *Neural Networks for Knowledge Representation and Inference* (pp. 83–120). Hillsdale, NJ: Lawrence Erlbaum. Also available from `web.eecs.utk.edu/~mclennan`.

MacLennan, B. J. (1994c). "Words lie in our way". *Minds and Machines*, 4(4), 421–437. Also available from `web.eecs.utk.edu/~mclennan` and at `cogprints.org/383/`.

MacLennan, B. J. (1995). Continuous formal systems: A unifying model in language and cognition. In *Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems* (pp. 161–172). Monterey, CA. Also available from `web.eecs.utk.edu/~mclennan` and at `cogprints.org/541`.

MacLennan, B. J. (1999). Field computation in natural and artificial intelligence. *Information Sciences*, 119, 73–89. Also available from `web.eecs.utk.edu/~mclennan`.

MacLennan, B. J. (2001). *Can Differential Equations Compute?* Technical Report UT-CS-01-459, Department of Computer Science, University of Tennessee, Knoxville. Also available from `web.eecs.utk.edu/~mclennan`.

MacLennan, B. J. (2003). Transcending Turing computability. *Minds and Machines*, 13, 3–22.

MacLennan, B. J. (2004). Natural computation and non-Turing models of computation. *Theoretical Computer Science*, 317, 115–145.

MacLennan, B. J. (2008). *Aspects of Embodied Computation: Toward a Reunification of the Physical and the Formal*. Technical Report UT-CS-08-610, Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2009a). Field computation in natural and artificial intelligence. In R. Meyers et al. (Ed.), *Encyclopedia of Complexity and System Science* chapter 6, entry 199, (pp. 3334–3360). Springer.

MacLennan, B. J. (2009b). Super-Turing or non-Turing? Extending the concept of computation. *International Journal of Unconventional Computing*, 5(3–4), 369–387.

MacLennan, B. J. (2010). The U-machine: A model of generalized computation. *International Journal of Unconventional Computing*, 6(3–4), 265–283.

MacLennan, B. J. (2013). Cognition in Hilbert space. *Behavioral and Brain Sciences*, 36(3), 296–7.

Maini, P. K. & Othmer, H. G., Eds. (2001). *Mathematical Models for Biological Pattern Formation*. Springer- Verlag.

Markov, A. (1961). *Theory of Algorithms*. Jerusalem, Israel: Israel Program for Scientific Translation. US Dept. of Commerce Office of Technical Service OTS 60-51085, transl. by Jacques J. Schorr-Kon & PST Staff. Translation of *Teoriya Algorifmov*, Academy of Sciences of the USSR, Moscow, 1954.

Maziarz, E. & Greenwood, T. (1968). *Greek Mathematical Philosophy*. New York: Frederick Ungar.

McClelland, J., Rumelhart, D., & the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*. Cambridge, MA: MIT Press.

Mead, C. (1987). Silicon models of neural computation. In M. Caudill & C. Butler (Eds.), *Proceedings, IEEE First International Conference on Neural Networks*, volume I (pp. 91–106). Piscataway NJ: IEEE Press.

Mead, C. (1989). *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley.

Mermin, N. D. (2007). *Quantum Computer Science: An Introduction*. Cambridge: Cambridge University Press.

Mills, J. W. (1996). The continuous retina: Image processing with a single-sensor artificial neural field network. In *Proceedings IEEE Conference on Neural Networks*: IEEE Press.

Mills, J. W., Himebaugh, B., Kopecky, B., Parker, M., Shue, C., & Weilemann, C. (2006). "Empty space" computes: The evolution of an unconventional supercomputer. In *Proceedings of the 3rd Conference on Computing Frontiers* (pp. 115–26). New York: ACM Press.

Milner, R. (1993). Elements of interaction. *Communications of the ACM*, 36(1), 78–89.

Milner, R., Parrow, J., & Walker, D. (1992). A calculus of mobile processes, I & II. *Information and Computation*, 100, 1–77.

Moore, C. (1996). Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162, 23–44.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8), 114–117.

Murray, J. D. (1977). *Lectures on Nonlinear Differential-Equation Models in Biology*. Oxford: Oxford.

Nielsen, M. A. & Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Cambridge, 10th anniversary edition edition.

Omohundro, S. (1984). Modeling cellular automata with partial differential equations. *Physica D*, 10, 128–34.

Orponen, P. (1997). A survey of continous-time computation theory. In *Advances in Algorithms, Languages, and Complexity* (pp. 209–224).

Orponen, P. & Matamala, M. (1996). Universal computation by finite two-dimensional coupled map lattices. In *Proceedings, Physics and Computation 1996* (pp. 243–7). Cambridge, MA: New England Complex Systems Institute.

Owens, L. (1986). Vannevar bush and the differential analyzer: The text and context of an early computer. *Technology and Culture*, 27(1), 63–95.

Peterson, G. R. (1967). *Basic Analog Computation*. New York: Macmillan.

Pothos, E. M. & Busemeyer, J. R. (2013). Can quantum probability provide a new direction for cognitive modeling? *Behavioral and Brain Sciences*, 36, 255–327.

Pour-El, M. (1974a). Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199, 1–29.

Pour-El, M. & Richards, I. (1979). A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17, 61–90.

Pour-El, M. & Richards, I. (1981). The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics*, 39, 215–239.

Pour-El, M. & Richards, I. (1982). Noncomputability in models of physical phenomena. *International Journal of Theoretical Physics*, 21, 553–555.

Pour-El, M. B. (1974b). Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society*, 199, 1–29.

Puchta, S. (1996). On the role of mathematics and mathematical knowledge in the invention of vannevar bush's early analog computers. *IEEE Annals of the History of Computing*, 18(4), 49–59.

Reiner, J. M. (1968). *The Organism as an Adaptive Control System*. Englewood Cliffs: Prentice-Hall.

Rieffel, E. & Polak, W. (2000). An introduction to quantum computing for non-physicists.

Rimon, E. & Koditschek, D. (1989). The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation, Scottsdale AZ* (pp. 21–6). New York: IEEE Press.

Rogers, A. & Connolly, T. (1960). *Analog Computation in Engineering DESIGN*. New York: McGraw-Hill.

Rubel, L. A. (1985). The brain as an analog computer. *Journal of Theoretical Neurobiology*, 4, 73–81.

Rubel, L. A. (1988). Some mathematical limitations of the general-purpose analog computer. *Advances in Applied Mathematics*, 9, 22–34.

Rubel, L. A. (1993). The extended analog computer. *Advances in Applied Mathematics*, 14, 39–50.

Rumelhart, D., McClelland, J., & the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA: MIT Press.

Sanger, T. (1996). Probability density estimation for the interpretation of neural population codes. *Journal of Neurophysiology*, 76, 2790–3.

Shannon, C. E. (1941). Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics of the Massachusetts Institute Technology*, 20, 337–354.

Shannon, C. E. (1993). Mathematical theory of the differential analyzer. In N. J. A. Sloane & A. D. Wyner (Eds.), *Claude Elwood Shannon: Collected Papers* (pp. 496–513). New York: IEEE Press.

Siegelmann, H., Ben-Hur, A., & Fishman, S. (1999). Computational complexity for continuous time dynamics. *Physical Review Letters*, 83(7), 1463–6.

Siegelmann, H. & Sontag, E. (1994a). Analog computation via neural networks. *Theoretical Computer Science*, 131, 331–360.

Siegelmann, H. & Sontag, E. (1994b). Analog computation via neural networks. *Theoretical Computer Science*, 131, 331–360.

Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Boston: Birkhäuser.

Small, J. S. (1993). General-purpose electronic analog computing. *IEEE Annals of the History of Computing*, 15(2), 8–18.

Small, J. S. (2001). *The Analogue Alternative*. London & New York: Routledge.

Sprecher, D. (1965). On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115, 340–355.

Stannett, M. (1990). X-machines and the halting problem: Building a super-Turing machine. *Formal Aspects of Computing*, 2, 331–341.

Stepney, S. (2004). Journeys in non-classical computation. In T. Hoare & R. Milner (Eds.), *Grand Challenges in Computing Research* (pp. 29–32). Swindon: BCS.

Thomson, W. (1876). Mechanical integration of the general linear differential equation of any order with variable coefficients. *Proceedings of the Royal Society*, 24, 271–275.

Thomson, W. (1878). Harmonic analyzer. *Proceedings of the Royal Society*, 27, 371–373.

Thomson, W. (1938). The tides. In *The Harvard Classics*, volume 30: Scientific Papers (Lord Kelvin) (pp. 274–307). New York: Collier.

Tong, D. (Lent Term, 2011 and 2012). *Statistical Physics: University of Cambridge Part II Mathematical Tripos.*

Truitt, T. D. & Rogers, A. E. (1960). *Basics of Analog Computers*. New York: John F. Rider.

van Gelder, T. (1997). Dynamics and cognition. In J. Haugeland (Ed.), *Mind Design II: Philosophy, Psychology and Artificial Intelligence* chapter 16, (pp. 421–450). Cambridge MA: MIT Press, revised & enlarged edition.

Wegner, P. (1997). Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5), 81–91.

Wegner, P. (1998). Interactive foundations of computing. *Theoretical Computer Science*, 192(2), 315–351.

Wegner, P. & Goldin, D. (2003). Computation beyond Turing machines: Seeking appropriate methods to model computing and human thought. *Communications of the ACM*, 46(4), 100–102.

Weyrick, R. C. (1969). *Fundamentals of Analog Computers*. Englewood Cliffs: Prentice-Hall.

Winfree, E. (1998). *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology.

Wolpert, D. (1991). *A Computationally Universal Field Computer which is Purely Linear*. Technical Report LA-UR-91-2937, Los Alamos National Laboratory.

Wolpert, D. H. & MacLennan, B. J. (1993). *A Computationally Universal Field Computer that is Purely Linear*. Technical Report CS-93-206, Dept. of Computer Science, University of Tennessee, Knoxville. Also available from `web.eecs.utk.edu/~mclennan`.