COMPUTATIONS AND COMPUTERS IN THE SCIENCES OF MIND AND BRAIN

by

Gualtiero Piccinini

BA, Università di Torino, 1994

Submitted to the Graduate Faculty of

University of Pittsburgh in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2003

UNIVERSITY OF PITTSBURGH

FACULTY OF ARTS AND SCIENCES

This dissertation was presented

by

Gualtiero Piccinini

It was defended on

June 20th, 2003

and approved by

John Earman, University Professor of History and Philosophy of Science

G. Bard Ermentrout, Professor of Mathematics

Paul Griffiths, Professor of History and Philosophy of Science

John D. Norton, Professor of History and Philosophy of Science

Peter K. Machamer, Professor of History and Philosophy of Science
Dissertation Director

To my parents and sisters

Section 3.1 of the present work is an adapted version of section 2 of Gualtiero Piccinini, "Alan Turing and the Mathematical Objection," *Minds and Machines* **13**(1), pp. 23-48. Copyright 2003 by Kluwer, reproduced by permission.

The following archives have given permission to use extended quotations from unpublished work. From the Warren S. McCulloch Papers, American Philosophical Society Library. Copyright by the American Philosophical Society Library. From the Norbert Wiener Papers, Institute Archives and Special Collections, MIT Libraries. Copyright by the MIT Libraries.

COMPUTATIONS AND COMPUTERS IN THE SCIENCES OF MIND AND BRAIN

Gualtiero Piccinini, PhD

University of Pittsburgh, 2003

Computationalism says that brains are computing mechanisms, that is, mechanisms that perform computations. At present, there is no consensus on how to formulate computationalism precisely or adjudicate the dispute between computationalism and its foes, or between different versions of computationalism. An important reason for the current impasse is the lack of a satisfactory philosophical account of computing mechanisms. The main goal of this dissertation is to offer such an account.

I also believe that the history of computationalism sheds light on the current debate. By tracing different versions of computationalism to their common historical origin, we can see how the current divisions originated and understand their motivation. Reconstructing debates over computationalism in the context of their own intellectual history can contribute to philosophical progress on the relation between brains and computing mechanisms and help determine how brains and computing mechanisms are alike, and how they differ. Accordingly, my dissertation is divided into a historical part, which traces the early history of computationalism up to 1946, and a philosophical part, which offers an account of computing mechanisms.

The two main ideas developed in this dissertation are that (1) computational states are to be identified functionally not semantically, and (2) computing mechanisms are to be studied by functional analysis. The resulting account of computing mechanism, which I call the *functional account of computing mechanisms*, can be used to identify computing mechanisms and the functions they compute. I use the functional account of computing mechanisms to taxonomize computing mechanisms based on their different computing power, and I use this taxonomy of computing mechanisms to taxonomize different versions of computationalism based on the functional properties that they ascribe to brains. By doing so, I begin to tease out empirically testable statements about the functional organization of the brain

that different versions of computationalism are committed to. I submit that when computationalism is reformulated in the more explicit and precise way I propose, the disputes about computationalism can be adjudicated on the grounds of empirical evidence from neuroscience.

PREFACE

In the 1940s, inspired by the birth and development of modern computers, Alan Turing, Warren McCulloch, Norbert Wiener, John von Neumann, and many others developed a new theory of the brain, here called *computationalism*. Computationalism says that brains are computing mechanisms, that is, mechanisms that perform computations. Computationalism expands the old idea that reasoning is a form of computation (from Hobbes to formal logic) into the stronger idea that all cognitive processes or even all neural processes are a form of computation. In the past fifty years, computationalism has shaped several fields. There are canonical explications of computationalism in computer science (Newell and Simon 1976), psychology (Pylyshyn 1984, Rumelhart and McClelland 1986), neuroscience (Churchland, Koch, and Sejnowski 1990), and philosophy (Fodor 1975).

Computationalists agree that brains are computing mechanisms, but in calling them computing mechanisms, they mean such radically different things that they often talk across each other. For some, the brain is functionally organized like the hardware of a desktop computer, on which different programs can run (e.g., Newell and Simon, Pylyshyn, and Fodor). For others, the brain is a set of networks of neurons, each of which computes its own function (e.g., Rumelhart and McClelland). Still others think that computations take place in the dendrites of single neurons (e.g., Koch 1999). Some investigators build computer simulations of cognitive phenomena studied by psychologists and argue that the neurological details are irrelevant to understanding the computational organization of the brain (e.g., Newell 1990, Fodor and Pylyshyn 1988). Other investigators model neurological phenomena described by neurophysiologists and maintain that on the contrary, it is simulating cognitive phenomena that has nothing to do with the computational organization of the brain (e.g., Koch 1999, Dayan and Abbott 2001).

Philosophers interested in the sciences of mind and brain have generally divided into computationalists and anti-computationalists. The former believe computationalism to be the best scientific theory of the brain, or even the only genuinely scientific theory of the brain. They have offered explications of computationalism and defended them on the grounds that they solve, or contribute to

solve, important philosophical problems. Anti-computationalists often believe that computationalism is absurd or false on *a priori* grounds, and have offered a number of objections to it (e.g., Searle 1980, Penrose 1994). But anti-computationalists fiercely disagree on what is misguided about computationalism.

At present, there is no consensus on how to formulate computationalism precisely or adjudicate the dispute between computationalism and its foes, or between different versions of computationalism. An important reason for the current impasse is the lack of a satisfactory philosophical account of computing mechanisms. The main goal of this dissertation is to offer such an account. I also believe that the history of computationalism sheds light on the current debate. By tracing different versions of computationalism to their common historical origin, we can see how the current divisions originated and understand their motivation. Reconstructing debates over computationalism in the context of their own intellectual history can contribute to philosophical progress on the relation between brains and computing mechanisms and help determine how brains and computing mechanisms are alike, and how they differ. Accordingly, my dissertation is divided into a historical part, which traces the early history of computationalism, and a philosophical part, which offers an account of computing mechanisms.

A good account of computing mechanisms can be used to express more explicitly and precisely the content of different versions of computationalism, thereby allowing a more rigorous assessment of the evidence for or against different versions of computationalism. Besides grounding discussions of computationalism, there is independent motivation for an account of computing mechanisms. Given the importance of computers within contemporary society, philosophical attention has been directed at them in recent years (Floridi 1999). An account of computing mechanisms contributes to the emerging field of philosophy of computation.

The first step towards an account of computing mechanisms is to put on the table the relevant notion of computation. There is consensus that the notion of computation that is relevant to computationalism, as well as to modern computer science and technology, is the one analyzed by Alan Turing in terms of Turing Machines. Turing Machines are a mathematical formalism for manipulating

symbols in accordance with fixed instructions so as to generate certain output strings of symbols from input strings of symbols. Turing's work on Turing Machines, together with work by other authors in the 1930s on the same notion of computability, led to the development of the classical mathematical theory of computability. The notion of Turing Machine, together with some important results of computability theory, is briefly reviewed in Chapter 1.

By building on Turing's notion of computability, Warren McCulloch and others developed computationalism in the 1940s. Chapters 2 through 6 describe how this happened. Chapter 2 describes the pre-Turing background to McCulloch's work. Chapter 3 describes McCulloch's efforts to formulate a mechanistic theory of mind and the impact of Turing's work on his effort. Chapter 4 is a detailed analysis of the first—and in my view the most influential—formulation of contemporary computationalism, McCulloch and Pitts's "A Logical Calculus of the Ideas Immanent in Nervous Activity." Chapter 5 and 6 describe early discussions (1943-1946) between McCulloch and others pertaining to brains, computers, and their mutual relations. This historical background paves the way for the philosophical part of this dissertation, which is contained in the remaining four chapters.

The foundation of the classical theory of computability is the thesis that any function that is effectively calculable (in an intuitive sense) is computable by some Turing Machine. This thesis, called *Church-Turing thesis* (CT), has given rise to three relevant streams of philosophical literature. First, some philosophers of mind have assumed CT to be true and used it as *a priori* evidence for computationalism. Second, some logicians and philosophers of mathematics have debated whether CT is true and what its proper scope is. And third, some physicists and philosophers of physics have debated whether CT applies to the physical world. These three streams of literature have proceeded largely independently of one another, with regrettable effects. In particular, arguments from CT to computationalism have not taken into account the best scholarship on CT's proper scope and the way CT applies to the physical world.

To remedy this situation, Chapter 7 begins to bring these three streams of literature together. It clarifies the proper scope of CT, the sense in which CT applies to the physical world, and proceeds to

assess arguments from CT to computationalism. It concludes that all such arguments are unsound, because CT—when properly understood—does not establish whether cognitive or neural processes (or any other processes) are computations.

After this assessment of CT's relevance to computationalism, the road is clear to discuss two philosophical topics that have heavily affected both discussions of computationalism and the way computing mechanisms are understood in the philosophical literature. These topics are the mind-body problem and the problem of giving a naturalistic explanation of intentionality. The next two chapters are devoted to how computationalism and computing mechanisms relate to these topics.

First, I discuss the relevance of the mind-body problem to computationalism. *Computational functionalism* is the thesis that the mind is the software of the brain, which entails that the brain is a computing mechanism (for running mental programs). Computational functionalism was proposed as a solution to the mind-body problem (Putnam 1967b), and became very influential. As a result, many philosophers became convinced that computationalism is a consequence of this popular solution to the mind-body problem.

In Chapter 8, I argue that this is not the case. First, I employ the language of functional analysis to explicate the idea that the mind is a program running on the brain. Then I distinguish functionalism, namely the thesis that the mind is the functional organization of the brain given by some functional analysis of the brain (which may or may not ascribe computations to it), from the stronger computational functionalism, namely the thesis that the functional organization of the brain is given by a (mental) computer program. I argue that none of the arguments given for functionalist (including computational functionalist) solutions to the mind-body problem offer any support for the conclusion that the mind is a program running on the brain. Accordingly, I argue that this latter view should be considered as an empirical hypothesis about the kind of functional analysis that applies to minds and brains. As a consequence, computational functionalism should be considered as the conjunction of a functionalist solution to the mind-body problem and the empirical hypothesis that the brain is functionally organized in accordance with a mental program.

After the mind-body problem, I discuss the relevance to computationalism of the naturalistic explanation of intentionality, i.e. the problem of finding a naturalistic explanation for the content of mental states. The *semantic view of computational states* is the view that inputs, internal states, and outputs of computing mechanisms have their content essentially, i.e., computational inputs, outputs, and internal states can be identified only by reference to their semantic properties. Almost all computationalist philosophers believe the semantic view of computational states. If the semantic view of computational states is correct, it gives some hope for a naturalistic explanation of intentionality. For on the one hand, no one doubts that computing mechanisms are natural objects that can be given naturalistic explanations. On the other hand, the semantic view of computational states entails that computationalism ascribes to the brain states that are essentially endowed with content. If the essentially contentful states ascribed to the brain by computationalism coincide with the mental states whose intentionality many philosophers would like to explain naturalistically, then computationalism offers the basis for a naturalistic explanation of intentionality. This has acted as a powerful motivation for computationalism.

In Chapter 9, I reject the semantic view of computational states in favor of the view that computational inputs, outputs, and internal states are identified by their functional properties, as described by a specific kind of functional analysis. This view fits better than the semantic view with the practices of computability theorists and computer designers, but it undercuts one of the main traditional philosophical motivations for computationalism.

The two main ideas developed in Chapters 8 and 9 are that (1) computational states are to be identified functionally not semantically, and (2) computing mechanisms are to be studied by functional analysis. These ideas come to fruition in Chapter 10, where the relevant kind of functional analysis is spelled out. The resulting account of computing mechanism, which I call the *functional account of computing mechanisms*, can be used to identify computing mechanisms and the functions they compute. I use the functional account of computing mechanisms to taxonomize computing mechanisms based on their different computing power, and I use this taxonomy of computing mechanisms to taxonomize different versions of computationalism based on the functional properties that they ascribe to brains. By

doing so, I begin to tease out empirically testable statements about the functional organization of the brain that different versions of computationalism are committed to. I submit that when computationalism is reformulated in the more explicit and precise way I propose, the disputes about computationalism can be adjudicated on the grounds of empirical evidence from neuroscience.

ACKNOWLEDGEMENTS

My greatest debt is to Peter Machamer, my advisor. The ways in which he helped me are too numerous to enumerate. My committee members, John Earman, Bard Ermentrout, Paul Griffiths, and John Norton, gave me the right balance of constructive criticism and support. John Norton also guided me through my initial search for a dissertation project.

I wrote my first paper on this topic for a class I took with Ken Manders. He kindly advised me to pursue my investigations further. At the time, Carl Craver was writing his dissertation on mechanisms in neuroscience. I jokingly asked Carl to do a good job so I could use his conclusions in my research. I'm pleased to say that I was influenced by Carl's dissertation as well as his subsequent work (e.g., Machamer, Darden and Craver 2000, Craver 2001a). Another important early influence on my dissertation was Wilfried Sieg's work on the history and philosophy of computation (e.g., Sieg 1994). In the last decade, Sieg and Jack Copeland have done more than anyone before them to clarify the Church-Turing thesis and fight the misconceptions about the Church-Turing thesis that pervade the philosophy of mind literature. My work, especially in Chapter 7, builds on theirs. While working on Chapters 8 through 10, I gained the most insight into computational theories of mind and brain by reading the works of Jerry Fodor. If I have succeeded at all in moving the debate forward, I owe it to a large extent to how I responded to Fodor's writings.

A number of people have given me feedback on parts of my dissertation or related material: Bob Brandom, Jack Copeland, Carl Craver, Reinaldo Elugardo, Uljana Feest, Clark Glymour, Rick Grush, Graham Hubbs, Ken Manders, Diego Marconi, Jim Moor, Bob Olby, Anastasia Panagopoulos, Elizabeth Paris, Merrilee Salmon, Andrea Scarantino, Susan Schneider, Oron Shagrir, Wilfried Sieg, Susan Sterrett, Julie Yoo, and Julie Zahle. If I have forgotten anyone, I apologize.

Other people helped me by conversing or corresponding with me on topics related to my dissertation. They include Erik Angner, Robert S. Cox, John Haugeland, Lance Lugar, Valerie-Anne

Lutz, Jay McClelland, Wendy Parker, and Martha Pollack. Again, I apologize to anyone I inadvertently omitted.

As a graduate student unknown to them, I wrote or approached Brian Davies, Daniel Dennett, Jerry Fodor, Gilbert Harman, Hilary Putnam, Dave Touretzky, and Bernard Widrow concerning my research. They generously responded with helpful remarks for which I am grateful.

I have presented parts of my dissertation to various philosophical audiences. Thanks to those present for their attention and responses.

I'd like to thank all my philosophy teachers from high school to college to graduate school, as well as all others who taught me the little I know about the ways of philosophy and of life. Without their example and encouragement, I would not be where I am.

Finally, I am grateful to Becka Skloot for her help and support during all these years.

My research was supported in part by the National Science Foundation under Grant No. SES-0216981, by an Adelle and Erwin Tomash Fellowship, by an Andrew Mellon Predoctoral Fellowship, and by a Regione Sardegna Doctoral Scholarship. I am grateful to those institutions, the administrators who run them, the politicians who back them, and the taxpayers who ultimately fund them. Any opinions, findings, conclusions, and recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of these funding institutions.

TABLE OF CONTENTS

LIST OF FIGURES

1 COMPUTABILITY

1.1 Effective Calculability

This chapter introduces some fundamental notions related to computation, which will be used throughout the dissertation. This first section is devoted to the pre-theoretical notion of effective calculability, or computability by an effective procedure (computability for short). This informal notion motivates the formally defined notion of Turing-computability, which I introduce in the following section. In the last section, I briefly introduce the Church-Turing thesis, which says that the formal notion is an adequate formalization of the informal one.

During the first decades of the 20th century, mathematicians' interest in computable functions lay in the *foundations* of mathematics. Different philosophical approaches were proposed. L. E. J. Brouwer was the main supporter of *intuitionism*, according to which an existence proof for a mathematical object was admissible only if constructive (Brouwer 1975). David Hilbert proposed his *proof theory* to formalize in axiomatic fashion mathematical reasoning in an attempt to establish the foundations of mathematics without endorsing Brouwer's restrictions (Hilbert 1925, 1927, reprinted in van Heijenoort 1967; Hilbert and Ackermann 1928). This formalization allowed Hilbert to formulate rigorously the decision problem for first-order logic. A *decision problem* requests a method for answering a yes-or-no question concerning a domain of objects: "Given any sequence of symbols, is it a formula?" or "Given any formula, is it provable?" A solution to a decision problem is an effective procedure: a uniform method or algorithm specified by a finite set of instructions, by which any instance of the question can be answered in a finite number of steps. "Effective procedure" is a term used by some mathematicians in place of "algorithm"; an effective procedure cannot appeal to non-extensionally definable capacities like intuition, creativity, or guesswork, and it always generates the correct answer.[1]

[1] After the work of Turing and others, "effective procedure" was also used for procedures *not* guaranteed to generate all values of a total function, that is, they calculated only the values of a partial function (cf. Wang 1974, p. 84).

Lacking a rigorous definition of "effective procedure," mathematicians called it an "intuitive concept" to distinguish it from formally defined mathematical concepts.[2] Kurt Gödel proposed replacing "effective procedures" with a rigorously defined concept, that of "recursive functions," but he didn't rule out that some effective procedures might not be included within recursive functions (1931, 1934). Alonzo Church (1936) and Alan Turing (1936-7) strengthened Gödel's tentative identification of effective procedures and recursive functions to a general thesis, now called the Church-Turing thesis. Based on the Church-Turing thesis, Church and Turing proved that some functions are not computable. For example, Turing pointed out that he and Church used different definitions but reached "similar conclusions," i.e., that "the Hilbertian Entscheidungsproblem [i.e., the decision problem for first-order logic] can have no solution" (Turing 1936-7, 116, 117, 145).[3]

The notion of *effective procedure* can be informally defined as a procedure with the following properties[4]:

1) It uses a *finite* number of *primitive operations* for a *finite* number of times specified by a *finite* number of *deterministic instructions* (i.e. instructions whose execution yields a unique next step in the procedure).
2) Instructions are *non-ambiguous* and *finite* in length.
3) The procedure requires *no intuitions* about the subject matter (e.g. intuitions about properties of numbers), no ingenuity, no invention, no guesses.
4) For any argument of the function computed by the procedure, the procedure is the same (*uniformity*).
5) If the procedure terminates, for any argument of the function computed by the procedure, the procedure generates the *correct* value.

About the role of effective procedures in foundations of mathematics, especially Hilbert's approach, see Hallett 1994, Sieg 1994, Shapiro 1983, 1995. For a history of foundations of mathematics, see Webb 1980, Mancosu 1998.

[2] To refer to the intuitive notion of effective procedure, different authors used different terms. Instead of "procedure," some used "process," "method," or "rule." Instead of "effective," some used "finite," "finite combinatorial," "mechanical," "definite," "constructively defined," or "algorithmic." Some of the terms used as synomyms of "effectively calculable" are listed by Gödel 1965, 72; Kleene 1987a, pp. 55-56.

[3] On the origin of CT and recursive function theory, see Davis 1982; Gandy 1988; Kleene 1979, 1987a; Sieg 1994, 1997; Piccinini 2003a.

[4] The properties are somewhat redundant, but are kept separate for explicitness.

When we have a formalized language in which both a domain and operations over objects in that domain are formally defined, we can talk about lists of formalized instructions and call them *programs*. Programs are the formal replacement of algorithms or effective procedures. Because of this, programs are said to *implement* algorithms (or procedures).

Not all mathematical procedures are effective procedures or algorithms. It may be possible to specify sequences of operations that are not guaranteed to find all the values of a function for every argument. These procedures, called *heuristics*, generate a search for the desired value: the search may find the value of the function being computed or it may find an output that only approximates that value.

A caveat needs to be added about the relationship between programs (and procedures) and the functions they compute. A program is a definition of a function, from its inputs to its outputs; when the program doesn't halt, the function isn't defined. So, by definition any program computes the values of the function that *it* defines: it implements an algorithm or effective procedure relative to that function. However, typically a program is written to find values for a function that is defined independently of the program's existence. In such a case, the program may or may not be implementing an algorithm that finds the values of that function. Many programs do not always find the values of the independently defined function they are designed to compute, but rather they find approximations to those values. In such cases, relative to the independently defined functions, those programs are said to implement not algorithms but *heuristics*.[5]

---

[5] The philosophical literature is not always clear on this point. For example:

> The possibility of heuristic procedures on computers is sometimes confusing. In one sense, every digital computation (that does not consult a randomizer) is algorithmic; so how can any of them be heuristic? The answer is again a matter of perspective. Whether any given procedure is algorithmic or heuristic depends on how you describe the task (Haugeland 1997, p. 14).

But whether a procedure (or program) is algorithmic or heuristic does not depend on how one describes its task. Relative to its task, a procedure is algorithmic or heuristic depending on whether or not it is guaranteed to solve each instance of the task. Instead, a program is always algorithmic with respect to the generation of its outputs given its inputs.

Another example of confusion about this point is manifested by Dennett's statement (1975, p. 83) that human beings may not be Turing Machines (TMs), because humans may be implementing heuristics rather than algorithms. This presupposes that TMs implement only algorithms and not heuristics. Now, it is true that every TM implements an algorithm that generates its outputs given its inputs. But relative to the problem TMs are designed to solve, TMs—like any other computing mechanisms—may well be implementing heuristics.

1.2 Computability Theory

This section reviews some notions and results of classical computability theory. Computability theory studies what functions are computable, what mathematical properties they have, and the mathematical properties of computing mechanisms. Computable functions are identified with the class of recursive functions, inductively defined. As a specific example of computing mechanism, I will give Turing Machines. Using Turing Machines and recursive functions, I will introduce the notion of universal computing mechanism and the unsolvability of the halting problem.

1.2.1 Notation

$\{a_1, a_2, \ldots a_n\}$	Set of n objects $a_1, \ldots a_n$
$(a_1, a_2, \ldots a_n)$	List (or n-tuple) of n objects $a_1, \ldots a_n$
$a \in A$	a is an element of set A
N	Set of natural numbers 0, 1, 2, …
$f: A_1 \rightarrow A_2$	f is a function from A_1 to A_2
Domain of f	Set of all a such that $(a, b) \in f$ for some b
Range of f	Set of all of f(a) for a in the domain of f
Partial function on A	Function whose domain is a subset of A
Total function on A	Function whose domain is A
Alphabet	Nonempty set Σ of objects called symbols
Word or string on Σ	List of symbols on Σ, (instead of $(a_1, a_2, \ldots a_n)$, we write $a_1a_2\ldots a_n$)
$\|u\| = n$, where $u = a_1a_2\ldots a_n$	n is the length of u
a_1^n	concatenation of n symbols a_1
Σ*	Set of all words on alphabet Σ
Language on Σ	Any subset of Σ*
uv, where $u, v \in \Sigma^*$	Concatenation of u and v

Predicate on a set A	A total function P: A $\rightarrow$ **N** such that for each a $\in$ A, either P(a) = 1 or P(a) = 0, where 1 and 0 represent truth values
R = {a $\in$ A|P(a)}, P a predicate on A	R is the set of all a $\in$ A such that P(a) = 1; P is called the characteristic function of R
Pr (k)	k^{th} prime in order of magnitude
$\Psi_M^n(x_1, \ldots x_n)$	n-ary function computed by TM program M; when n = 1 we omit n

Computability theory applies to general word functions f: $\Sigma^* \rightarrow \Sigma'^*$, where Σ^* is the set of all words on alphabet Σ. Since words can be effectively encoded as natural numbers and *vice versa* (see section 1.2.4 below for an example of such an encoding), in this section we follow the standard convention of developing the theory with respect to number-theoretic functions f: **N** $\rightarrow$ **N**, without loss of generality.[6] Hence in this section, unless otherwise specified, "number" means natural number, and "function" means function on natural numbers. For the exposition of the material in this section, I drew mostly from Davis 1958 and Davis *et al*. 1994.

1.2.2 Recursive Functions

This section introduces the definition of the primitive recursive functions on the basis of three primitive base functions and two primitive operations. Then, by means of one further primitive operation, the class of partial recursive functions is defined.

The class of primitive recursive functions is defined inductively as follows.

Base functions:

Null function. n(x) = 0.

Successor function. s(x) = x + 1.

Projection functions. $u_i^n(x_1, \ldots x_n) = x_i$.

Operations:

[6] Computability theory can also be developed directly in terms of string functions (Machtey and Young 1978). This possibility will be significant in the Chapter on Computation and Content.

Composition. Let f be a function of k variables and let $g_1, \ldots g_k$ be functions of n variables. Let:

$h(x_1, \ldots x_n) = f(g_1(x_1, \ldots x_n), \ldots g_k(x_1, \ldots x_n))$.

Then h is obtained from f and $g_1, \ldots g_k$ by composition.

Primitive recursion. Let f be a function of n variables and let g be a function of n+2 variables. Let:

$h(x_1, \ldots x_n, 0) = f(x_1, \ldots x_n)$

$h(x_1, \ldots x_n, t+1) = g(t, h(x_1, \ldots x_n, t), x_1, \ldots x_n)$

Then h is obtained from f and g by primitive recursion.

Definition 1. A function f of n variables is *primitive recursive* if and only if it can be obtained from the base functions by finitely many operations of composition and primitive recursion.

Examples of primitive recursive functions include *addition*, *multiplication*, *exponentiation*, *predecessor*, and many other useful functions (see Davis *et al*. 1994, section 3.4).

In the present context, predicates are total functions whose values are 0 or 1 (representing true and false). Any primitive recursive function whose values are 0 and 1 is called a primitive recursive predicate. An example of a primitive recursive predicate is *equality*.

It can be easily shown, by induction on the definition of primitive recursive function, that every primitive recursive function is total.

Next, we introduce a further operation:

Minimalization (unbounded). Let P be a predicate of n+1 variables. We write $\min_y P(x_1, \ldots x_n, y)$ for the least value of y for which the predicate P is true if there is one. If there is no such value of y, then $\min_y P(x_1, \ldots x_n, y)$ is undefined.

Unbounded minimalization of a predicate can easily produce a function that is not total. An example is provided by *subtraction*:

$x - y = \min_z(y + z = x)$,

which is undefined for $x < y$.

Definition 2. A function f is *partial recursive* if and only if it can be obtained from the base functions by finitely many operations of composition, primitive recursion, and minimalization.

A partial recursive function that is total is called total recursive.

1.2.3 Turing Machines

Turing Machines (TMs) are perhaps the best-known computing mechanisms. They have two main components. First, there is a two-way potentially infinite tape divided into squares; each square contains one symbol (which may be an empty square). Second, there is an active device that can be in one of a finite number of states. The active device acts on the tape in one of four ways: it reads the symbol on a square, writes a symbol on a square, moves one square to the left, or moves one square to the right. TM's active devices operate in discrete time. At any instant, the active device reads the symbol on one of the tape's squares. Then, the symbol on that square and the device's current state determine what the active device does: what state it goes into and whether it moves left, or moves right, or writes a symbol on the current square (and which symbol it writes). When this happens, we say that an active device *responds* to its internal state and symbol on the tape. All TMs have this structure in common.

Although strictly speaking it is the active devices of TMs that perform operations (on the tape, which is passive), for simplicity we follow the standard convention of ascribing activities to TMs *tout court*. TMs are distinguished from one another by the alphabet they operate on, by the number of their internal states, and more importantly by the particular actions they perform in response to their internal states and the symbols on the tape. A description of the way a particular TM responds to a particular state and symbol is here called an *instruction*. A set of instructions, which uniquely identifies a TM, is called a *TM program*.

To avoid confusion, TMs should be kept distinct from the TM programs that describe their behavior. Unlike digital computers, which compute by executing programs, ordinary TMs do not operate by responding to the TM programs that describe their behavior. Ordinary TMs simply behave in the way described by their TM programs; in other words, their behavior satisfies the instructions contained in their TM program. A TM program identifies a computational process uniquely, and a TM that satisfies the instructions listed in the program is its canonical implementation (i.e., the implementation given by

Turing). But the computations defined by TM programs can also be carried out by humans or machines other than TMs.

Moreover, in section 1.2.4 we shall see that TM programs can be encoded using the alphabet that TMs operate on, and then written on TM tapes. There are special TMs, called universal TMs, which can respond to any TM program written on their tape so as to mimic the behavior of the TMs described by the program. Since universal TMs do compute by responding to TM programs written on their tape, we say that they *execute* TM programs. Needless to say, the behavior of universal TMs is also described by their own TM programs, called universal TM programs. Universal TMs execute the programs written on their tape, but not the universal TM programs that describe their behavior.

In formally defining TM tables, we will use the following ingredients:

Symbols denoting *internal states* of TMs' active devices: $q_1, q_2, q_3, \ldots$

Symbols denoting *symbols* that TMs can print on the tape: $S_0, S_1, S_2, \ldots$ The set of S_i's is our *alphabet*.

Symbols denoting *primitive operations*: R (move to right), L (move to left).

Expressions: finite sequences of symbols.

Instructions: expressions having one of the following forms:

(1) $q_i\ S_j\ S_k\ q_l$,

(2) $q_i\ S_j\ R\ q_l$,

(3) $q_i\ S_j\ L\ q_l$,

(4) $q_i\ S_j\ q_k\ q_l$.

Quadruples of the first type mean that in state q_i reading symbol S_j, the active device will print S_k and go into state q_l. Quadruples of the second type mean that in state q_i reading symbol S_j, the active device will move one square to the right and go into state q_l. Finally, quadruples of the third type mean that in state q_i reading symbol S_j, the active device will move one square to the left and go into state q_l.[7]

We are now ready to define (deterministic) TM programs, their alphabets, and their instantaneous descriptions or snapshots:

[7] Instructions of the fourth type serve to define special TMs called *oracle* TMs and will not be used here.

(Deterministic) TM program: set of instructions that contains no two instructions whose first two symbols are the same.

Alphabet of a TM program: all symbols S_i in the instructions except S_0. For convenience, sometimes we shall write S_0 as B (blank), and S_1 as 1.

Snapshot: expression that contains exactly one q_i, no symbols for primitive operations, and is such that q_i is not the right-most symbol.

A snapshot describes the symbols on a TM tape, the position of the active device along the tape, and the state of the active device. In any snapshot, the S_i's represent the symbols on the tape, q_i represents the state of the active device, and the position of q_i among the S_i's represents the position of the device on the tape. For any tape and any TM program at any computation step, there is a snapshot representing the symbols written on the tape, the state of the device, and its position on the tape. At the next computation step, we can replace the old snapshot by its *successor snapshot*, whose difference from its predecessor indicates all the changes (of the tape, position, and state of the device) that occurred at that step. A snapshot without successors with respect to a TM program M is called a *terminal snapshot* with respect to that program.

Using the notion of snapshot, we can rigorously define computations by TM programs:

Computation by a TM program M: finite sequence of snapshots $a_1 \ldots a_n$ such that $1 \leq i < p$, a_i+1 is the successor of a_i, and a_n is terminal with respect to M. We call a_n the *resultant* of a_1 with respect to M.

For example, let M consist of the following instructions:

$q_1\ S_0\ R\ q_1$,

$q_1\ S_1\ R\ q_1$.

The following are computations of M, whose last line is the resultant of the first line with respect to M:

(1) $q_1S_0S_0S_0$

$S_0q_1S_0S_0$

$S_0S_0q_1S_0$

$S_0S_0S_0q_1$

(2) $q_1S_1S_1S_1$

$S1q_1S_1S_1$

$S_1S_1q_1S_1$

$S_1S_1S_1q_1$

(3) $q_1S_1S_0$

$S_1q_1S_0$

$S_1S_0q_1$.

With each number n we associate the string $\mathbf{n} = 1^{n+1}$. Thus, for example, 4 = 11111. With each k-tuple $(n_1, n_2, \ldots n_k)$ of integers we associate the tape expression $(\mathbf{n_1}, \mathbf{n_2}, \ldots \mathbf{n_k})$, where:

$(\mathbf{n_1}, \mathbf{n_2}, \ldots \mathbf{n_k}) = \mathbf{n_1}B\mathbf{n_2}B \ldots B\mathbf{n_k}$.

Thus, for example, (1, 3, 2) = (**1**, **3**, **2**) = 11B1111B111.

Given an initial snapshot and a program, either there is a computation or there isn't (if there isn't, it's because the list of snapshots is infinite).

Definition 3. An n-ary function $f(x_1, \ldots x_n)$ is *Turing-computable* if and only if there is a Turing Machine M such that: $f(x_1, \ldots x_n)$ is defined if and only if there is a computation of M whose first snapshot is $q_1(\mathbf{x_1}, \ldots \mathbf{x_n})$ and whose resultant contains n_j+1 occurrences of the symbol 1, where $f(x_1, \ldots x_n) = n_j$. We write:

$f(x_1, \ldots x_n) = \Psi_M^n(x_1, \ldots x_n)$.

Turing-computability and partial recursiveness are equivalent notions in the following sense. A function is partial recursive if and only if it is Turing-computable, and it is total recursive if and only if it is a total Turing-computable. In one direction, this is shown by constructing TM programs computing each of the base functions and by showing that TM programs can be manipulated in ways corresponding to the three operations (for the details of the construction, see Davis 1958). The other direction is addressed in the following section.

1.2.4 Gödel Numbers of TM Programs

One way to develop the theory of TM programs is by using recursive functions. I use a method, developed by Gödel (1931), that allows us to use natural numbers as a code for TM instructions, and therefore for TM programs. By studying the properties of TM programs in this way, we will demonstrate the results that we are interested in, namely the existence of universal TMs and the unsolvability of the halting problem. The method followed here has the great advantage of avoiding long and laborious mathematical constructions.

The basic symbols used in formulating TM programs are the following:

R, L

$S_0, S_1, S_2, \ldots$

$q_1, q_2, q_3, \ldots$

We associate each of these symbols to an odd number ≥ 3, as follows:

3	R
5	L
7	S_0
9	q_1
11	S_1
13	q_2

etc.

Hence, for any expression M there is now a finite sequence of odd integers $a_1, a_2, \ldots a_n$ associated to M. Now we'll associate a single number with each such sequence and hence with each expression.

Definition 4. Let M be an expression consisting of the symbols $a_1, a_2, \ldots a_n$. Let $b_1, b_2, \ldots b_n$ be the corresponding integers associated with these symbols. Then the *Gödel number* of M is the following integer:

$$r = \prod_{k=1}^{n} \Pr(k)^{a_k}$$

We write $g_n(M) = r$, and $M = \mathrm{Exp}(r)$. If M is the empty expression, we let $g_n(M) = 1$.

Definition 5. Let $M_1, M_2, \ldots M_n$ be a finite sequence of expressions. Then, the Gödel number of this sequence of expressions is the following integer:

$$r = \prod_{k=1}^{n} \Pr(k)^{gn(M_k)}$$

It is easy to prove that any expression and any sequence of expressions have a unique Gödel number. Since TM programs are sets of instructions not lists of them, any TM program consisting of n instructions has n! Gödel numbers.

Definition 6. For each n>0, let $T_n(z, x_1, \ldots x_n, y)$ be the predicate that means, for given $z, x_1, \ldots x_n, y$, that z is a Gödel number of a TM program Z, and that y is the Gödel number of a computation, with respect to Z, beginning with snapshot $q_1(\mathbf{x_1}, \ldots \mathbf{x_n})$.

These predicates express the essential elements of the theory of TM programs.

Davis 1958 contains the detailed construction proving that for each n>0, $T_n(z, x_1, \ldots x_n, y)$ is primitive recursive, that every Turing-computable function is partial recursive, and that every total Turing-computable function is total recursive.

1.2.5 Universal TM programs

We are now ready to demonstrate that there are universal TMs, which compute any function computable by a TM. Consider the partial recursive binary function $f(z, x) = u_1^1(\min_y T(z, x, y))$. Since this function is Turing-computable, there is a TM program U such that:

$$\Psi_U^2(z, x) = f(z, x)$$

This program is called *universal* TM program. It can be employed to compute any partially computable (singulary; but generalizable to n-ary) function as follows: If Z_0 is any TM program and if z_0 is a Gödel number of Z_0, then:

$$\Psi_U^2 (z_0, x) = \Psi_{Z0}(x)$$

Thus, if the number z_0 is written on the tape of U, followed by the number x_0, U will compute the number $\Psi_{Z0} (x_0)$.

1.2.6 Unsolvability of the Halting Problem

We now discuss the function HALT(x, y), defined as follows. For a given y, let P be the TM program such that $g_n(P) = y$. Then HALT(x, y) = 1 if $\Psi_P(x)$ is defined and HALT(x, y) = 0 otherwise. In other words HALT(x, y) = 1 if and only if the TM program with Gödel number y eventually halts on input x, otherwise it's equal to 0. We now prove the unsolvability of the halting problem.

Theorem. HALT(x, y) is not a recursive function.

Proof. Define the total function g(x) = HALT(x, x), and the partial function h(x) = 0 if g(x) = 0, h(x) undefined if g(x) = 1. If h is partial recursive, then there is a TM program P' with Gödel number i such that for all x, $h(x) = \Psi_{P'}(x)$. But then:

$$h(i) = \Psi_{P'}(i) = 0 \text{ if and only if } g(i) = 0 \text{ if and only if } \Psi_{P'}(i) \text{ is undefined,}$$

which is a contradiction. Therefore, h cannot be partial recursive, so that g and hence HALT cannot be total recursive. QED

This theorem gives us an example of a function that is not computable by a TM program. Computability theory shows that there are infinitely many such functions. Assuming the truth of the Church-Turing thesis, which will be discussed in the next section, we conclude that there is no algorithm computing the halting function. The same holds for any other non-Turing-computable total function.

1.3 The Church-Turing Thesis

Turing (1936-7) introduced his machines as a way to make precise the informal notion of algorithmic computability or effective calculability, as I introduced them in the first section. Church (1936) proposed a similar thesis using recursive functions, which, as we've seen, are computationally equivalent to TMs. After its proponents, Stephen Kleene (1952) dubbed this the Church-Turing thesis:

(CT) A function is effectively calculable if and only if it is Turing-computable.

CT is generally accepted among mathematicians and computer scientists on what they consider overwhelming evidence in its favor.

In summary, this chapter introduced the informal notion of effective calculability and its formal counterpart—Turing computability. According to the canonical view, CT connects the informal notion of effective calculability, or computability by effective procedure, with the formal one of Turing-computability. There can be no rigorous proof of CT, but there is overwhelming evidence in its favor. In Chapter 7, I will list the evidence for CT, discuss the most important alternatives to this canonical view, and I will conclude with a cautious endorsement of a clarified version of the canonical view. I will then discuss the relevance of CT for computationalism.

Before we get to the detailed discussion of CT, we shall take a close look at how in the 1940s, the notions described in this chapter were used to formulate a novel theory of the brain, computationalism, according to which the brain is a mechanism that performs computations.

2 WARREN MCCULLOCH ON LOGIC, MIND, AND BRAIN, CA. 1920-1936

2.1 Introduction

In the 1940s, Turing's notion of computability and of a universal computing mechanism—which were reviewed in Chapter 1—played an important role in the history of computationalism, namely the view that the brain is a computing mechanism. The central figure in the development of computationalism was neurophysiologist and psychiatrist Warren McCulloch. Other key figures, besides Turing and McCulloch, were mathematicians Norbert Wiener and John von Neumann. This chapter begins to tell how McCulloch and others developed computationalism.

Despite the centrality of computationalism to many disciplines, its early historical development, which took place during the early 1940s, has received little attention and remains poorly understood. Most of the existing work focuses on symbolic artificial intelligence (AI) and the cognitivist movement since the late 1950s (McCorduck 1979, Gardner 1985, Crevier 1993). This research pays little attention to the origin of computationalism in the 1940s and early 1950s. One partial exception is the work of Dupuy (2000), who urges historians to look at the cybernetic movement as the source of computationalism. But Dupuy's work is more a philosophical argument in favor of a brand of cognitive science based on the study of "complexity" than a scholarly study of the history of computationalism (cf. Piccinini 2002).

Outside of the history of cognitive psychology and symbolic AI, there is some work on the relation between computationalism and logic (Webb 1980), but nothing has been written on computationalism in neuroscience, even though computationalism was originally proposed and discussed by a neurophysiologist as a theory of the brain. Another limitation of the existing literature is that it often addresses the origin of computationalism by listing the ideas of intellectual-heroes like Hobbes, Leibniz, and Babbage, usually culminating in Turing's contributions (the best example of this kind is Davis 2000). It's time to move beyond this genius-centered history and study how the dreams of those thinkers

eventually turned into computationalism as a conceptual framework for several research programs, which involved the creation of new scientific communities and institutions, and even new disciplines.

Aside from published sources, I have relied on collections of unpublished material pertaining to Alan Turing, Warren McCulloch, John von Neumann and Norbert Wiener.[1] My research builds on recent scholarship in the history of computability theory (Sieg 1994, 1997; Hallett 1994; Shapiro 1995), Turing's early work on mechanical intelligence (Copeland 2000, Copeland and Proudfoot 1996), the history of biophysical and connectionist modeling (Abraham 2001a, 2001b, 2002; Arbib 2000; Frank 1994; Smalheiser 2000), and the discovery of neural mechanisms (Craver and Darden 2001, Craver 2001). Another resource is the recently published interviews with some early connectionist modelers (Anderson and Rosenfeld 1998). My research complements this recent work by showing how early attempts at mathematical modeling of the brain, the development of new mathematical and modeling tools, and new hypotheses about memory mechanisms came together to form a novel conceptual and methodological framework for studying the functional organization of the brain.

In this historical part of the dissertation, I will cover the period going roughly from the mid-1930s, when a number of scientists interested in computation and the brain began to meet one another, to 1946, when these scientists formed a small scientific community, whose members were sharing their work and organizing conferences.

2.2 Background

McCulloch was an eclectic neurophysiologists and psychiatrist, whose main goal was to explain the mind in terms of neural mechanisms. This was not a new project: for instance, an illustrious antecedent was

[1] The Alan Mathison Turing Papers are at Archives of King's College, Cambridge, UK. The Warren S. McCulloch Papers are at the Library of the American Philosophical Society, Philadelphia, PA. The Papers of John von Neumann are at the Library of Congress, Washington, DC. The Norbert Wiener Papers are at the Institute Archives and Special Collections, MIT Libraries, Cambridge, Massachusetts, under MC 22. Whenever possible, I will indicate the box and file folder (abbreviated ff.) of the unpublished documents I refer to.

Sigmund Freud's unpublished *Project for a Scientific Psychology* (Freud 1895).[2] Freud thought that neuronal activity must embody ideas. Since the nineteenth century, neurophysiologists realized that nerve fibers carry electrical pulses, and that these pulses can have either excitatory or inhibitory actions on other nerve fibers. By the end of the nineteenth century, neurophysiologists reached a kernel of consensus that the nervous system is made out of individual cells called neurons, which are connected together in vast networks. The pulse trains on nerve fibers were largely interpreted as carrying meaningful messages, but there was no detailed account of how these messages were processed by the brain. Freud's theory was that energy was discharged from neuron to neuron, and that a neuron's energy level corresponded to the activation of an idea in the mind.

McCulloch's views about mind and brain originated during the 1920s and reached maturity at the beginning of the 1940s, half a century after Freud's. Unlike Freud, McCulloch did not primarily rely on the energy levels and energy flow between neurons. Instead, McCulloch thought that the relevant entity transmitted from neurons to neurons was "information." McCulloch's notion of "information" derived from the power of formal logic to derive conclusions from premises. According to McCulloch, an important aspect of mind was "formal," and was constituted by inferences that can be modeled by a logical calculus. Since logical calculi could be implemented in computing mechanisms, McCulloch thought the brain must be a computing mechanism embodying a logical calculus that constituted the formal aspect of the mind. McCulloch didn't stop at his formulation of a general theory of the brain: he spent a good portion of his time after 1943 devising testable hypotheses about specific neural mechanisms and how they might explain some mental function. So, McCulloch was one of the originators not only of computationalism as a doctrine, but also of the methodology of building models of neural or cognitive mechanisms, based on the computationalist doctrine, to explain various aspects of the mind. Understanding McCulloch's project is crucial to understanding both the cybernetic movement and the subsequent history of AI and cognitive science.

[2] For an account of Freud's *Project* and its significance for Freud's psychoanalysis, see Fancher 1973.

The difference between McCulloch and Freud's projects was made possible by two main historical developments: the rise of mathematical logic and the establishment of the all-or-none law of neural activity.

In the years 1910-1013, Alfred North Whitehead and Bertrand Russell published their *Principia Mathematica*. It contained a powerful formal logical system, whose purpose was to prove all mathematical theorems on logical grounds. Whether Whitehead and Russell succeeded in deriving mathematics from logic alone remained controversial, but the unquestionable deductive power of their formal system popularized mathematical logic in both philosophy and mathematics. Mathematicians developed and applied Whitehead and Russell's techniques to study the foundations of mathematics, whereas philosophers applied those techniques to problems in epistemology, philosophy of language, and other areas.[3]

In subsequent work, Russell himself developed a view called logical atomism, according to which ordinary physical objects should be understood as constructions made out of logical atoms (such as "red here now") by means of logical techniques. Also, according to Russell, our knowledge of ordinary physical objects could be reduced by logical means to our knowledge of sense data, analogously to how mathematical theorems could be derived from the logical system of the *Principia* (Russell 1914).[4] The most detailed and rigorous attempt to carry out this epistemological project was made by Rudolf Carnap (Carnap 1928).[5] We shall see that both the *Principia* and Russell's epistemological project motivated McCulloch's project for reducing the mind to the brain.[6]

---

[3] For more details on the *Principia*, see Irvine 2002.

[4] For more details on Russell's philosophy, see Irvine 2001.

[5] Clark Glymour has gone as far as ascribing Carnap the first computational theory of mind:

> The first explicitly computational theory of cognitive capacities is Rudolf Carnap's *Der Logische Aufbau der Welt*. Carnap's book offered an account of how concepts of color, sound, place, and object could be formed from elements consisting of gestalt experiences and a relation ("recollection of similarity") between such experiences. The theory was given as a logical construction, but also as what Carnap called a "fictive procedure". The procedural characterization is in fact a series of algorithms that take as input a finite list of pairs of objects (the "elementary experiences") such that there is a recollection of similarity between the first and the second member of each pair. The book was of course written before there were computers or programming languages, but it would nowadays be an undergraduate effort to put the whole thing into LISP code (Glymour 1990, p. 67).

In the first quarter of the 20th century, much work in neurophysiology focused on the nature of the impulses traveling through nerve fibers. In 1926, Edgar Adrian started publishing his groundbreaking recordings from nerve fibers (Adrian 1926, Adrian and Zotterman 1926). Adrian's work was reported in newspapers in London and New York, and Adrian continued to extend his results and publish them through the late 1920s. In 1930, A.V. Hill proposed Adrian for the Nobel Prize in physiology or medicine, which was awarded to Adrian (shared with Charles Sherrington) in 1932. Adrian's work was being publicly recognized as the definitive experimental demonstration of the all-or-none law, namely "that the intensities of sensation and response depend simply upon the *number* of nerve impulses which travel to or from the nervous system, per unit of time."[7] We shall see that around 1929, the all-or-none law would have an important impact on McCulloch's thinking about brain and mind.

2.3 Logic, Epistemology, and the Brain

McCulloch's interest in logic and epistemology goes as far back as his college years. In 1917, he was a freshman at Haverford College, where he studied mathematics and medieval philosophy. He was fond of recalling an exchange he had at Haverford with a philosophy teacher, Rufus Jones. McCulloch told his teacher that he wanted to know, "What is a number, that a man may know it; and a man, that he may know a number?" Jones replied: "Friend, thee will be busy as long as thee lives." McCulloch described his work of a lifetime as pursuing and accomplishing that project, and more generally of answering "the general questions of how we can know anything at all."[8]

See also Glymour, Ford, and Hayes 1995. Glymour's point here may be justified conceptually, but there is little evidence that Carnap's *Aufbau* played a very important role in the history of computationalism.

[6] Lettvin, a long-time friend of McCulloch and Pitts, wrote:

> Strongly in the minds of both McCulloch and Pitts were the notions of Russell as contained in his essays on mind, the notions of Peirce, and to a great extent, the notions of Whitehead, in particular as regards the structure of mind and experience (Lettvin 1989a, p. 12).

[7] Letter by A.V. Hill to the Nobel Committee, cited by Frank 1994, p. 209. For a detailed history of the experimental demonstration of the all-or-none law, see Frank 1994.

[8] Biographical Sketch of Warren S. McCulloch, ca. 1942. Warren S. McCulloch Papers, ff. Curriculum Vitae. McCulloch 1961; McCulloch 1974, pp. 21-22. The episode involving his philosophy teacher is also cited by many friends of McCulloch.

After serving in the Navy during World War I, in 1920 McCulloch transferred to Yale, where he majored in philosophy and minored in psychology in 1921. Among other things, he recalled reading Immanuel Kant's *Critique of Pure Reason*, whose notion of synthetic *a priori* knowledge would be a major influence on his thinking about the brain. He also studied Aristotle, the British Empiricists, Charles Sanders Peirce, Georg Wilhelm Friedrich Hegel, and Whitehead and Russell's *Principia Mathematica*. He thought that by defining numbers as classes of classes and formally deriving mathematics from logic, Whitehead and Russell had answered satisfactorily his first question, "What is a number, that a man may know it?" He would devote his efforts to the other question.[9]

That other question was, what is a man, that he may know a number? McCulloch's first step towards an answer was a sort of mental atomism, according to which there must be atomic mental events that carry truth values. More complex mental events can be logically constructed out of atomic ones. It is not known exactly when or how McCulloch developed his mental atomism, but McCulloch's retrospective on his philosophical education suggests that even at this early stage of his life, he freely interpreted previous philosophical work along mental atomistic lines:

> I turned to Russell and Whitehead—(Principia Mathematica)—who in the calculus of atomic propositions were seeking the least event that could be true and [*sic*] false. Leibnitz's problem of the *petit perception* had become the psychophysical problem of the just noticeable difference, called JND, which has since been found in the middle range of perception to be roughly proportional to the size of the stimulus.[10]

JND is the smallest difference between two stimuli that can be discriminated by a subject. McCulloch continued his retrospective by arguing that JND entailed that there were atomic mental signals:

> In searching for these unit signals of perception I came on the work of René Descartes… For him the least true or false event would have been his postulated hydraulic pulse in a single tube, now called an axon.[11]

These comments are revealing in several ways. They show McCulloch's keen interest in philosophy as well as his attempt to recruit past philosophical works to serve his own projects. They also exhibit

[9] McCulloch 1974, p. 22. Lettvin 1989a, p. 11.
[10] McCulloch 1974, pp. 24-25.
[11] McCulloch 1974, p. 26.

McCulloch's elliptical writing style and some bits of his personal jargon. He did not explain what he took a "least true or false event" to be.

McCulloch said that in 1920, he tried to construct what he called "a logic for transitive verbs of action and contra-transitive verbs of perception," on which he continued to work until February 1923.[12] By then, he was completing an M.A. in psychology at Columbia University, which he had started after his B.A. At Columbia he became very interested in physiological psychology and studied mathematics, physics, chemistry, and neuroanatomy.[13] By the time he took his M.A. from Columbia, "he had become convinced that to know how we think and know, he must understand the mechanism of the organ whereby we think and know, namely the brain."[14] After that, he enrolled in medical school at Columbia, with the goal of learning enough physiology to understand how brains work.[15] Both as a student of medicine and later as an intern, he mostly focused on neurology and psychiatry, with the hope of developing a theory of neural function.

While pursuing his medical studies, and after he abandoned his project of a logic of verbs of action and perception, McCulloch allegedly developed a psychological theory of mental atoms. He postulated atomic mental events, which he called "psychons," in analogy with atoms and genes:

> My object, as a psychologist, was to invent a kind of least psychic event, or "psychon," that would have the following properties: First, it was to be so simple an event that it either happened or else it did not happen. Second, it was to happen only if its bound cause had happened … that is, it was to imply its temporal antecedent. Third, it was to propose this to subsequent psychons. Fourth, these were to be compounded to produce the equivalents of more complicated propositions concerning their antecedents.[16]

McCulloch said he tried to develop a propositional calculus of psychons. Unfortunately, the only known records of this work are a few passages in later autobiographical essays by McCulloch himself.[17] In the absence of primary sources, it's difficult to understand the exact nature of McCulloch's early project. A

[12] He did not say why he dropped the project, though he said he encountered "many pitfalls" (McCulloch 1974, pp. 28-29). Unfortunately, I found no record of this work.
[13] McCulloch 1974, p. 30.
[14] Biographical Sketch of Warren S. McCulloch, ca. 1948. Warren S. McCulloch Papers, ff. Curriculum Vitae.
[15] McCulloch 1974, p. 30.
[16] McCulloch 1961, p. 8.
[17] On McCulloch's early psychological theory, see McCulloch 1961, pp. 8-9; McCulloch 1965, pp. 392-393; Abraham 2002, p. 7.

key point is that a psychon is "equivalent" to a proposition about its temporal antecedent. In more modern terminology, McCulloch seemed to think that a psychon had a propositional content, which contained information about that psychon's cause. A second key point is that a psychon "proposes" something to a subsequent psychon. This seems to mean that the content of psychons could be transmitted from psychon to psychon, generating "the equivalents of" more complex propositions. These themes would play an important role in McCulloch's mature theory of the brain.

McCulloch did his internship in organic neurology under Foster Kennedy at Bellevue Hospital in New York, where he finished in 1928.[18] While working as an intern, he "was forever studying anything that might lead me to a theory of nervous function."[19] He developed a long-term interest in closed loops of activity in the nervous system, namely activity flowing through neurons arranged in closed circuits. Since neural activity flowing in circles along close circuits could feed itself back onto the circuit, thereby sustaining itself indefinitely, McCulloch called this process "reverberation." At that time, there was no evidence of closed anatomical loops within the central nervous system, although McCulloch attributed to Ramón y Cayal the hypothesis that they existed.

The tremors of Parkinson's disease, McCulloch thought, could be explained by closed loops of activity connecting the spinal cord and the contracting muscles. With his fellow intern Samuel Wortis, McCulloch discussed whether the loops that would explain Parkinson's were a local "vicious circle"—namely a closed loop involving only the spine and the muscles but not the brain—or the effect of a close loop of activity in the central nervous system, which sent a cyclical signal to the region of the body affected by the tremor. McCulloch and Wortis wondered whether other diseases, such as epilepsy, could be explained by closed loops of neural activity. They did not consider that closed loops of activity could be a normal feature of the nervous system, in part because their discussions were taking place before Lawrence Kubie published the first theoretical paper postulating closed loops in the central nervous

[18] Biographical Sketch of Warren S. McCulloch, ca. 1948. Warren S. McCulloch Papers, ff. Curriculum Vitae.
[19] McCulloch 1974, p. 30.

system to explain memory (Kubie 1930).[20] Later in his life, McCulloch would hypothesize closed loops as explanations for many normal neural functions.

By the end of his internship at Bellevue in 1928, McCulloch "had become convinced that to understand the workings of the nervous system he needed more physics and chemistry."[21] During the following couple of years, McCulloch studied those subjects, as well as more mathematics, at New York University. During the same period, he taught physiological psychology at Columbia Extension in Brooklyn and he did research with Frank Pike in the Laboratory of Neurosurgery at Columbia and with Wortis in the Laboratory of Experimental Neurology at Bellevue.[22]

In 1929, McCulloch met mathematician R. V. Hartley of Bell Labs. Hartley had defined, for engineering purposes, a quantifiable notion of information divorced from meaning. In Hartley's words, his was "a definite quantitative measure of information based on physical considerations alone."[23] Hartley studied how much information can be transmitted using given codes as well as the effect of noise on the transmission of information. As McCulloch knew, Hartley's work was an important part of the background against which Shannon later formulated his mathematical theory of communication (Shannon 1948; Shannon and Weaver 1949).[24] This shows that early on, McCulloch was interested in engineering problems of communication and acquainted with attempts to formulate a mathematical theory of information.

In 1931, an otherwise unspecified "friend" of McCulloch's translated to him and others a recently published paper by Kurt Gödel "on the arithmetization of logic."[25] This shows that McCulloch was keeping up with important results in the foundations of mathematics.

---

[20] For an account of these events, see McCulloch 1974, pp. 30-31.
[21] Biographical Sketch of Warren S. McCulloch, ca. 1948. Warren S. McCulloch Papers, ff. Curriculum Vitae.
[22] Biographical Sketch of Warren S. McCulloch, ca. 1948. Warren S. McCulloch Papers, ff. Curriculum Vitae.
[23] Hartley 1929, p. 538.
[24] McCulloch 1974, 32. For more on Hartley's work and its relation to Shannon's, see Aspray 1985, pp. 120-122.
[25] McCulloch 1974, p. 32. The translated paper was probably Gödel's famous paper on incompleteness (Gödel 1931), as indicated by a deleted phrase in a draft of McCulloch's 1974 paper (Warren S. McCulloch Papers, Series V Miscellaneous, Box 3).

McCulloch's views about a calculus of psychons underwent an important transformation in 1929. It occurred to him that the all-or-none electric impulses transmitted by each neuron to its neighbors might correspond to the mental atoms of his psychological theory, where the relations of excitation and inhibition between neurons would perform logical operations upon electrical signals corresponding to inferences of his propositional calculus of psychons. His psychological theory of mental atoms turned into a theory of "information flowing through ranks of neurons."[26]

This was McCulloch's first attempt "to apply Boolean algebra to the behavior of nervous nets."[27] The brain would embody a logical calculus like that of Whitehead and Russell, which would account for how humans could perceive objects on the basis of sensory signals and how humans could do mathematics and abstract thinking. This was the beginning of McCulloch's search for the "logic of the nervous system," on which he kept working until his death. A major difficulty to the formulation of his logical calculus was the treatment of closed loops of neural activity. McCulloch was trying to describe the causal structure of neural events by assigning temporal indexes to them. But he thought a close loop meant that one event was its own ancestor, which did not make sense to him. He wanted "to close the loop" between chains of neuronal events, but did not know how to conceive of the events in the close loops. He would not find a solution to this difficulty until he met Walter Pitts in the early 1940s.[28]

In 1932-3, McCulloch was at Rockland State Hospital, still in New York City, to earn money as a psychiatrist. There, he met the German psychiatrist Eilhard von Domarus, who was earning his philosophy Ph.D. at Yale under Filmer Northrop, with a dissertation *On the Philosophic Foundation of Psychology and Psychiatry*. Von Domarus's dissertation interpreted psychoses, such as schizophrenia, as

---

[26] McCulloch 1974, p. 32.

[27] Biographical Sketch of Warren S. McCulloch, ca. 1948. Warren S. McCulloch Papers, ff. Curriculum Vitae. The same Biographical Sketch also says that this was the time when McCulloch "attempted to make sense of the logic of transitive ver[b]s," which conflicts with what he wrote in his later autobiographical essays. Given the lack of primary sources and given McCulloch's inconsistencies in his writings, it is hard to date his early work with certainty. But in spite of some inconsistencies with dates, in all his relevant writings McCulloch emphasized his early interest in logic and his attempts to apply logic to psychology and later to a theory of the brain. It is thus hard to believe Lettvin when he wrote that until McCulloch worked with Pitts in the early 1940s, McCulloch had not applied "Boolean logic" to the working of the brain (Lettvin 1989a, p. 12). Lettvin gave no evidence for this claim. Since Lettvin met McCulloch only around 1940, Lettvin may never have discovered McCulloch's early efforts in this direction.

[28] For an account of these events, see McCulloch 1961; McCulloch 1974, pp. 30-32; Arbib 2000, p. 213.

logical disturbances of thought. But Von Domarus could not write well in English, so McCulloch helped him write his dissertation. Years later, in commenting on von Domarus's dissertation, McCulloch found that "no other text so clearly sets forth the notions needed for an understanding of psychology, psychiatry and finite automata."[29]

2.4 Strychnine Neuronography and the Functional Organization of the Brain

Until 1934, McCulloch was an ambitious psychiatrist with original ideas but little scientific track record. He had only four publications in scientific journals. In 1934, he moved to Yale to work in Joannes Dusser de Barenne's Laboratory of Neurophysiology. Dusser de Barenne was a distinguished Dutch neurophysiologist who had moved from Holland to Yale in 1930.[30] McCulloch worked at Yale until shortly after Dusser de Barenne's death in 1940. McCulloch's work during those years launched his academic career.[31]

With Dusser de Barenne, McCulloch worked mostly on mapping the connections between brain areas. To discover those connections, Dusser de Barenne had developed the method of strychnine neuronography. When strychnine was applied to one brain area, it caused neurons to fire. The pulses from those neurons would activate whichever areas were connected to the first area. By applying strychnine to a cortical area and recording the activity of other brain areas, it was thus possible to map the projections of any area of the cortex. Dusser de Barenne and McCulloch mapped cortico-cortical connections as well as connections between cortical areas and other areas of the monkey brain. McCulloch continued working with strychnine neuronography after leaving Yale for Chicago, where he worked with Percival Bailey, Gerhard von Bonin, and others. He published over forty papers on the subject between 1934 and 1944. In 1944, McCulloch published two review articles on cortical connections, one in the journal *Physiology* (McCulloch 1944a), the other in a reference volume on *The*

[29] Thayer 1967, p. 350, cited by Heims 1991, p. 133; see also McCulloch 1974, pp. 32-3.
[30] McCulloch 1940, p. 271.
[31] McCulloch's many publications on neurophysiology are reprinted in his *Collected Works* (McCulloch 1989).

Precentral Motor Cortex (McCulloch 1944b). His work using strychnine neuronography established him as a leading expert on what he called "the functional organization of the brain."[32]

McCulloch's work in Dusser de Barenne's lab explicitly connected him to an intellectual lineage in neurophysiology that goes from Wilhelm von Helmholtz to Rudolf Magnus to Dusser de Barenne. These authors were concerned with the physiological foundations of perception and knowledge, including the idea that Kant's synthetic *a priori* knowledge is grounded in the anatomy and physiology of the brain. This idea was well expressed in Magnus's lecture "The Physiological *A Priori*" (Magnus 1930), which McCulloch knew and cited. Dusser de Barenne consciously inherited the quest for the physiological *a priori* from his mentor Magnus and transmitted it to his pupil McCulloch.[33] McCulloch saw himself as continuing the tradition from Kant to Dusser de Barenne, and would refer to his theory of the brain as solving the problem of the physiological *a priori*. Partly because of this, he called his intellectual enterprise *experimental epistemology*—in the 1950s, a sign reading "experimental epistemology" hung from his MIT lab's door.[34]

McCulloch said that the work with Dusser de Barenne was important to him, because it made him deal with brains and their activity: "For me it proved that brains do not secrete thought as the liver secretes bile, but that they compute thoughts the way computing machines calculate numbers."[35] Unfortunately, he did not explain in what sense or by what means this neurophysiological work "proved" that brains compute thoughts.

Some clue as to the relationship between McCulloch's work with Dusser de Barenne and McCullolch's view that the brain performs computations was offered by Jerome Lettvin. Lettvin—one of McCulloch's life-long collaborators and friends—offered an explanation that he is likely to have heard

---

[32] For more on Dusser de Barenne and McCulloch's neurophysiological work, see Gershwind 1989; Abraham 2002, pp. 8-11.

[33] According to McCulloch, Dusser de Barenne had worked "intimately" with Magnus (McCulloch 1940, p. 270; McCulloch 1974, p. 22).

[34] Interview with Lettvin, in Anderson and Rosenfeld 1998. McCulloch put it as follows:

> The main theme of the work of my group in neurophysiology in the Research Laboratory of Electronics at the Massachusetts Institute of Technology has been in this tradition, namely, experimental epistemology, attempting to understand the physiological foundation of perception (McCulloch 1974, pp. 22-23).

[35] McCulloch 1974, p. 33.

from McCulloch. According to Lettvin, two aspects of McCulloch's neurophysiological work were especially relevant. First, there was the observation of nerve specificity: stimulating specific nerves or specific brain areas would lead to excitation (or inhibition) of very specific other neural areas, or give rise to specific movements, or specific sensations. This suggested that there were pre-existing paths between specific portions of the nervous system, carrying specific pulses from certain areas to others, and that those pulses gave rise to "all the kinds of perception, thinking, and memory that we enjoy."[36] Second, synaptic action—i.e. the action occurring between neurons—was irreversible, occurring only in one preferred direction and not in the reverse direction. Pulses could travel through the pre-existing paths only in one direction. According to Lettvin:

> It would be impossible to devise a logical system in which the connections were reversible; that is, active informationally in both directions. So, to McCulloch's mind, the existence of a single direction in the nervous system for information reinforced the idea of an essentially logical device.[37]

McCulloch was presumably interpreting his neurophysiological observations on the basis of his pre-existing assumptions about information flow through ranks of neurons. According to McCulloch, by the time he went to work with Dusser de Barenne, McCulloch had already reached the conclusion that neural activity can be modeled by logic. As much as his neurophysiological observations were compatible with such a view, McCulloch's claim that they "proved" it seems to be an overstatement.

While working as an experimental neurophysiologist at Yale, McCulloch established connections with colleagues in the field. An especially important one was with a young Mexican researcher, Arturo Rosenblueth, who was working with Walter Cannon at Harvard Medical School on homeostasis and other topics. At least by 1938, McCulloch and Rosenblueth knew each other and had initiated a dialogue over experimental methods and results in neurophysiology.[38] In the summer of 1941, McCulloch visited

[36] Lettvin 1989a, p. 14.
[37] Ibid.
[38] Letter by McCulloch to J. F. Tönnies, dated April 11, 1938. Warren S. McCulloch Papers, ff. Tönnies. Letter by McCulloch to Rosenblueth, dated December 22, 1939. Warren S. McCulloch Papers, ff. Rosenblueth.

Rosenblueth's lab and they ran a few experiments together.[39] By then, they had also started discussing more theoretical topics, such as von Domarus's dissertation on the foundations of psychiatry.[40] McCulloch and Rosenblueth developed a friendship that lasted for decades.

Rosenblueth shared McCulloch's dissatisfaction with the current lack of theory in neurophysiology, which he expressed to McCulloch as follows:

> It is always difficult to strike the right balance between experiment and hypothesis, but it seems to me, in the main, that a good many of our colleagues—perhaps even ourselves—do not do enough thinking about the large number of experiments carried out. In other words, I found our discussions very stimulating.[41]

Rosenblueth's laudatory reference to his conversations with McCulloch right after his complaint about the lack of hypotheses in neurophysiology suggests that in conversation, McCulloch had manifested a more theoretical bent than many of their colleagues, which Rosenblueth appreciated. This would not be surprising, for McCulloch cared much about theory and throughout his life, he manifested and publicly defended a tendency to formulate hypotheses and theories even in the absence of data to test them.

At Yale, McCulloch also attended a philosophical seminar for research scientists organized by Filmer Northrop, who was von Domarus's old advisor. At one of those seminars, Frederic Fitch, a distinguished logician from Yale's Philosophy Department, presented the theory of deduction of *Principia Mathematica*. McCulloch also attended advanced lectures by Fitch on logical operators and urged Fitch to work on the logic of neural nets.[42]

While McCulloch was at Yale, he became acquainted with the work of J. H. Woodger (1937), who advocated the axiomatic method in biology. In a letter to a colleague written in 1943, McCulloch wrote:

> I personally became acquainted with Woodger because the great interest of the biologists in Yale had led to his coming thither to tackle some of their problems. When he finally departed, it was not because they were not convinced of the value of his attempt but because he was convinced

[39] Two letters by Rosenblueth to McCulloch, dated June 21 and September 5, 1941. Warren S. McCulloch Papers, ff. Rosenblueth.
[40] Letter by McCulloch to Rosenblueth, dated May 1, 1941. Letter by Rosenblueth to McCulloch, dated December 3, 1941. McCulloch papers, ff. Rosenblueth.
[41] Letter by Rosenblueth to McCulloch, dated September 5, 1941. Warren S. McCulloch Papers, ff Rosenblueth.
[42] Heims 1991, p. 34ff.

> that the ambiguity of their statements prevented logical formulation. It was to discussions with him and with Fitch that I owe much of my persistence in attempting a logical formulation of neuronal activity. Until that time I had merely used the nomenclature of the Principia Mathematica to keep track of the activity of neuronal nets.[43]

In the same letter, McCulloch suggested that it was only around this time that he started seeing his theory of the brain as a "theory of knowledge":

> [T]he theory … began originally as a mere calculus for keeping track of observed realities. It was at work for seven years before it dawned on me that it had those logical implications which became apparent when one introduces them into the grandest of all feed-back systems, which runs from the scientist by manipulations through the objects of this world, back to the scientist—so producing in him what we call theories and in the great world are little artifacts.[44]

McCulloch had known Northrop, another member of Yale's Philosophy Department, since 1923, and continued to be in contact with him through the 1930s. Much of Northrop's philosophy was about science and scientific methodology. Northrop believed that scientific disciplines reach maturity when they start employing logic and mathematics in formulating rigorous, axiomatic theories:

> The history of science shows that any empirical science in its normal healthy development begins with a more purely inductive emphasis, in which the empirical data of its subject matter are systematically gathered, and then comes to maturity with deductively-formulated theory in which formal logic and mathematics play a most significant part (Northrop 1940, p. 128; cited by Abraham 2002, p. 6).

Northrop argued that biology was finally reaching its maturity with the work of Woodger (1937) and Nicolas Rashevsky (1938), who had imported formalisms and techniques from mathematical physics into biology.[45]

While McCulloch was working in Dusser de Barenne's lab at Yale, Alan Turing published his famous paper on computability (1936-7), where he drew a clear and rigorous connection between computing, logic, and machines. By the early 1940s, McCulloch had read Turing's paper. In 1948, in a public discussion of his theory of the brain at the Hixon Symposium, McCulloch declared that it was the reading of Turing's paper that led him in the "right direction."[46]

---

[43] Letter by McCulloch to Ralph Lillie, ca. February 1943. Warren S. McCulloch Papers, ff. Lillie.
[44] Ibid.
[45] For a more detailed account of Northrop's philosophy of science, see Abraham 2002, pp. 6-7.
[46] Von Neumann 1951, p. 33.

3 TOWARDS A THEORY OF THE BRAIN, 1936-1942

3.1 What Computing Mechanisms Can Do[1]

The modern mathematical notion of computation, which was developed by Alan Turing in his 1936-7 paper and reviewed in Chapter 1, played a crucial role in the history of computationalism. This section concerns Turing's use of "computable" and "machine" in his logic papers, his version of the Church-Turing Thesis (CT, i.e. that every effectively calculable function is computable by a Turing Machine), and why his early work on computability was not initially read as an attempt to establish, or even to imply, that the mind is a machine.

Today, both the term "computable" and formulations of CT are utilized in many contexts, including discussions of the nature of mental, neural, or physical processes. Some of these uses are discussed at length in the Chapter 7.[2] None of these uses existed at Turing's time, and their superposition onto Turing's words yields untenable results. For instance, according to a popular view, Turing's argument for CT was already addressing the problem of how to mechanize the human mind, while the strength of CT—perhaps after some years of experience with computing machines—eventually convinced Turing that thinking could be reproduced by a computer.[3]

This reading makes Turing appear incoherent. It conflicts with the fact that he, who reiterated CT every time he talked about machine intelligence, never said that the mechanizability of the mind was a consequence of CT. Quite the opposite: in defending his view that machines could think, he felt the need

[1] This section is adapted from a section of larger paper, devoted to Turing's ideas on logical proofs and machine intelligence (Piccinini 2003a).
[2] For a survey of different uses see Odifreddi 1989, I.8. (Odifreddi writes "recursive" instead of "computable.")
[3] See e.g. Hodges 1983, esp. p. 108; also Hodges 1988, 1997; Leiber 1991, pp. 57, 100; Shanker 1995, pp. 64, 73; Webb 1980, p. 220. Turing himself is alleged to have argued, in his 1947 "Lecture to the London Mathematical Society," that "the Mechanist Thesis ... is in fact *entailed* by his 1936 development of CT" (Shanker 1987, pp. 615, 625). Since Shanker neither says what the Mechanist Thesis is, nor provides textual evidence from Turing's lecture, it is difficult to evaluate his claim. If the Mechanist Thesis holds that the mind is a machine or can be reproduced by a machine, we'll see that Shanker is mistaken. However, some authors—other than Turing—do believe CT to entail that the human mind is mechanizable (e.g., Dennett 1978a, p. 83, Webb 1980, p. 9). Their view is discussed in the Chapter 7.

to respond to many objections.[4] If one wants to understand the development of Turing's ideas on mechanical intelligence, his logical work on computability must be understood within its context. The context would change in the 1940s with the publication of McCulloch and Pitts's computational theory of the brain—which will be discussed in the next chapter—and the subsequent rise of computationalism. This change in context explains why in the second half of the 20th century, many found it so natural to read Turing's logical work as defending a form of computationalism.

But in the 1930s there were no working digital computers, nor was cognitive science on the horizon. There did exist some quite sophisticated computing machines, which at the time were called differential analyzers and would later be called analog computers. Differential analyzers had mechanical gears that obeyed certain types of differential equations. By setting up the gears in appropriate ways, differential analyzers could solve certain systems of differential equations. At least as early as 1937, Turing knew about the Manchester differential analyzer, which was devoted to the prediction of tides, and planned to use a version of it to find values of the Riemann zeta function.[5]

In the 1930s and up through the 1940s, the term "computer" was used to refer to people reckoning with paper, pencil, and perhaps a mechanical calculator. Given the need for laborious calculations in industry and government, skilled individuals, usually young women, were hired as "computers." In this context, a "computation" was something done by a computing human.

The origins of "Computable Numbers" can be traced to 1935, when Turing graduated in mathematics from King's College, Cambridge, and became a fellow of King's. In that year, he attended an advanced course on Foundations of Mathematics by topologist Max Newman. Newman, who became Turing's lifelong colleague, collaborator, and good friend, witnessed the development of Turing's work

[4] Indeed, in his most famous paper on machine intelligence, Turing admitted: "I have no very convincing arguments of a positive nature to support my views. If I had I should not have taken such pains to point out the fallacies in contrary views" (Turing 1950, p. 454).
[5] Hodges 1983, pp. 141, 155-8.

on computability, shared his interest in the foundations of mathematics, and read and commented on Turing's typescript before anyone else.[6]

In his biography of Turing as a Fellow of the Royal Society, Newman links "Computable Numbers" to the attempt to prove rigorously that the decision problem for first order logic, formulated by David Hilbert within his program of formalizing mathematical reasoning (Hilbert and Ackermann 1928), is unsolvable in an absolute sense. "[T]he breaking down of the Hilbert programme," said Newman, was "the application [Turing] had principally in mind."[7] In order to show that there is no effective procedure—or "decision process"—solving the decision problem, Turing needed:

> … to give a definition of 'decision process' sufficiently exact to form the basis of a mathematical proof of impossibility. To the question 'What is a "mechanical" process?' Turing returned the characteristic answer 'Something that can be done by a machine,' and embarked in the highly congenial task of analyzing the general notion of a computing machine.[8]

Turing was trying to give a precise and adequate definition of the intuitive notion of effective procedure, as mathematicians understood it, in order to show that no effective procedure could decide first order logical provability. When he talked about computations, Turing meant sequences of operations on symbols (mathematical or logical), performed either by humans or by mechanical devices according to a finite number of rules—which required no intuition or invention or guesswork—and whose execution always produced the correct solution.[9] For Turing, the term "computation" by no means referred to all that mathematicians, human minds, or machines could do.

---

[6] Hodges 1983, pp. 90-110.
[7] Newman 1955, p. 258.
[8] Ibid.
[9] See his argument for the adequacy of his definition of computation in Turing, 1936-7, pp. 135-8. The last qualification—about the computation being guaranteed to generate the correct solution—was dropped after "Computable Numbers." In different writings, ranging from technical papers to popular expositions, Turing used many different terms to explicate the intuitive concept of effective procedure: "computable" as "calculable by finite means" (1936-7), "effectively calculable" (1936-7, pp. 117, 148; 1937, p. 153), "effectively calculable" as a function whose "values can be found by a purely mechanical process" (1939, p. 160), "problems which can be solved by human clerical labour, working to fixed rules, and without understanding" (1945, pp. 38-9), "machine processes and rule of thumb processes are synonymous" (1947, p. 112), "'rule of thumb' or 'purely mechanical'" (1948, p. 7), "definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner" (1951, p. 1), "calculation" to be done according to instructions explained "quite unambiguously in English, with the aid of mathematical symbols if required" (1953, p. 289).

Turing rigorously defined "effectively calculable" with his famous machines: a procedure was effective if and only if a Turing Machine could carry it out. "Machine" requires a gloss. Given the task of "Computable Numbers," viz. establishing a limitation to what could be achieved in mathematics by effective methods of proof, it is clear that Turing Machines represented (at the least) computational abilities of human beings. As a matter of fact, the steps these machines carried out were determined by a list of instructions, which must be understandable unambiguously by human beings.

But Turing's machines were not portrayed as understanding instructions—let alone intelligent. Even if they were anthropomorphically described as "scanning" the tape, "seeing symbols," having "memory" or "mental states," etc., Turing introduced all these terms in quotation marks, presumably to underline their metaphorical use.[10] If one thinks that carrying out a genuine, "meaningful" computation—as opposed to a "meaningless" physical process—presupposes understanding the instructions, one should conclude that only humans carry out genuine computations. Turing Machines, in so far as they computed, were abstract representations of idealized human beings. These considerations, among others, led some authors to a restrictive interpretation: Turing's theory bears on computability by humans not by machines, and Turing Machines are "*humans* who calculate."[11]

This interpretation is at odds with Turing's use of "computation" and "machine," and with his depiction of his work. Turing never said his machines should be regarded as idealized human beings—nor anything similar. We saw that, for him, a computation was a type of physical manipulation of symbols. His machines were introduced to define rigorously this process of manipulation for mathematical purposes. As Turing used the term, machines were idealized mechanical devices; they

[10] Turing 1936-7, pp. 117-8.

[11] This widely cited phrase is in Wittgenstein 1980, sec. 1096. Wittgenstein knew Turing, who in 1939 attended Wittgenstein's course on Foundations of Mathematics. Wittgenstein's lectures, including his dialogues with Turing, are in Wittgenstein 1976. Discussions of their different points of views can be found in Shanker 1987; Proudfoot and Copeland 1994. Gandy is more explicit than Wittgenstein: "Turing's analysis makes no reference whatsoever to calculating machines. Turing machines appear as a result, as a codification, of his analysis of calculations by humans" (Gandy 1988, p. 83-4). Sieg quotes and endorses Gandy's statement (Sieg 1994, p. 92; see also Sieg 1997, p. 171). Along similar lines is Copeland, 2000, pp. 10ff. According to Gandy and Sieg, "computability by a machine" is first explicitly analyzed in Gandy, 1980. In the present chapter I am only concerned with the historiographical merits of the Gandy-Sieg view, and not with its philosophical justification. The latter issue is addressed in Chapter 7.

could be studied mathematically because their behavior was precisely defined in terms of discrete, effective steps.

There is evidence that Turing, in 1935, talked about building a physical realization of his universal machine.[12] Twelve years later, to an audience of mathematicians, he cited "Computable Numbers" as containing a universal digital computer's design and the theory establishing the limitations of the new computing machines:

> Some years ago I was researching on what might now be described as an investigation of the theoretical possibilities and limitations of digital computing machines. I considered a type of machine which had a central mechanism, and an infinite memory which was contained on an infinite tape. This type of machine appeared to be sufficiently general. One of my conclusions was that the idea of a 'rule of thumb' process and a 'machine process' were synonymous … Machines such as the ACE [Automatic Computing Engine] may be regarded as practical versions of this same type of machine.[13]

Therefore, a machine, when Turing talked about logic, was not (only) a mathematical representation of a computing human, but literally an idealized mechanical device, which had a

---

[12] Newman 1954; Turing 1959, p. 49. Moreover, in 1936 Turing wrote a précis of "Computable Numbers" for the French *Comptes Rendues*, containing a succinct description of his theory. The definition of "computable" is given directly in terms of machines, and the main result is appropriately stated in terms of machines:

> On peut appeler 'computable' les nombres dont les décimales se laissent écrire par une machine . . . On peut démontrer qu'il n'y a aucun procédé général pour décider si une machine M n'écrit jamais le symbole 0 (Turing, 1936).

The quote translates as follows: We call "computable" the numbers whose decimals can be written by a machine … We demonstrate that there is no general procedure for deciding whether a machine M will never write the symbol 0. Human beings are not mentioned.

[13] Turing 1947, pp. 106-7. See also ibid., p. 93. Also, Turing machines "are chiefly of interest when we wish to consider what a machine could in principle be designed to do" (Turing 1948, p. 6). In this latter paper, far from describing Turing machines as being humans who calculate, Turing described human beings as being universal digital computers:

> It is possible to produce the effect of a computing machine by writing down a set of rules of procedure and asking a man to carry them out. Such a combination of a man with written instructions will be called a 'Paper Machine.' A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine (Turing 1948, p. 9).

Before the actual construction of the ACE, "paper machines" were the only universal machines available, and were used to test instruction tables designed for the ACE (Hodges 1983, chapt. 6). Finally:

> A digital computer is a universal machine in the sense that it can be made to replace . . . any rival design of calculating machine, that is to say any machine into which one can feed data and which will later print out results (Turing, 'Can digital computers think?' Typescript of talk broadcast in BBC Third Programme 15 May 1951, AMT B.5, Contemporary Scientific Archives Centre, King's College Library, Cambridge, p. 2).

Here, Turing formulated CT with respect to all calculating machines, without distinguishing between analog and digital computers. This fits well with other remarks by Turing, which assert that any function computable by analog machines could also be computed by digital machines (Turing 1950, pp. 451-2). And it strongly suggests that, for him, any device mathematically defined as giving the values of a non-computable function, that is, a function no Turing machine could compute—like the "oracle" in Turing 1939, pp. 166-7—could not be physically constructed.

potentially infinite tape and never broke down. Furthermore, he thought his machines could compute any function computable by machines. This is not to say that, for Turing, every physical system was a computing machine or could be mimicked by computing machines. The outcome of a random process, for instance, could not be replicated by any Turing Machine, but only by a machine containing a "random element."[14]

Such was the scope of CT, the thesis that the numbers computable by Turing Machines "include all numbers which could naturally be regarded as computable."[15] To establish CT, Turing compared "a man in the process of computing . . . to a machine."[16] He based his argument on limitations affecting human memory and perception during the process of calculation. At the beginning of "Computable Numbers," one reads that "the justification [for CT] lies in the fact that the human memory is necessarily limited."[17] In the argument, Turing used human sensory limitations to justify his restriction to a finite number of primitive symbols, as well as human memory limitations to justify his restriction to a finite number of "states of mind."[18]

Turing's contention was that the operations of a Turing machine "include all those which are used in the computation of a number" by a human being.[19] Since the notion of the human process of computing, like the notion of effectively calculable, was an intuitive one, Turing asserted that "all arguments which can be given [for CT] are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically."[20] In other words, CT was not a mathematical theorem.[21]

From "Computable Numbers," Turing extracted the moral that effective procedures, "rule of thumb processes," or instructions explained "quite unambiguously in English," could be carried out by his machines. This applied not only to procedures operating on mathematical symbols, but to any symbolic

[14] Turing 1948, p. 9; 1950, p. 438.
[15] Turing 1936-7, p. 116.
[16] Ibid., p. 117.
[17] Ibid., p. 117.
[18] Turing 1936-7, pp. 135-6.
[19] Ibid., p. 118.
[20] Ibid., p. 135.
[21] CT is usually regarded as an unprovable thesis for which there is compelling evidence. The issue of the provability of CT is discussed at length in Chapter 7.

procedure so long as it was effective. A universal machine, if provided with the appropriate instructions, could carry out all such processes. This was a powerful thesis, but very different from the thesis that "thinking is an effective procedure."[22] In "Computable Numbers" Turing did not argue, nor did he have reasons to imply from CT, that all operations of the human mind could be performed by a Turing Machine.

After Turing published his paper, a number of results were quickly established to connect his work to other proposed formal definitions of the effectively calculable functions, such as general recursiveness (Gödel 1934) and λ-definability (Church 1932, Kleene 1935). It was shown that all these notions were extensionally equivalent in the sense that any function that fell under any one of these formal notions fell under all of them. Mathematicians took this as further evidence that the informal notion of effective calculability had been captured.

In the 1930s and 1940s, Turing's professionally closest colleagues read his paper as providing a general theory of computability, establishing what could and could not be computed—not only by humans, but also by mechanical devices. This is how McCulloch and John von Neumann, among others, read Turing's work.[23]

At the time he published "Computable Numbers," Turing moved to Princeton to work on a Ph.D. dissertation under Alonzo Church. At Princeton, Turing met von Neumann, who was a member of the Institute for Advanced Studies. Von Neumann was a Hungarian-born mathematician who had worked in logic and foundations of mathematics but was then working mostly in mathematical physics. In 1938, von Neumann invited Turing to become his assistant. Turing declined and went back to England in 1939.

---

[22] According to Shanker, this was Turing's "basic idea" (1995, p. 55). But Turing never made such a claim.

[23] E.g., see Church 1937; 1956, p. 52, n.119; Watson 1938, p. 448ff; Newman 1955, p. 258; Kleene said: "Turing's formulation comprises the functions computable by machines" (1938, p. 150). When von Neumann placed "Computable Numbers" at the foundations of the theory of finite automata, he introduced the problem addressed by Turing as that of giving "a general definition of what is meant by a computing automaton" (von Neumann 1951, p. 313). Most logic textbooks introduce Turing machines without qualifying "machine," the way Turing did. More recently, doubts have been raised about the generality of Turing's analysis of computability by machines (e.g., by Siegelmann 1999). These doubts are discussed in Chapter 7.

3.2 Teleological Mechanisms

When he met Turing, von Neumann already knew Norbert Wiener. Wiener was an MIT mathematician who shared von Neumann's background in logic and current mathematical interests. Wiener was also trained in philosophy, in which he had published several papers in the 1910s.[24] Wiener and von Neumann met around 1933 and became friends, beginning a scientific dialogue that lasted many years.[25] Von Neumann sometimes referred to their meetings as "mathematical conversations" to which he looked forward.[26]

Wiener was interested in designing and building computing mechanisms to help solve problems in mathematical physics. His correspondence about mechanical computing with Vannevar Bush, the main designer of the differential analyzer and other analog computing mechanisms, stretches as far back as 1925.[27] At least since the mid 1930s, Wiener became involved in the design of analog computing devices. According to Bush, Wiener made the original suggestion for the design of an analog computing mechanism called the optical integraph, and was an expert on analog computing in general.[28] In 1935, Wiener proposed to Bush "a set up of an electrical simultaneous machine," which was another analog machine. Wiener briefly described the machine and its proposed use. In the same letter, he also commented on a paper by Bush on some other analog machine, which might have been the differential analyzer.[29]

In 1940, Bush was Chairman of the National Defense Research Committee of the Council of National Defense. World War II had started and the US would soon enter it. The National Defense Research Committee was in charge of recruiting talented scientists and assigning them to "national defense" projects. Wiener proposed Bush a new design for a computing machine for solving boundary

[24] Reprinted in Wiener 1976.
[25] Letter by von Neumann to Wiener, dated November 26, 1933. Norbert Wiener Papers, Box 2, ff. 38. Letter by von Neumann to Wiener, dated March 9, 1953. Norbert Wiener Papers, Box 11, ff. 166. For a comprehensive study of the relationship between Wiener and von Neumann, see Heims 1980.
[26] Letter by von Neumann to Wiener, dated April 9, 1937. Norbert Wiener Papers, Box 3, ff. 47.
[27] Letter by Wiener to Bush, dated August 18, 1925. Norbert Wiener Papers, Box 2, ff. 27.
[28] Letter by Bush to Ku, dated May 26, 1935. Norbert Wiener Papers, Box 3, ff. 42.
[29] Letter by Wiener to Bush, dated September 22, 1935. Norbert Wiener Papers, Box 3, ff. 43.

value problems in partial differential equations. Unlike analog computing machines, where the values of differential equations were represented by continuously varying physical variables, the method proposed by Wiener consisted in replacing a differential equation with a difference equation asymptotically equivalent to it, and using the difference equation to generate successive representations of the states of the system using the "binary system" (i.e., binary notation).[30] Wiener's proposed machine was thus of a kind that a few years later would be called digital rather than analog.

Bush responded to Wiener's proposal with interest and asked for clarifications on the design and range of applicability of the machine.[31] Wiener convinced Bush that his proposed machine would be "valuable" and "could be successfully constructed," and Bush seriously considered whether to assign funds for its construction.[32] He ultimately declined to fund the project, because Wiener's machine would yield mostly long-term advantages to national defense, and the researchers who were qualified to work on developing Wiener's machine were needed for more urgent "defense research matters."[33]

Wiener also had a long-standing interest in biology. In the late 1930s he met Arturo Rosenblueth and started discussing with him the possibility of formulating a mathematical characterization of the possible behaviors of organisms, analogously to how engineers described the possible behaviors of machines. Wiener explained his project in a letter to an old acquaintance, the British biologist J. B. S. Haldane:

> I am writing to you … [about] some biological work which I am carrying out together with Arturo Rosenblueth… Fundamentally the matter is this: Behaviorism as we all know is an established method of biological and psychological study but I have nowhere seen an adequate attempt to analyze the intrinsic possibilities of types of behavior. This has become necessary to me in connection with the design of apparatus to accomplish specific purposes in the way of the repetition and modification of time patterns. … [T]he problem of examining the behavior of an instrument from this point of view is fundamental in communication engineering and in related fields where we often have to specify what the apparatus between four terminals in a box is to do for we take up the actual constitution of the apparatus in the box. We have found this method of examining possibilities of time pattern behavior, quite independently of their realization, a most useful way of preparing for their later realization. Now this has left us with a large amount of

[30] Letters by Wiener to Bush, dated September 20 and 23, 1940. Norbert Wiener Papers, Box 4, ff. 58.
[31] Letters by Bush to Wiener, dated September 24 and 25, 1940. Norbert Wiener Papers, Box 4, ff. 58.
[32] Letters by Bush to Wiener, dated October 7 and 19, and December 31, 1940. Norbert Wiener Papers, Box 4, ff 58.
[33] Letter by Bush to Wiener, dated December 31, 1940. Norbert Wiener Papers, Box 4, ff 58.

> information on a priori possible types of behavior which we find most useful in discussing the normal action and the disorders of the nervous system.[34]

Perhaps because McCulloch was friends with Rosenblueth, McCulloch found out about Wiener's work on control engineering and his interest in biology. Rosenblueth arranged a meeting between McCulloch and Wiener, which according to McCulloch occurred during the spring of 1940 or 1941. As McCulloch recalled it, he was:

> … amazed at Norbert [Wiener]'s exact knowledge, pointed questions and clear thinking in neurophysiology. He talked also about various kinds of computation and was happy with my notion of brains as, to a first guess, digital computers, with the possibility that it was the temporal succession of impulses that might constitute the signal proper.[35]

In using the term "digital computers," here McCulloch was being a bit anachronistic. In 1941 there were no modern digital computers in operation, and the term "digital computer" had not been used yet. Nevertheless, it is likely that McCulloch explained to Wiener his ideas about information flow through ranks of neurons in accordance with Boolean algebra.[36]

Due to his expertise in computing mechanisms, in 1941 Wiener was appointed as a consultant to the government on machine computation. He began a strenuous research program on fire control for antiaircraft artillery with a young Research Associate at MIT, Julian Bigelow.[37] Bigelow had graduated in electrical engineering from MIT in 1939, and then worked as an electronics engineer at IBM's Endicott Laboratories in 1939-1941. Wiener developed a mathematical theory for predicting the curved flight paths of aircraft, and Bigelow designed and built a machine that performed the necessary computations.[38]

Wiener and Bigelow's work on mechanical prediction was classified, but they were allowed to communicate with other researchers interested in mechanical computation. In October 1941, Wiener and

---

[34] Letter by Wiener to Haldane, dated June 22 1942. Norbert Wiener Papers, Box 4 ff 62.
[35] McCulloch 1974, pp. 38-39.
[36] This may be the source of Wiener's early comparison between digital computing mechanisms and brains, which Wiener attributes to himself—without crediting McCulloch—in Wiener 1958. If Wiener's meeting with McCulloch occurred before September 1940, and if Wiener is correct in attributing his September 1940 proposal of a digital computer to an attempt to imitate the brain's method of calculation, then McCulloch's theory partially inspired one of the first designs of a digital computer. Cf. Aspray 1985, p. 125.
[37] Letter by Wiener to J. Robert Kline, dated April 10, 1941. Norbert Wiener Papers, Box 4, ff. 59.
[38] Julian Bigelow's Vita, undated, Papers of John von Neumann, Box 2, ff. 7.

Bigelow received a visit by John Atanasoff, the designer of the ABC computer.[39] The ABC computer, whose construction began in January 1941, was completed in May 1942 and was the first machine to perform computations using electronic equipment.[40] One day in 1942, Wiener and Bigelow "had a lively discussion on the stairs in the math building at Harvard" with Howard Aiken, another leader in mechanical computing.[41] Aiken was working on a giant electro-mechanical digital computer, the Harvard Mark 1, which was completed in 1944.

According to Wiener's recollection, Bigelow convinced Wiener of the importance of feedback in control mechanisms like those they were designing, and that all that mattered to automatic control of mechanisms was not any particular physical quantity (such as energy or length or voltage) but only "information" (used in an intuitive sense), conveyed by any means.[42] Wiener seized on the importance of feedback and information in control mechanisms, and turned it into a cornerstone of his thinking about the most complex of all control mechanisms—the nervous system.

Wiener merged his work with Bigelow on feedback and information in control mechanisms with the project on possible behaviors that he was carrying out with Rosenblueth. Rosenblueth, Wiener, and Bigelow jointly wrote a famous paper on "teleological" mechanisms (Rosenblueth, Wiener, and Biegelow 1943). The paper classified behaviors as follows. First, it distinguished between purposeful (goal-seeking) and non-purposeful behavior. Then, it divided purposeful behavior into teleological (involving negative feedback) and non-teleological behavior. Teleological behavior, in turn, was divided into predictive and non-predictive behavior. The authors argued that by taxonomizing behaviors in this way, it was possible to study organisms and machines in the same behavioristic way, i.e. by looking at the correlation between their inputs and outputs without worrying about their internal structure. These ideas, and especially the role played by feedback in accounting for teleological behavior, would soon attract a lot of attention. Their impact started to be felt before the paper was published.

---

[39] Letter by Warren Weaver to Atanasoff, dated October 15, 1941. Norbert Wiener Papers, Box 4, ff. 61
[40] On Atanasoff's computer and its relation to later electronic computers, see Burks 2002.
[41] Letter by Bigelow to Wiener, dated 7 August 1944, Norbert Wiener Papers, Box 4 ff. 66.
[42] Wiener 1948, Introduction; see also McCulloch 1974, p. 39.

In April 1942, McCulloch was invited to a conference on Cerebral Inhibition, sponsored by the Josiah Macy, Jr. Foundation.[43] The meeting was organized by Frank Fremont-Smith, M.D., who was Director of the Foundation's Medical Division. The discussion at the meeting was to focus on conditioned reflexes in animals and hypnotic phenomena in humans, both of which were believed to be related to cerebral inhibition. "It is hoped that by focussing the discussion upon physiological mechanisms underlying the two groups of phenomena," wrote Fremont-Smith, "gaps in our knowledge, as well as correlations, may be more clearly indicated."[44] The format of the conference, which took place in New York City on May 14th and 15th 1942, included two formal presentations on conditioned reflexes followed by two days of informal discussion among the participants. One of those participants was Rosenblueth. During the meeting, Rosenblueth talked about the ideas he had developed with Wiener and Bigelow about teleological mechanisms. According to McCulloch, Rosenblueth "implanted the feedback explanation of purposive acts in [Fremont-Smith]'s mind."[45]

Fremont-Smith hoped to get the same group to meet again,[46] and McCulloch saw this as an opportunity to foster his ideas about mind and brain.[47] As soon as McCulloch found the time, he wrote a long letter to Fremont-Smith, in which he indicated:

> … what I hope we can chew over at great length when we are next together, for they are points which I feel are very important to the understanding of the problems confronting us; and I feel sure that your procedure in keeping the group together, discussing long enough to get through the words to the ideas, is the most profitable form of scientific investigation of such problems.

In his letter, McCulloch outlined his views about neural explanations of mental phenomena, including his use of symbolic logic to model neural activity, and endorsed Rosenblueth's point about "the dependence of 'goal directed' behavior upon 'feed-back' mechanisms."[48]

McCulloch's connection with Fremont-Smith soon turned into a friendship.[49] Their relationship would bear fruits in a few years, under the guise of several grants by the Macy Foundations to

[43] Letter by Fremont-Smith to McCulloch, dated April 27, 1942. Warren S. McCulloch Papers, ff. Fremont-Smith.
[44] Memorandum by Fremont-Smith, dated May 11, 1942. Warren S. McCulloch Papers, ff. Fremont-Smith.
[45] Letter by McCulloch to Rosenblueth, dated February 14, 1946. Warren S. McCulloch Papers, ff. Rosenblueth.
[46] Letter by Fremont-Smith to McCulloch, dated April 27, 1942. Warren S. McCulloch Papers, ff. Fremont-Smith.
[47] McCulloch 1974, p. 39.
[48] Letter by McCulloch to Fremont-Smith, dated June 24, 1942. Warren S. McCulloch Papers, ff. Fremont-Smith.

McCulloch's lab as well as what came to be known as the Macy Meetings on cybernetics. By 1942, though, McCulloch was about to publish his long-in-the-work theory of the brain, with help from a new and important character.

3.3 Walter Pitts

Walter Pitts fled his parental home around the age of 15, and never spoke to his family again.[50] In 1938—at the age of 15—he attended a lecture by Bertrand Russell at the University of Chicago. During the lecture, he met an eighteen-year-old fellow in the audience, Jerome Lettvin, who was preparing for medical school by studying biology at the University of Chicago. Pitts and Lettvin became best friends.[51]

According to Lettvin, by the time he met Lettvin, Pitts "had, for a long time, been convinced that the only way of understanding nature was by logic and logic alone."[52] Here is Lettvin's recollection of the origin of that view, as well as a poignant description of Pitts's unique personality:

> Pitts was married to abstract thought. Once, Pitts told us that when he was twelve years old he was chased by some bullies into a public library, where he hid in the stacks. There he picked up Russell and Whitehead's Principia Mathematica and could not put it down. For the next week he lived in the library from opening to closing time, going through all three volumes. It seemed to him then that logic was magic, and if he could master that magic and practice it, the whole world would be in his hegemony—he would be Merlin. But to do this one had to do away with self. Ego must never enter, but only Reason. And at that moment of revelation he committed ontological suicide. That is the peculiar truth about Pitts, whom all of us loved and protected. We never knew anything about his family or his feelings about us. He died mysterious, sad and remote, and not once did I find out, or even want to find out more about how he felt or what he hoped. To be interested in him as a person was to lose him as a friend.[53]

Other witnesses concurred with Lettvin's assessment. People who knew him personally described Pitts as shy, introverted, and socially awkward.[54]

---

[49] At least by August 1943, the two started addressing their letters "Dear Warren" and "Dear Frank" rather than "Dear Doctor McCulloch" and "Dear Doctor Fremont-Smith." Warren S. McCulloch Papers, ff. Fremont-Smith.
[50] Lettvin 1989b, p. 514.
[51] Heims 1991, p. 40; Smalheiser 2000, p. 219. Letter by Lettvin to Wiener, dated ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70. Accounts of Pitts's life contain fictionalized stories, apparently propagated by McCulloch. Smalheiser gives a nice summary of Pitt's life, work and personality. The most reliable source on Pitts seems to be his "life-long friend" Lettvin (Lettvin 1989a, 1989b)
[52] Lettvin 1989a, p. 12.
[53] Lettvin 1989b, p. 515.
[54] Smalheiser 2000, p. 220-221.

Nevertheless, there is consensus that Pitts became knowledgeable and brilliant. The "magic" he performed at twelve with the *Principia Mathematica* may have worked, because Lettvin continued his story as follows:

> [I]f a question were asked about anything whatever—history, literature, mathematics, language, any subject at all, even empirics such as systematic botany or anatomy, out would come an astonishing torrent, not of disconnected bits and pieces of knowledge, but an integral whole, a corpus, an organized handbook with footnotes and index. He was the very embodiment of mind, and could out-think and out-analyze all the rest of us.[55]

In the late 1930s, Pitts started auditing classes at the University of Chicago, without enrolling as a student. He studied logic with Carnap and biophysics with Nicolas Rashevsky.[56] Rashevsky was a Russian physicist who had established the Committee on Mathematical Biology, a pioneering research group in biophysics that included Frank Offner, Herbert Landahl, and Alston Householder. Rashevsky advocated the development of mathematical models of idealized biological processes, applying to biology the methodology of theoretical physics (Rashevsky 1936, 1937, 1938).[57] One area on which Rashevsky and his group worked was the nervous system.

Pitts became a member of Rashevsky's group, quickly starting to do original research without ever earning a degree. In the early 1940s, Pitts published several papers on neural networks in Rashevsky's journal, the *Bulletin of Mathematical Biophysics* (Pitts 1942a, 1942b, 1943). According to Lettvin, it was during this time, namely before meeting McCulloch, that Pitts developed the view that the brain is a "logical machine."[58]

---

[55] Lettvin 1989b, p. 515.

[56] Glymour notes that Carnap's role in the history of computationalism includes his teaching both Pitts and another pioneer, Herbert Simon (Glymour 1990). To them, I should add another student of Carnap that was going to play an important role in the history of computationalism, namely Ray Solomonoff. Unfortunately, a search through the Rudolf Carnap Collection at the Archives of Scientific Philosophy, University of Pittsburgh, uncovered no information on Carnap's relationship to Pitts, Simon, or Solomonoff. Aside from Carnap's teaching these individuals (and inspiring Solomonoff's early work), I have found no evidence of a significant impact by Carnap or his work on the main founders of computationalism.

[57] On Rashevky and his group, see Abraham 2001b; Abraham 2002, pp. 13-18.

[58] Lettvin interview, in Anderson and Rosenfeld 1998, p. 3. Lettvin also put it this way (using slightly anachronistic terminology):

> Quite independently, McCulloch and Pitts set about looking at the nervous system itself as a logical machine in the sense that if, indeed, one could take the firings of a nerve fiber as digital encoding of information, then the operation of nerve fibers on each other could be looked at in an arithmetical sense as a computer for combining and transforming sensory information (Lettvin 1989a, p. 10).

3.4 McCulloch Meets Pitts

In the fall of 1941, McCulloch moved to the University of Illinois in Chicago. He was hired by the Department of Psychiatry to build up a team of specialists and study the biological foundations of mental diseases.[59] At the University of Illinois, the lab's electrical engineer was Craig Goodwin, who knew the theory and design of control devices. Goodwin introduced McCulloch to this area of research, which included topics like automatic volume controls and self-tuning devices. According to McCulloch, he learned from Goodwin that when the mathematics of the hardware, e.g. coupled nonlinear oscillators, was intractable, i.e. when the equations representing the system could not be solved analytically, they could still build a working model and use it to think about the problem.[60]

In 1939, Lettvin started medical school at the University of Illinois, where his anatomy teacher was Gerhard von Bonin. After McCulloch moved to Chicago in 1941, von Bonin introduced Lettvin to McCulloch.[61] McCulloch, who once called Lettvin "the brightest medical student I have ever known," exerted a strong influence on Lettvin, and later convinced him to do research on the brain.[62] Once in Chicago, McCulloch also made contact with Rashevsky's group. He started attending the group's seminar, where Lettvin introduced him to then almost eighteen-year-old Pitts.[63] Like Carnap and Rashevsky before him, McCulloch was "much impressed" by Pitts.[64]

When McCulloch presented his ideas about information flow through ranks of neurons to Rashevsky's seminar, Pitts was in the audience. Pitts showed interest in a problem that McCulloch had

[59] McCulloch 1974, p. 35.
[60] McCulloch 1974, p. 35. This may be significant in light of McCulloch's later ideas about building mechanical models of the brain.
[61] Lettvin 1989b, p. 514.
[62] Letter by McCulloch to Henry Moe, dated December 30, 1959. Warren S. McCulloch Papers, ff. Gerard. Letter by Lettvin to Wiener, dated ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70.
[63] Lettvin 1989b, p. 515.
[64] McCulloch 1974, pp. 35-36.

struggled with: the problem of how to give mathematical treatment to regenerative nervous activity in closed neural loops.[65]

McCulloch hypothesized that closed neural loops explained neural processes that, once started, continued on by themselves. Initially, McCulloch was thinking about pathologies, such as the neural activity of epileptic patients and other pathological conditions, including phantom limbs, compulsive behavior, anxiety, and the effects of shock therapy. But Lawrence Kubie had postulated closed loops of activity to explain memory (Kubie 1930), and Lorente de Nó had shown the significance of closed loops in vestibular nistagmus (Lorente de Nó 1938). This convinced McCulloch that closed loops of activity could fulfill positive neural functions. By the time McCulloch met Pitts, McCulloch thought closed loops could account for memory and conditioning, but he still didn't know how to think mathematically about them.[66]

McCulloch and Pitts started working together; they worked so closely that Pitts (as well as Lettvin) moved in with McCulloch and his family for about a year in Chicago. McCulloch and Pitts became intimate friends and they remained so until their death in 1969.[67] For two years, they worked largely on the problem of how to treat closed loops of activity mathematically. According to McCulloch, the solution was worked out mostly by Pitts using techniques that McCulloch didn't understand. To build up their formal theory, they adopted what they saw as Carnap's rigorous terminology, which Pitts knew from having studied with Carnap. Thus, according to McCulloch, Pitts did all the difficult technical work.[68] The resulting paper was published in Rashevsky's journal in 1943, with a brief follow-up written by McCulloch and Pitts with Herbert Landahl, another member of Rashevsky's group, on a statistical application of the theory.

---

[65] McCulloch 1974, pp. 35-36.
[66] McCulloch 1974, p. 36.
[67] Shortly before both of them died, Pitts wrote McCulloch from his hospital bed, commenting in detail on their conditions and expressing the wish that they meet again and talk about philosophy. Letter by Pitts to McCulloch, dated April 21, 1969. Warren S. McCulloch Papers, ff. Pitts.
[68] McCulloch 1974, p. 36.

4 BRAINS COMPUTE THOUGHTS, 1943

> One would assume, I think, that the presence of a theory, however strange, in a field in which no theory had previously existed, would have been a spur to the imagination of neurobiologists… But this did not occur at all! The whole field of neurology and neurobiology ignored the structure, the message, and the form of McCulloch's and Pitts's theory. Instead, those who were inspired by it were those who were destined to become the aficionados of a new venture, now called Artificial Intelligence, which proposed to realize in a programmatic way the ideas generated by the theory (Lettvin 1989a, p. 17).

4.1 A Mechanistic Theory of Mind

McCulloch believed that the goal of neurophysiology and psychiatry was to explain the mind, and that scientists had not seriously tried to construct a neural theory to this effect. A serious obstacle was what philosophers called the mind-body problem. In a commentary to a paper presented in May 1943 at the Illinois Psychiatric Society, McCulloch explained:

> We have a dichotomy in medicine, which has grown increasingly… Psychiatric approach on one side, particularly the psychoanalytic approach, has produced one group; the organic approach to the physiology of particular organs and disease processes has made organicists of another group. It has grown difficult for us to talk to each other. I am afraid that there is still in the minds of most of us, and that there probably will be for years, that difficulty which concerned and still concerns many thinking people—I mean the dichotomy between mind and body.[1]

McCulloch continued his commentary by saying that there were "two types of terminology": "mental terms" were used to describe "psychological processes, for these exhibit ideas and intentions"; "physical terms" were used to describe "bodily processes, for these exhibit matter and energy." But:

> … it remains our great difficulty that we have not ever managed to conceive how our patient—our monad—can have a psychological aspect and a physiological aspect so divorced. You may think that I am exaggerating the difficulty here, but there have appeared within the last few years two books which tilt at the same windmill. One is Sherrington, called "Man and His Nature," and in it Sherrington, the marvelously honest physiologist, attempts to make head and tail of the mind-body relation, but is frustrated because in that world "Mind goes more ghostly than a ghost." The other book, by Wolfgang Koehler (the founder of Gestalt psychology), is entitled "The Place of Value in a World of Fact," but in spite of his endless searching, you will be convinced that he has not found the place of value in the world of fact. Such was the unsatisfactory state of our theory until very recently.[2]

[1] Discussion by Dr. McCulloch of a paper by Dr. Alexander on Fundamental Concepts of Psychosomatic Research, Illinois Psychiatric Society, dated May 22, 1943. Warren S. McCulloch Papers.
[2] Ibid. The works referenced by McCulloch are Sherrington 1940 and Köhler 1938.

After thus stating the mind-body problem, McCulloch pointed at two recent developments that gave hope for its solution.

As an answer to the question of "the place of values in a world of fact," McCulloch cited the newly published work of Rosenblueth, Wiener, and Bigelow (1943), which used the notion of feedback to account for teleological behavior. As to what McCulloch called the "formal" aspect of mind, he promised he was going to have something to contribute soon:

> At the present time the other mental aspect of behavior—I mean its ideational or rational, formal or logical aspect—is coming to the fore. This work … should be coming to fruition in the next year or two… We do resent the existing hiatus between our mental terminology and our physical terminology. It is being attacked in a very realistic fashion today. So while we do at the moment think of it as a "leap from psyche to soma," we are busy bridging the gap between mental processes and physical processes. To this audience it is interesting that that bridge is being made by demonstrating that the properties of systems which are like our nervous system necessarily show those aspects of behavior that make us call it "mental"—namely, ideas and purposes.[3]

The explanation for the "formal" aspect of the mind, and hence the solution to that component of the mind-body problem, was about to be offered by McCulloch in the paper he was writing with Walter Pitts. Their way of solving the problem was to demonstrate how a system of neuron-like elements embodied ideas.

In a letter by McCulloch to Frank Fremont-Smith, written a few months before the commentary cited above, McCulloch was more detailed and explicit as to what he hoped to accomplish with his theory and the role that logic played in it:

> As to the "formal" properties [of the mind], it is perfectly possible today (basing the work on the all-or-none law and the requirement of summation at a synapse and of inhibition either at a synapse or by preoccupation of a requisite pool of internuncials) to show that neuronal reactions are related to antecedent neuronal reactions—I mean reactions in parts of the nervous system afferent to the reaction in question—in a manner best schematized by symbolic logic; in brief, that the efferent impulses are related to the afferent impulses as logical consequences are related to logical antecedents, and hence that classes of the latter are so related to classes of the former.
>
> Little consideration is necessary to show that neuronal and all other reactions which derive their energy metabolically and are triggered off by something else, being reactions of the zero order with respect to what initiates them, bear to their precipitating causes the same relation that propositions do to that which they propose. If then, from the sense organ forward, the reaction of subsequent neurones is dependent upon any selection from the totality of energy delivered to the system, the response corresponds to an abstraction from that totality, so that

[3] Ibid.

> neural behavior is not only essentially propositional but abstract with respect to its precipitating cause.[4]

Once again, McCulloch was describing the work he was pursuing with Pitts. The all-or-none law of neural activity—namely that the effects of neural activity depended only on the *number* of nerve impulses traveling through the nervous system—allowed McCulloch and Pitts to use symbolic logic to describe neural activity, so that inferential relations among propositions described causal streams of neural events. This, for McCulloch, was enough to show that "neural behavior is essentially propositional" in a way that explained mechanistically the "formal" aspect of the mind.

The sense in which neural behavior was essentially propositional was further clarified by McCulloch in a letter to a neurophysiologist at the University of Chicago, Ralph Lillie. In February 1943, he explained how "we might be able to see mechanistically the problem of ideas":

> [W]hat was in my mind was this: that neuronal activity bore to the world external to the organism the relationship that a proposition bears to that to which it proposes. In this sense, neuronal activity so reflects the external world as to account for that all-or-none characteristic of our logic (and of our knowledge) which has been one of the greatest stumbling blocks to epistemology. I think that for the first time we are in a position to regard scientific theory as the natural consequence of the neuronal activity of an organism (here the scientist)… And this has come about because the observed regularity—all-or-none of neurones, bears a one-to-one correspondence to those peculiar hypothetical psychic atoms called psychons which preserve in the unity of their occurrence both the all-or-none law and the property of reference characteristic of propositions.[5]

Thanks to the all-or-none law, neural events stood in "one-to-one correspondence" to psychons, and just like psychons and propositions, neuronal activity had "the property of reference."

To solve the mind-body problem, McCulloch and Pitts formulated what they called a "logical calculus of the ideas immanent in nervous activity" (McCulloch and Pitts 1943). As Frederic Fitch pointed out in reviewing the paper for the *Journal of Symbolic Logic*, this was not quite a logical calculus in the sense employed by logicians.[6]

A common misconception is that McCulloch and Pitts demonstrated that neural nets can compute anything that Turing Machines can:

---

[4] Letter by McCulloch to Fremont-Smith, dated June 24, 1942. Warren S. McCulloch Papers, ff. Fremont-Smith.
[5] Letter by McCulloch to Ralph Lillie, ca. February 1943. Warren S. McCulloch Papers, ff. Lillie.
[6] Fitch 1944.

> McCulloch and Pitts proved that a sufficiently large number of these simple logical devices, wired together in an appropriate manner, are capable of universal computation. That is, a network of such 'lineal threshold' units with the appropriate synaptic weights can perform any computation that a digital computer can, though not as rapidly or as conveniently.[7]

As we shall see, this is incorrect in two respects. First, McCulloch and Pitts did not *prove* any results about what their nets can compute, although they claimed that there were results to prove; second, McCulloch-Pitts nets—as the McCulloch and Pitts recognized—are computationally less powerful than Turing Machines.

The computation power of McCulloch-Pitts nets is only one among many issues raised by their theory. Although this paper is cited often, it has received little careful attention. The great historical importance of this paper, and its common misrepresentation, warrant that we study it closely. The rest of this chapter is devoted to it.

4.2 Motivation

The paper started with the rehearsal of some established neurophysiological facts: the nervous system was a network of neurons connected through synapses; neurons sent to each other excitatory and inhibitory pulses[8]; and each neuron had a threshold determining how many excitatory and inhibitory inputs are necessary and sufficient to excite it at a given time.[9]

Then, the authors introduced the main premise of their theory: the identification of neuronal signals with propositions. This was presumably what justified their title, which mentioned a calculus of *ideas immanent in nervous activity*. They introduced this identification in a curious and rather obscure

[7] Koch and Segev 2000, p. 1171.
[8] According to Lettvin, an important source of the logic gate model of the neuron was the recent discovery by David Lloyd of direct excitation and inhibition between single neurons: "it was not until David Lloyd's work in 1939-41 that the direct monosynaptic inhibitory and excitatory actions of nervous pulses were demonstrated. This finding, more than anything else, led Warren and Walter to conceive of single neurons as doing logical operations (a la Leibnitz and Boole) and acting as gates" (Lettvin's 1988, Foreward to the second edition of *Embodiments of Mind*, cited by Heims 1991, pp. 233-234). In light of McCulloch's professions of belief in his logical conception of the nervous system since the early 1930s, it is unclear how crucial Lloyd's work was in motivating McCulloch and Pitts, besides providing experimental validation of some of their ideas.
[9] McCulloch and Pitts 1943, pp. 19-21.

way, appealing not to some explicit motivation but to "considerations," made by one of the authors, which they did not give:

> Many years ago one of us, by *considerations impertinent to this argument*, was led to conceive of the response of any neuron as *factually equivalent* to a proposition which proposed its adequate stimulus. He therefore attempted to record the behavior of complicated nets in the notation of the symbolic logic of propositions. The "all-or-none" law of nervous activity is sufficient to insure that the activity of any neuron may be represented as a proposition. Physiological relations existing among nervous activities correspond, of course, to relations among the propositions; and the utility of the representation depends upon the identity of these relations with those of the logic of propositions. To each reaction of any neuron there is a corresponding assertion of a simple proposition. This, in turn, implies either some other simple proposition or the disjunction or the conjunction, with or without negation, of similar propositions, according to the configuration of the synapses upon and the threshold of the neuron in question.[10]

In light of what was said in Chapters 2 and 3, the author of the "considerations" was McCulloch, and the considerations were those that led him to formulate first his theory of psychons, and then his theory of information flow through ranks of neurons. A proposition that "proposes a neuron's adequate stimulus" was a proposition that said that the neuron received a certain input at a certain time. The authors did not explain what they meant by "factual equivalence" between neuronal pulses and propositions, but their language suggested they meant both that neuronal pulses were represented by propositions, and that neuronal pulses had propositional content.

The theory was divided into two parts: one dealing with nets without closed loops of neural activity, which in this paper are referred to as "circles," the other dealing with nets with circles (*cyclic* nets). The authors pointed out that the nervous system contains many circular, "regenerative" paths.[11] The term "circle" may have been borrowed from Turing (1936-7), who had used "machines with circles" for Turing Machines whose computations never halt, and "machines without circles" for Turing Machines whose computations eventually halt.

4.3 Assumptions

In formulating the theory, McCulloch and Pitts made the following five assumptions:

[10] Ibid., p. 21; emphasis added.
[11] Ibid., p. 22.

1. The activity of the neuron is an "all-or-none" process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
3. The only significant delay within the nervous system is synaptic delay.
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
5. The structure of the net does not change with time.[12]

These assumptions constituted an idealization of the known properties of neural nets. Assumption (1) was simply the all-or-none law: neurons were believed to either pulse or be at rest. As to (2), it is not strictly true but in many cases it was considered a good approximation. As to (3), this was probably the least explicit and least physiologically justified assumption of the theory. Under the heading of "synaptic delay," McCulloch and Pitts assumed that the timing of the activity of neural nets was uniformly discrete, such that any neural event in a neural net occurred within one time interval of fixed duration. This assumption had the effect of discretizing the continuous temporal dynamics of the net, so that logical functions of discrete states could be used to describe the transitions between neural events. As to (4) and (5), McCulloch and Pitts admitted that they are false of the nervous system. However, under the other assumptions, they showed that nets that do not satisfy (4) and (5) are functionally equivalent to nets that do.[13]

McCulloch and Pitts were perfectly aware that the neuron-like elements in their theory were quite distant from real neurons: "Formal neurons were deliberately as impoverished as possible."[14] In a letter

[12] Ibid., p. 22.
[13] Ibid., pp. 29-30.
[14] McCulloch 1974, p. 36.

written to a colleague asking for clarification after a public presentation of the theory, McCulloch wrote as follows:

> [W]e in our description restricted ourselves to the regular behavior of the nervous system, knowing full well that irregularities can be and are frequently brought about by physical and chemical alterations of the nervous system. As a psychiatrist, I am perhaps more interested in these than in its regular activity, but they lead rather to a theory of error than a theory of knowledge, and hence were systematically excluded from the description.[15]

In McCulloch's eyes, the differences between real neurons and the elements employed in his theory were inessential. His goal was not to understand neural mechanisms *per se*, but rather to explain how something close enough to a neural mechanism could exhibit "knowledge," the kind of "ideational," "rational," "formal," or "logical" aspect that was associated with the mind. McCulloch's goal was to offer, for the first time, an explanation of the mind in terms of neural-like mechanisms.

4.4 Nets Without Circles

McCulloch and Pitts's technical language was cumbersome; here their theory is given in a slightly streamlined form that makes it easier to follow. The neurons of a net N are denoted by c_1, c_2, … c_n. A primitive expression of the form $N_i(t)$ means that neuron c_i fires at time t. Expressions of the form $N_i(t)$ can be combined by means of logical connectives to form complex expressions that describe the behavior of different neurons at certain times. For example, $N_1(t)\&N_2(t)$ means that neurons c_1 and c_2 fire at time t, $N_1(t\text{-}1)\vee N_2(t\text{-}2)$ means that either c_1 fires at t-1 or c_2 fires at t-2 (or both), etc. These complex expressions can in turn be combined by the same logical connectives. As well-formed combinations, McCulloch and Pitts allowed only the use of conjunction (A&B), disjunction (A∨B), conjunction and negation (A&~B), and a special connective S that shifts the temporal index of an expression backwards in time, so that $S(N_i(t)) = N_i(t\text{-}1)$. A complex expression formed from a number of primitive expressions $N_1(t)$, … $N_n(t)$

[15] Letter by McCulloch to Ralph Lillie, ca. February 1943. Warren S. McCulloch Papers, ff. Lillie.
Cf. also Lettvin:

> The *Logical Calculus*, McCulloch knew, was not even a caricature of any existing nervous process. Indeed he made that very clear at the time of writing. But is [*sic*] was a possible and useful assembly of axiomatized neurons, and that seemed to him a far greater accomplishment than a true description of any definitely known neuronal circuit (of which none then existed) (Lettvin 1989b, p. 518).

by means of the above connectives is denoted by $Exp_j(N_1(t), \ldots N_n(t))$. In any net without circles, there are some neurons with no axons inputting on them; these are called *afferent* neurons.

The two main technical problems McCulloch and Pitts wanted to solve were "to calculate the behavior of any net, and to find a net which will behave in a specified way, when such a net exists."[16] In terms of the theory, the problems can be formulated as follows:

First problem: given a net, find a class of expressions **C** such that for every neuron c_i, in **C** there is a true expression of the form

$N_i(t)$ if and only if $Exp_j((N_{i-g}(t-1), \ldots N_{i-2}(t-1), N_{i-1}(t-1))$,

where neurons c_{i-g}, … c_{i-2}, and c_{i-1} have axons inputting c_i.

The significance of this expression is that it describes the behavior of any (non-afferent) neuron in terms of the behavior of the neurons that are afferent to it. If a class **C** of such expressions is found, propositional logic can describe the behavior of any non-afferent neuron in the net in terms of the behavior of the neurons afferent to it.

Second problem: given an expression of the form

$N_i(t)$ if and only if $Exp_j((N_{i-g}(t-1), \ldots N_{i-2}(t-1), N_{i-1}(t-1))$,

find a net for which it is true.

McCulloch and Pitts showed that these problems were easily solved. To solve the first problem, they showed how to write an expression describing the relation between the firing of any neuron in a net and the inputs it receives from its afferent neurons. To solve the second problem, they showed how to construct nets that satisfy their four combinatorial schemes (conjunction, disjunction, conjunction-cum-negation, and temporal predecessor), giving diagrams that show the connections between neurons that satisfy each scheme (figure 4-1). Then, by induction on the size of the nets, all expressions formed by those combinatorial schemes are realizable by McCulloch-Pitts nets.[17]

[16] Ibid., p. 24.

[17] Their actual proof was not quite a mathematical induction because they didn't show how to combine nets of arbitrary size, but the technical details are unimportant here.

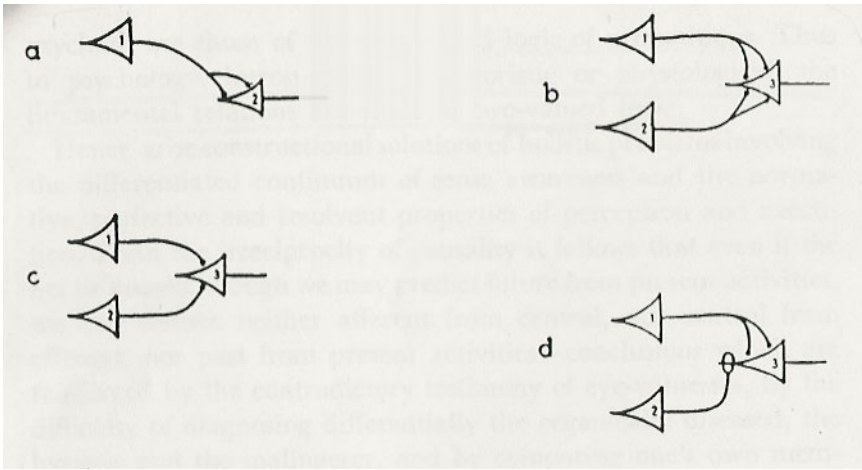

Figure 4-1. Diagrams of McCulloch and Pitts nets.

By giving diagrams of nets that satisfy simple logical relations between propositions and by showing how to combine them to satisfy more complex logical propositions, McCulloch and Pitts developed a powerful technique for designing circuits that satisfy given logical functions by using a few primitive building blocks.[18]

McCulloch and Pitts's goal was to explain mental phenomena. As an example, they offered an explanation of a well-known heat illusion by constructing an appropriate net. A cold object touching the skin normally causes a sensation of cold, but if it is held for a very brief time and then removed, it can cause a sensation of heat. In designing their net, McCulloch and Pitts reasoned as follows. They started from the known physiological fact that there are different kinds of receptors affected by heat and cold, and they assumed that there are neurons whose activity "implies a sensation" of heat.[19] Then, they assigned one neuron to each function: heat reception, cold reception, heat sensation, and cold sensation. Finally, they observed that the heat illusion corresponded to the following relations between three

[18] This is the main aspect of their theory used by von Neumann in describing the design of digital computers (see the next chapter). Today, McCulloch and Pitts's technique is part of logic design, an important area of computer design devoted to designing digital circuits for digital computers. The building blocks of contemporary logic design are called logic gates. In modern terminology, McCulloch and Pitts's nets are logic gates and combinations of logic gates. For more on logic and computer design, see Chapter 10.
[19] Ibid., p. 27.

neurons: the heat-sensation neuron fires either in response to the heat receptor or to a brief activity of the cold receptor (figure 4-2).

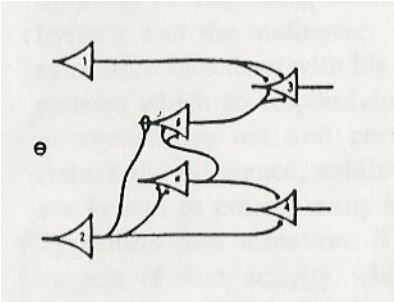

Figure 4-2. Net explaining heat illusion. Neuron 3 (heat sensation) fires if and only if it receives two inputs, represented by the lines terminating on its body. This happens when either neuron 1 (heat reception) fires or neuron 2 (cold sensation) fires once and then immediately stops firing. When neuron 2 fires twice in a raw, the intermediate (unnumbered) neurons excite neuron 4 rather than neuron 3, generating a sensation of cold.

McCulloch and Pitts used this example for a general observation about the relation between perception and the world:

> This illusion makes very clear the dependence of the correspondence between perception and the "external world" upon the specific structural properties of the intervening nervous net.[20]

Then, they pointed out that, under other assumptions about the behavior of the heat and cold receptors, the same illusion could be explained by different nets (ibid., p. 28).

4.5 Nets With Circles, Computation, and the Church-Turing Thesis

The problems for nets with circles are analogous to those for nets without circles: given the behavior of a neuron's afferents, find a description of the behavior of the neuron; and find the class of expressions and a method of construction such that for any expression in the class, a net can be constructed that satisfies the expression. The authors pointed out that the theory of nets with circles is more difficult than the theory of nets without circles. This is because activity around a circle of neurons can continue for an indefinite amount of time, hence expressions of the form $N_i(t)$ may have to refer to times that are indefinitely remote

[20] Ibid., p. 28.

in the past. For this reason, the expressions describing nets with circles are more complicated, involving quantification over times. McCulloch and Pitts offered solutions to the problems of nets with circles, but their treatment of this part of the theory was very obscure, admittedly sketchy,[21] and contained some errors that make it hard to follow.[22]

At the end of this section, McCulloch and Pitts drew the connection between their nets and computation:

> It is easily shown: first, that every net, if furnished with a tape, scanners connected to afferents, and suitable efferents to perform the necessary motor-operations, can compute only such numbers as can a Turing machine; second, that each of the latter numbers can be computed by such a net; and that nets with circles can be computed by such a net; and that nets with circles can compute, without scanners and a tape, some of the numbers the machine can, but no others, and not all of them. This is of interest as affording a psychological justification of the Turing definition of computability and its equivalents, Church's λ-definability and Kleene's primitive recursiveness: If any number can be computed by an organism, it is computable by these definitions, and conversely.[23]

This brief passage is the only one mentioning computation. By stating that McCulloch-Pitts nets compute, this passage provided the first known published link between computation and brain theory. It was a pivotal statement in the history of computationalism.

It is often said that McCulloch and Pitts proved that their nets could compute anything that Turing Machines can compute (e.g., Koch and Segev 2000). This misconception was initiated and propagated by McCulloch himself. For instance, in summarizing the significance of their paper, McCulloch wrote to a colleague:

> [T]he original paper with Pitts entitled "A Logical Calculus of Ideas Immanent in Nervous Activity" … sets up a calculus of propositions subscripted for the time of their appearance for any net handling all-or-none signals, and shows that such nets can compute any computable number or, for that matter, do anything any other net can do by the way of pulling consequences out of premises.[24]

---

[21] Ibid., p. 34.

[22] Every commentator points this out, starting with Fitch 1944, p. 51. See also Arbib 1989. McCulloch and Pitts's theoretical treatment of their nets was soon superseded by Kleene's, who explicitly describes his task as that of sorting out the obscure claims made by McCulloch and Pitts (Kleene 1956).

[23] Ibid., p. 35. This reference to "a Turing machine" and to "the Turing definition of computability" proves that both McCulloch and Pitts knew of Turing's 1936-7 work. Lettvin was thus mistaken in stating that at the time they wrote their 1943 paper, "neither Warren [McCulloch] nor Walter [Pitts] knew of Turing's paper of 1937" (Lettvin 1989a, p. 515).

[24] Letter by McCulloch to Schouten, dated October 18, 1948. Warren S. McCulloch Papers.

But in discussing computation in their paper, McCulloch and Pitts did not *prove* any results about the computation power of their nets; they only stated that there were results to prove. And their conjecture was not that their nets could compute anything that can be computed by Turing Machines. Rather, they claimed that *if* their nets were provided with a tape, scanners, and suitable efferents, *then* they would compute what Turing Machines could compute; without a tape, McCulloch and Pitts expected even nets with circles to compute a smaller class of functions than the class computable by Turing Machines. In spite of their comment that these results were "easily shown," the precise characterization of the class of functions computable by McCulloch-Pitts nets (without tapes) required some heavy-duty mathematical work. The problem was solved thirteen years later by Stephen Kleene (1956), who invented the formalism of finite state automata and showed that McCulloch-Pitts nets are computationally equivalent to finite state automata.

McCulloch and Pitts did not explain what they meant by saying that nets computed. As far as the first part of the passage is concerned, the sense in which nets computed seemed to be a matter of describing the behavior of nets by the vocabulary of computability theory, which allows McCulloch-Pitts nets to be seen as a formalism for designing circuits for computing mechanisms. This is how von Neumann would later use them. If this was all there was to it, which functions were computed by McCulloch-Pitts nets was an innocent technical question devoid of epistemological significance.

But one of the most interesting aspects of the above passage about nets and computation is the way it related McCulloch-Pitts nets to the Church-Turing thesis (CT). The passage says that the fact that McCulloch-Pitts nets computed certain computable functions provided a psychological justification for CT. It did not explain why CT should be given a psychological justification, or in what sense it can be given one. As we saw in the previous chapter, CT was the thesis that any effectively calculable function is computable by Turing Machines. Turing justified it by "intuitive considerations," which included the cognitive limitations of computing humans.[25] It seems that McCulloch and Pitts here were attempting to

[25] These intuitive considerations, as well as the contemporary debate on the status of CT, are discussed in Chapter 7.

go beyond appeals to intuition and offering a justification for CT based on the computation power of the nets that constitute the nervous system.

By adding that the fact that nets computed certain functions afforded a psychological justification of CT, McCulloch and Pitts seemed to believe that ordinary human computations using pencil and paper (which is what CT is about), and more generally humans' "pulling of consequences out of premises," could be explained by the computations performed by their nets. Thus McCulloch and Pitts attributed the fact that neurons compute epistemological significance.

Indeed, the main purpose of the theory was to explain mental functions, such as computation and inference, by possible neural-like mechanisms. As McCulloch explained a few years later, he and Pitts were interpreting neural inputs and outputs as if they were symbols written on the tape of a Turing Machine:

> What we thought we were doing (and I thought we succeeded fairly well) was treating the brain as a Turing machine... The important thing was, for us, that we had to take a logic and subscript it for the time of occurrence of a signal (which is, if you will, no more than a proposition on the move). This was needed in order to construct theory enough to be able to state how a nervous system could do anything. The delightful thing is that the very simplest set of appropriate assumptions is sufficient to show that a nervous system can compute any computable number. It is that kind of a device, if you like—a Turing machine.[26]

In comparing brains to a Turing Machine, McCulloch thought that by constructing their theory, they were able to show how brains "could do anything," including performing "the kind of [mental] functions which a brain must perform if it is only to go wrong and have a psychosis."[27] "Treating the brain as a Turing machine" was a crucial part of McCulloch and Pitts's attempt at solving the mind-body problem.

4.6 "Consequences"

McCulloch and Pitts ended their paper by drawing what they called "consequences." Their discussion introduced a number of metaphysical and epistemological themes, and deserves to be followed it in its

[26] This statement is attributed to McCulloch in a discussion published in von Neumann 1951, pp. 32-33.
[27] Ibid.

entirety. They started with a general point about the causal structure of their nets, which was such that, from a given event in a net, it may be impossible to infer its exact causal antecedent:

> Causality, which requires description of states and a law of necessary connections relating them, has appeared in several forms in several sciences, but never, except in statistics, has it been as irreciprocal as in this theory. Specification for any one time of afferent stimulation and of the activity of all constituent neurons, each an "all-or-none" affair, determines the state. Specification of the nervous net provides the law of necessary connection whereby one can compute from the description of any state that of the succeeding state, but the inclusion of disjunctive relations prevents complete determination of the one before. Moreover, the regenerative activity of constituent circles renders reference indefinite as to time past.[28]

From this relatively straightforward observation about the causal structure of McCulloch-Pitts nets, they drew striking epistemological conclusions:

> Thus our knowledge of the world, including ourselves, is incomplete as to space and indefinite as to time. This ignorance, implicit in all our brains, is the counterpart of the abstraction which renders our knowledge useful. The role of brains in determining the epistemic relations of our theories to our observations and of these to the facts is all too clear, for *it is apparent that every idea and every sensation is realized by activity within that net*, and by no such activity are the actual afferents fully determined.[29]

This passage makes it clear that McCulloch and Pitts thought of individual neuronal pulses and their relations as realizations of sensations, ideas, and their epistemic relations. This assumption—which had been introduced in the paper by an oblique reference to "considerations impertinent to this argument"—allowed them to draw conclusions about epistemological limitations of the mind directly from the causal structure of their nets.

The next passage drew further epistemological conclusions, echoing the Kantian theme that the mind could know only phenomena not things in themselves. According to McCulloch and Pitts, changing a net would destroy our knowledge no matter what theory we had or observations we made; but if a network were fixed in place, we could know things in themselves:

> There is no theory we may hold and no observation we can make that will retain so much as its old defective reference to the facts if the net be altered. Tinnitus, paraesthesias, hallucinations, delusions, confusions and disorientations invervene. Thus empiry confirms that if our nets are undefined, our facts are undefined, and to the "real" we can attribute not so much as one quality

[28] McCulloch and Pitts 1943, p. 35.
[29] Ibid., pp. 35-37, emphasis added.

> or "form." With determination of the net, the unknowable object of knowledge, the "thing in itself," ceases to be unknowable.[30]

After drawing their epistemological consequences, McCulloch and Pitts went on to offer some morals to the psychologists. They started with two points: first, they stated a reductionist doctrine according to which their theory had the resources to reduce psychology to neurophysiology; second, they argued that, because of the all-or-none character of neural activity, the most fundamental relations among psychological events are those of two-valued logic. In making their case about psychology, McCulloch and Pitts stated very explicitly that they interpreted nervous activity as having "intentional character":

> To psychology, however defined, specification of the net would contribute all that could be achieved in that field—even if the analysis were pushed to ultimate psychic units or "psychons," for a psychon can be no less than the activity of a single neuron. Since that activity is inherently propositional, all psychic events have an intentional, or "semiotic," character. The "all-or-none" law of these activities, and the conformity of their relations to those of the logic of propositions, insure that the relations of psychons are those of the two-valued logic of propositions. Thus in psychology, introspective, behavioristic or physiological, the fundamental relations are those of two-valued logic.[31]

The long final paragraph begins with a summary of the "consequences" and a restatement that mental phenomena are now derivable from neurophysiology:

> Hence arise constructional solutions of holistic problems involving the differentiated continuum of sense awareness and the normative, perfective and resolvent properties of perception and execution. From the irreciprocity of causality it follows that even if the net be known, though we may predict future from present activities, we can deduce neither afferent from central, nor central from efferent, nor past from present activities—conclusions which are reinforced by the contradictory testimony of eye-witnesses, by the difficulty of diagnosing differentially the organically diseased, the hysteric and the malingerer, and by comparing one's own memories or recollections with his contemporaneous records. Moreover, systems which so respond to the difference between afferents to a regenerative net and certain activity within that net, as to reduce the difference, exhibit purposive behavior; and organisms are known to possess many such systems, subserving homeostasis, appetition and attention. Thus both the formal and the final aspects of that activity which we are want to call *mental* are rigorously deducible from present neurophysiology.[32]

The same paragraph continues with "consequences" relevant to psychiatry. One is that the causal structure of nets was used to infer that knowing the history of a subject was unnecessary for curing mental illness. A more general one is that mental diseases reduced to properties of neural networks, and even

[30] Ibid., p. 37.
[31] Ibid., pp. 37-38.
[32] Ibid., p. 38.

more generally that the mind-body problem was solved. McCulloch was giving a direct answer to Sherrington, who had written that "mind goes more ghostly than a ghost" (Sherrington 1940):

> The psychiatrist may take comfort from the obvious conclusion concerning causality—that, for prognosis, history is never necessary. He can take little from the equally valid conclusion that his observables are explicable only in terms of nervous activities which, until recently, have been beyond his ken. The crux of this ignorance is that inference from any sample of overt behavior to nervous nets is not unique, whereas, of imaginable nets, only one in fact exists, and may, at any moment, exhibit some unpredictable activity. Certainly for the psychiatrist it is more to the point that in such systems "Mind" no longer "goes more ghostly than a ghost." Instead, diseased mentality can be understood without loss of scope or rigor, in the scientific terms of neurophysiology.[33]

The essay ends with an appeal to neurology and mathematical biophysics:

> For neurology, the theory sharpens the distinction between nets necessary or merely sufficient for given activities, and so clarifies the relations of disturbed structure to disturbed function. In its own domain the difference between equivalent nets and nets equivalent in the narrow sense indicates the appropriate use and importance of temporal studies of nervous activity: and to mathematical biophysics the theory contributes a tool for rigorous symbolic treatment of known nets and an easy method of constructing hypothetical nets of required properties.[34]

The last point, the method of construction of "hypothetical nets of required properties," highlights what was one the most fruitful legacies of the paper. From then on, McCulloch and Pitts, soon followed by generations of researchers, would use the techniques developed in this paper, and modifications thereof, to design neural networks to explain neural and cognitive phenomena.

4.7 The Historical Significance of McCulloch-Pitts Nets

McCulloch and Pitts's paper offered a mathematical technique for designing neural nets to implement certain inferential relations among propositions, and suggested that those inferences were mathematically equivalent to certain numerical computations. They didn't talk about computers yet, because modern computers didn't exist at the time of their article. Nonetheless, their technique could be used in designing circuits for digital computers, because it allowed designing and representing circuits that computed any desired Boolean function. Since circuits computing Boolean functions became the building blocks of

[33] Ibid., p. 38.
[34] Ibid., pp. 38-39.

modern digital computers, McCulloch and Pitts's technique got co-opted by von Neumann (1945) as part of what is now called logic design.

In the 1950s, the question raised by McCulloch and Pitts about what functions were computed by their nets led to the development of finite automata, one of the most important formalisms in the theory of computation.

McCulloch and Pitts's nets were ostensibly "neural" in the sense that the *on* and *off* values of their units were inspired by the all-or-none character of neuronal activity. However, McCulloch-Pitts nets were heavily simplified relative to the then known properties of neurons and neural nets. The theory did not offer many immediately testable predictions or explanations for observable neural phenomena. It was quite removed from what neurophysiologists could do in their labs. This may be why neurophysiologists largely ignored McCulloch and Pitts's theory. Even McCulloch and Pitts, in their later neurophysiological work, did not make direct use of their theory.

McCulloch and Pitt's project was not to systematize and explain observations about the nervous system; it was to explain knowledge and other mental phenomena in terms of mechanisms that resembled neural ones. To do this, they assumed that mental states could be analyzed in terms of mental atoms endowed with propositional content, the psychons, and that the neural correlates of mental phenomena corresponded to precise configurations of neuronal pulses: individual pulses corresponded to individual psychons, and causal relations among pulses corresponded to inferential relations among psychons.

McCulloch and Pitts's theory found a sympathetic audience in scientists interested in epistemology but trained in mathematics or engineering more than in neurophysiology, such as Norbert Wiener and John von Neumann. For one thing, they liked the claim that the mind had been reduced to the brain; today, some of their intellectual heirs still see the solution to the mind-body problem as McCulloch and Pitts's great contribution.[35] For another thing, they liked the operationalist flavor of the theory, whereby the design of nets was seen as all there was to the performance of inferences and more generally

[35] Arbib 2000, p. 212 and 213; Lettvin 1989b, p. 514. McCulloch and Pitts was also important motivation behind what came to be known in philosophy as the functionalist solution to the mind-body problem. That story is told in Chapter 8.

to mental phenomena.[36] Most of all, they liked the mathematical tools and what they saw as their potential for building intelligent machines. They started exploring the technique offered by McCulloch and Pitts. The mathematical techniques got elaborated, modified, and enriched, but the goal remained to explain knowledge in particular and the mind in general using "computing" mechanisms.

The connection drawn by McCulloch and Pitts, via their nets, between mental phenomena and computation had an effect on the way the Church-Turing Thesis (CT) was interpreted. Under McCulloch and Pitts's theory, every net could be described as computing a function. So in the sense in which McCulloch-Pitts nets computed, and to the extent that McCulloch-Pitts nets were a good model of the brain, every neural activity was a computation. This was particularly significant given that McCulloch and Pitts considered the computations of their nets to be explanations of mental processes. Given their theory, the whole brain was turned into a computing mechanism. So computing was transformed from a specific human activity among others (e.g. cooking, walking, or talking) into all that brains did, and CT was turned from a thesis about what functions can be effectively calculated into a thesis about the intrinsic limitations of brains.[37]

After McCulloch and Pitts's paper, the idea that CT is somehow a psychological thesis about human cognitive faculties would stick and would be used time and again to justify computational theories of the brain. For example, von Neumann made statements that resembled this interpretation of CT. Another idea would be that since the brain can only do what is computable, there is a computational theory of the brain (e.g. Webb 1980). It is ironic that McCulloch and Pitts made many of their simplifying assumptions about networks of neurons in order to solve the mind-body problem by using

[36] Cf. Wiener 1948, p. 147; Shannon and McCarthy 1956.

[37] If people had realized that this shift in the interpretation of CT was an effect of McCulloch and Pitts's theory, with its assumptions about the computational and epistemological significance of neural activity, this would be unproblematic. The problem is that after this paper, a lot of people took McCulloch and Pitts's reinterpretation of neural activity and CT without question, and thought it based on mathematically proven facts about brains. In this changed context, it became natural for many people to read even Turing's own argument for CT as a defense of computationalism. The question of the exact relationship between computationalism and CT is addressed at length in Chapter 7.

logic and Turing Machines as descriptions of the nervous system, but after their theory was formulated, their theory was used as evidence that the brain is indeed a computing mechanism.

5 FROM BRAINS TO COMPUTERS AND BACK AGAIN, 1943-1945

5.1 Migrations

During 1943, several of our characters moved. The first was Lettvin, who left Chicago for a neurological internship at the Harvard Neurological Service in Boston City Hospital.[1] Lettvin became friends with his fellow intern Alden Rainsbeck, and Rainsbeck introduced Lettvin to his distant relative, Norbert Wiener. Wiener's young protégé, Paley, had recently died in a mountaineering accident,[2] and Lettvin heard that Wiener was looking for a new pupil.

When Lettvin mentioned Walter Pitts and his talents, Wiener "professed disbelief":

> So Warren [McCulloch] and I got Walter [Pitts] on a train to Boston. Walter's meeting with Wiener at M.I.T. was characteristic and beautiful. Wiener didn't greet us as we entered but said, "I want to show you something interesting about the ergodic theorem." We followed him to the blackboard. After about five minutes Walter interrupted with an objection. The board extended over two sides of the small classroom. By the end of the hour they were deeply immersed on the second wall. I left, because there was no question about what had happened.[3]

Of Pitts's first visit, Wiener remembered a different, equally telling, episode:

> [Pitts] came to Boston to get in touch with me[,] and Rosenblueth had the two of us over for dinner. On that occasion we were extremely eager to find out what Mr. Pitts' ability was as we had heard very impressive statements about him from McCulloch and others. The result was that informally at the dinner table we subjected Pitts to what amounted to a very severe consecutive Doctor's examination in physiology and physiological psychology, mathematical logic and mathematics. The result of this inquisition was so remarkable that I immediately urged on our department the acquisition of Pitts.[4]

Wiener invited Pitts to work with him in Boston. But Pitts was committed to working with McCulloch, and especially Rashevsky, in Chicago.

McCulloch saw Wiener's offer as an opportunity for Pitts to grow as a scientist. Meanwhile, Rashevsky was growing impatient with Pitts's unorthodox ways. On August 9, 1943, Rashevsky wrote to McCulloch complaining that Pitts was not writing enough:

---

[1] Letter by Lettvin to Wiener, dated ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70.
[2] Lettvin 1989b, p. 515.
[3] Lettvin 1989b, p. 516.
[4] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers, Box 4, ff. 68.

> Mr. Pitts told me that after your return from vacation early in August you planned to have him work with you on the problem of thinking. While I am always entirely in favor of Pitts working with you at any time, I should call your attention to the fact, that by now he still has not completed the second half of the psychiatry paper. While it is not entirely out of question to publish that paper in the December issue, as planned originally, I still feel that we should have it as soon as possible. He originally promised it by the 1st of July. Then by the 6th. When he was here between July 6 and 13, he told me that it was all ready practically, and definitely promised it in the latter part of the month. When I saw him in Chicago on Saturday night, July 21, he said he was <u>just completing it</u>, and intended to mail it to me Monday. It never came. There is something more than a mere dislike for writing, and it makes me feel definitely pessimistic about his future work. At present I feel that if he goes off on another problem with you, he will never complete the psychiatry paper. May I ask you therefore to look into that matter, and if possible, make Pitts complete and turn over to Dr. Householder his ms., before he starts working with you.[5]

The unreliability here described by Rashevsky was going to get Pitts in trouble again in the future, but not with McCulloch.

McCulloch preferred the role of the middleman to that of the disciplinarian. On August 27th, as soon as Wiener's offer to Pitts came to the fore, McCulloch wrote Wiener in an attempt to prevent tensions with Rashevsky:

> Pitts has told me of your offer. I'm delighted, but to lose him at this particular time will be doubly hard on Rashevsky, for whose sake I would be grateful if you would write to him, stating 1) that you know Pitts himself, his work and his academic quandary at Chicago, 2) that it would be possible for him to get his Ph.D. in Mathematics at MIT in a couple of years, 3) that there is a position for him with you beginning (date) at so much per - . Above all, if possible do state that your institution is interested in mathematical biophysics and would like to have this member of his (Rashevsky's) group on that account. Leave me to convince Rashevsky that Chicago has nothing further to offer Pitts in mathematics, and that his going to you is a feather in R[ashevsky]'s cap. I cannot go into details as to Rashevsky's circumstances which at this juncture make me ask of you so much milk of human kindness.
>
> You know that you are hijacking my bootlegged collaborator, and that if you give my name as reference you can only expect hyperbolic extravaganza. Lacking him, I shall probably turn to you both for help with my own naughtiest notions.[6]

By Pitts's "quandary" in Chicago, McCulloch was probably referring to the formal requirements of the University of Chicago, which would have forced Pitts to take years' worth of courses before earning a Ph.D. McCulloch hoped that the more flexible MIT rules could be used as an excuse to convince Rashevsky to let Pitts leave.

[5] Letter by Rashevsky to McCulloch, dated August 9, 1943. Warren S. McCulloch Papers, ff. Rashevsky.
[6] Letter by McCulloch to Wiener, dated August 27, 1943. Warren S. McCulloch Papers.

On August 30th, Wiener wrote Rashevsky doing exactly what McCulloch had requested, and wrote McCulloch to let him know.[7] McCulloch wrote back on September 1st to warn Wiener that in spite of their careful tactics, Rashevsky was still likely to get upset:

> Thanks for the celerity. You will, nevertheless, expect at the present time some sort of explosion from R[ashevsky]—unless I miss my guess. After all, he knows what he's losing. I cannot imagine just what form it will take. He calls me, in jest, "psychiatrist to the Department of Mathematical Biophysics"—principally because I have managed to avert a couple of gratuitous catastrophes. I know full well Pitts' academic story, and that it dates from his association with R[ashevsky] and is in no small measure due to that petty sentimental tyranny called paternalism. From the bottom of my heart, I am glad that at least this man is out from under a situation which I myself would find intolerable. I say this as a psychiatrist so that you are duly forewarned—whatever form the explosion takes.[8]

We do not know whether Rashevsky "exploded," but we do know that over the following years, McCulloch would have to play the role of the psychiatrist again.

Pitts, at any rate, moved to Boston, where Wiener was planning to get Pitts involved with Rosenblueth:

> Until Rosenblueth goes wherever destiny may take him, he and myself and Pitts and a few other people are going to run a seminar on scientific mathematics privately at Rosenblueth's house. I think we are going to get somewhere.[9]

The seminar planned by Wiener could not last very long, because Rosenblueth had lost his job at Harvard in what he called a "nasty experience."[10]

McCulloch offered Rosenblueth a job at the University of Illinois in Chicago. In October 1943, McCulloch was sufficiently convinced that Rosenblueth would accept his offer that he started lobbying Frank Fremont-Smith to procure funds to set up Rosenblueth's lab.[11] Indeed Rosenblueth was "eager" to go to Chicago, but the University of Illinois informed him that in order to be offered a permanent position, he needed to be an American citizen. Rosenblueth, however, felt that in order to become

---

[7] Letter by Wiener to Rashevsky, dated August 30, 1943. Norbert Wiener Papers, Box 4, ff. 65. Letter by Wiener to McCulloch, dated August 30, 1943. Warren S. McCulloch Papers. Also Norbert Wiener Papers, Box 4, ff. 65.
[8] Letter by McCulloch to Wiener, dated September 1, 1943. Warren S. McCulloch Papers.
[9] Letter by Wiener to McCulloch, dated August 30, 1943. Warren S. McCulloch Papers. Also Norbert Wiener Papers, Box 4, ff. 65.
[10] Letter by Rosenblueth to McCulloch, dated October 26, 1943. Warren S. McCulloch Papers, ff Rosenblueth.
[11] Letter by McCulloch to Fremont-Smith, dated October 23, 1943. Warren S. McCulloch Papers, ff. Fremont-Smith.

American, he would need to have a "permanent, adequate, and happy job."[12] It was a catch 22. Part of the problem was that Rosenblueth had also been offered a permanent post at the Instituto Nacional de Cardiologia, in Mexico City, which he would have had to give up if he had become American.[13]

Rosenblueth took the Mexican job, but he remained in close contact with the American group. Initially, Rosenblueth encountered some logistical difficulties and he worried that he might be unable to set up a proper research laboratory in Mexico. But by August of the following year, he wrote Wiener that he was "working at full speed," he had funding for research, the Instituto was constructing a new building devoted to his lab, and he was about to publish his first paper stemming from his new research, "on the mechanism of production of the action potential of nerve."[14] Rosenblueth seemed to have found the "permanent, adequate, and happy job" he was looking for.

5.2 Brains and Computers

Pitts and Wiener started working closely together. With Wiener, Pitts studied both pure and applied mathematics.[15] According to Rosenblueth, Pitts developed "a very deep and sincere love and esteem and loyalty, and admiration for [Wiener]."[16] Unfortunately, most of the work Pitts did with Wiener was never published, and Pitts later destroyed the manuscripts. In order to know what Pitts worked on, we must rely on second-hand reports.

We know Wiener had an interest in thermodynamics. He thought there was a deep connection between the thermodynamic notion of entropy and the notion of information that he had developed while working with Bigelow: information was the inverse of entropy (Wiener 1948). Wiener had done work in statistical mechanics, developing tools to study thermodynamic systems composed of large numbers of

[12] Two letters by Rosenblueth, one to McCulloch and one to Francis Gerty, dated October 26, 1943. Warren S. McCulloch Papers, ff. Rosenblueth.
[13] Two letters by Rosenblueth, one to McCulloch and one to Francis Gerty, dated October 26, 1943. Warren S. McCulloch Papers, ff. Rosenblueth.
[14] Letter by Rosenblueth to Wiener, dated August 13, 1944. Norbert Wiener Papers Box 4, ff. 66.
[15] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers Box 4, ff. 68.
[16] Letter by Rosenblueth to Wiener, dated April 9, 1947. Norbert Wiener Papers Box 5, ff. 77.

components interacting at random, such as gases. Because brains processed information and were made out of a huge number of neurons, Wiener and Pitts thought that large neuronal ensembles might be amenable to analysis by the mathematical tools of statistical mechanics, provided that the neurons were randomly connected to each other.[17]

According to McCulloch, over a course of several years Pitts did very significant work on random nets and other biological systems that could be analyzed by statistical mechanical means:

> Walter [Pitts] agreed with Norbert [Wiener] as to the importance of random nets. Walter found a way of computing this by means of the probability of a neuron being fired in terms of the probability of activity in the neurons affecting it… [Pitts and Wiener] worked on the scyncytium of heart muscle, which is random in detail of connections of neighbors by protoplasmic bridges, and on the pools of motor neurons with fairly random connections to them from afferent peripheral neurons playing upon them.[18]

And in January 1945, Wiener praised Pitts's work to Rosenblueth. "Pitts," Wiener wrote, "has some very fine ideas on the statistical mechanics of nerve nets."[19]

Pitts became a trusted collaborator of Wiener, who consulted him on much of his work. For instance, in May 1944, while Lettvin and Pitts were on a trip, Wiener sent Pitts "the Statistical Mechanics manuscript," asking for suggestions before submitting it for publication.[20] In August, Wiener wrote Pitts saying that "we shall be delighted to see you down here around the 19th" and invited Pitts to stay at his house: "I have a lot of new stuff to talk with you both scientific and personal."[21] Wiener later said that Pitts gave him "suggestions in my own research work which I have found sound and important."[22]

One of their topics of conversation was the new digital computers that were being developed at the time. Wiener and Pitts were following the developments of computing machinery, in part thanks to Wiener's connection with Julian Bigelow. After working with Wiener, in 1943 Bigelow joined the

---

[17] Incidentally, the application of mathematical tools taken from statistical mechanical to the study of neural networks was one of the main innovations behind the second big wave of mathematical research on neural-like networks, which started with the work of John Hopfield (1982). By then, the units of the networks were no longer conceived of as logic gates, as in McCulloch and Pitts's theory. As far as I know, there is little direct historical connection between these two efforts.
[18] McCulloch 1974, p. 44.
[19] Letter by Wiener to Rosenblueth, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.
[20] Letter by Wiener to Pitts, dated May 11, 1944. Norbert Wiener Papers Box 4, ff. 66.
[21] Letter by Wiener to Pitts, dated August 12, 1944. Norbert Wiener Papers Box 4, ff. 66.
[22] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers Box 4, ff. 68.

Applied Mathematics Group led by Warren Weaver at Columbia University in New York City.[23] Wiener and Pitts visited Bigelow together, and in August 1944, Bigelow sent Wiener a clipping on the Harvard Mark 1 Computer, adding his thoughts on Aiken's machine and Bigelow's own future role in mechanical computing: "I think the write-up is simply dazzling. All joking aside, it is a good piece of apparatus, and it behooves M.I.T's Center of Analysis to look to its laurels and, in fact, to hire guys like me after the war, I hope."[24]

According to Wiener, only after Pitts went to MIT the two of them thought seriously about the connection between brains and computers. Pitts realized "how definitely his [1943] work bore on the design of computing machines," and "plunged into this line of investigation with great enthusiasm."[25] The connection they saw had to do with the fact that both brains (as described by McCulloch and Pitts's theory) and digital computers did things using binary components that employed a binary notation.[26] Through Pitts's move to MIT, McCulloch and Pitts's theory of the brain was starting to have an impact, and their "neurons" were being compared to the relays and vacuum tubes of digital computers.

An exchange between Wiener and Edwin Boring, a Harvard psychologist, sheds further light on Wiener's thinking about brains and computers at that time. The two probably met at an "Inter-Scientific Discussion Group" that took place in Boston. On November 13, 1944, Boring wrote to Wiener:

> Your suggestion that all the functions of the brain might be duplicated in electrical systems is one that I find very attractive. I did not pursue the matter with you, because … I have so much war work on these days that it is hard to squeeze in even a luncheon conversation.
>
> But here is a proposition for you. Let me, as I find time, try to make out what I consider to be a pretty complete list of psychological functions. Any psychological fact can, of course, be expressed in terms of stimulus and response by putting it in operational terms. A symbolic process is simply a delayed, adequately differential reaction. Introspection is a reaction to a reaction, or some reaction in that regress. And so on. If I take time, I might get other psychologists to criticize my list and perhaps amplify it.
>
> Then let me turn the list over to you to see if you can specify electrical or electronic systems that will give the same specificity of 'output' to 'input.' I do not know that you can, but I should be betting on you.

[23] Julian Bigelow's Vita, undated, Papers of John von Neumann, Box 2, ff. 7.
[24] Letter by Bigelow to Wiener, dated 7 August 1944. Norbert Wiener Papers, Box 4 ff. 66.
[25] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[26] Wiener 1948, p. 23.

> If you can, then you will find that I am after you to publish a paper for the benefit of the psychologists, for this is a point that many of us operationally-minded psychologists should like to see made.
>
> Is it a go?
>
> I have scribbled down 14 properties that the system should have, but I am not satisfied. I think I left some essentials out, like Generalization and Abstraction, which must, of course, be reduced operationally to system properties.[27]

Unfortunately, Boring was unable to pursue his project of writing the complete list of psychological functions that needed to be imitated by an "electronic brain,"[28] because on December 2nd, Boring fell gravely ill. On February 8, he wrote Wiener again:

> What I had done before December 2 is to make out a list of what I thought all the functions of the brain are, putting them in positivistic reaction terms of the organism, terms which could be translated into in-put, out-put and adjustment of a mysterious box with binding posts and knobs on it. That's about what a person is really. I wanted to get this revised to the best of my ability, then get some other psychologists to criticize it. Then I wanted to send it to you. I still want to do all that, and I hope that your talk next week is not going to discharge your whole interest in this affair.[29]

Boring had to "take things easily" for several months because of his illness, and there is no evidence that he and Wiener continued their exchange.

Parallel to Wiener and Pitts, and in constant communication at least with Wiener, John von Neumann was also thinking hard about brains and computers. Specifically, von Neumann was trying to develop an understanding of the functional organization of modern digital computers in full generality, using ideas from Turing and McCulloch and Pitts. Digital computers, as von Neumann understood them, would generate a lot of excitement in the small community of researchers that was being formed around McCulloch and Wiener.

5.3 Electronic Brains

Early computing machines, up to the mid-1940s, were designed to solve specific classes of problems in mathematics and physics, typically solving certain classes of differential equations. These machines

[27] Letter by Boring to Wiener, dated November 13, 1944. Norbert Wiener Papers, Box 4, ff. 66.
[28] Boring used the expression "electronic brain" in his next letter to Wiener, dated February 8, 1945. Norbert Wiener Papers.
[29] Letter by Boring to Wiener, dated February 8, 1945. Norbert Wiener Papers.

could be set up to solve any particular problem within the relevant class, but they could not be set up to compute any computable function. This was true of both analog devices, such as the differential analyzers, and of digital devices, such as the ABC designed by John Atanasoff at Iowa State College in 1940. One exception was the old Analytical Engine, designed by Charles Babbage in 1856. But Babbage's computer was designed a long time before Turing clearly defined the notion of computability, and it was never built.

Turing's paper of 1936-7 contained the first theoretically justified notion of a universal computing machine, which could be programmed to compute any computable function by writing the appropriate program on its tape. Turing hoped, some day, to build one of those universal machines. But in the meantime, his work was known only within a restricted circle of mathematicians interested in foundational topics. It took several years before the physicists and engineers who were involved in designing computing machinery paid attention to Turing's work. Much of the credit for the spread of Turing's notion of universality into the engineering community belongs to von Neumann.

At least since 1938, von Neumann was interested in the nature of the observer in Quantum Mechanics. In connection with this topic, from 1939 to 1941 von Neumann corresponded with Hungarian physicist Rudolf Ortvay about brain theory. Ortvay expressed the view that physicists or mathematicians would soon make a theoretical breakthrough about brains, explaining their functioning in terms of the activities of cells and their connections. He also drew some analogy between brains and electronic calculating equipment. In 1940, Ortvay suggested that von Neumann pursue the theoretical study of brains as networks of neurons communicating with one another as automatic telephone switching equipment.[30] Von Neumann did not heed Ortvay's advice to pursue brain theory until he read McCulloch and Pitts's paper, which happened shortly after its publication.[31]

In August 1944, von Neumann began consulting for the Moore School of Electrical Engineering at the University of Pennsylvania. There, two engineers, J. Presper Eckert and John Mauchly led a team

[30] For an account of these events, see Aspray 1990, pp. 178-179. More generally, Aspray 1990 contains a detailed account of von Neumann's work on computation and computers.
[31] Aspray 1990, p. 180.

that was building the ENIAC—an electronic, digital version of the differential analyzer. With von Neumann, the Moore team discussed the design of a computer that was supposed to solve a wider range of problems than the ENIAC, the EDVAC. The EDVAC was to have a larger storage capacity than the ENIAC, so as to store more intermediate values of calculations.[32] The EDVAC was designed to solve non-linear partial differential equations of two or three independent variables and other similarly complex problems. It must also recognize some of its own malfunctions.[33]

Von Neumann wrote a "First Draft of a Report on the EDVAC," which a member of the Moore School team, Herman Goldstine, started circulating in computing machinery circles in June 1945. Although it was called a "first draft," it was a fairly polished product. And although the "First Draft" was nominally on the EDVAC, in it von Neumann articulated in a general form the functional organization of a programmable, stored-program, universal computer. Von Neumann's "First Draft" never became a final draft, because it was soon superseded by more detailed work written by von Neumann with some collaborators (e.g., Goldstine and von Neumann 1946, Burks, von Neumann and Goldstine 1946).

Von Neumann began his "First Draft" by introducing automatic digital computing in general. For him, the paradigm of mechanical computing was the finding of numerical solutions to differential equations. He pointed out that in order to define the problem and give the numerical data, a "code" was required.[34] Then von Neumann described the five main components of a digital computer, which he called "organs."

In order to solve differential equations, elementary arithmetical operations (addition, subtraction, multiplication, and division) must be performed very frequently. Because of this, von Neumann suggested that computers should contain "specialized organs" for arithmetical operations, namely arithmetic organs.[35]

[32] Aspray 1990, p. 37.
[33] Aspray 1990, p. 39.
[34] Von Neumann 1945, p. 383.
[35] Ibid., p. 384.

Given a particular problem to be solved, different sequences of arithmetical operations needed to be performed. In order to determine what sequences of operations the machine must perform in any given case, the user of the machine must write down appropriate instructions using the appropriate code. The machine must have an appropriate "organ," called the logical control, which responded to the instructions so as to generate the relevant sequence of operations. Von Neumann explained that by appropriately distinguishing the logical control from the instructions, a new generation of "all-purpose" computers could be built, which could solve more problems than previous, "special-purpose" computing devices:

> If the device is to be *elastic*, that is as nearly as possible *all purpose*, then a distinction must be made between the specific instructions given for and defining a particular problem, and the general control organs which see to it that these instructions … are carried out. The former must be stored in some way … the latter are represented by definite operating parts of the device.[36]

The notion of "all purpose" computer was related to Turing's notion of universal computing machine. In a letter written in 1949, von Neumann drew the connection as follows:

> [A] "general purpose" mathematical machine of the type which is being now developed in various places … should be (or at least might be) "universal" in the sense of A. M. Turing … The only differences would be differences of convenience: The mathematical "all-purpose" machines have special arrangements to accelerate certain particular processes which are exceptionally frequent in normal mathematics (like the four species of digital arithmetic).[37]

Aside from the existence of an arithmetic unit, there was another important difference between Turing's universal machine and the new computers described by von Neumann. The new machines had an internal memory component, whose purpose was to store intermediate results, data, look-up tables, and even the instructions for the machine, i.e. the programs. This is why they would be called stored-program computers.

After introducing the three main internal components of the machine, namely central arithmetic, central control, and memory, von Neumann drew an analogy between the computer and the brain:

> The three specific parts correspond to the associative neurons in the human nervous system. It remains to discuss the equivalents of the sensory or afferent and the motor or efferent neurons. These are the input and the output organs of the device.[38]

---

[36] Ibid., p. 384; emphasis original.
[37] Letter by von Neumann to McKinsey, dated February 18, 1949, Papers of John von Neumann, Box 17, ff. 6.
[38] Von Neumann 1945, p. 385.

The analogues of sensory neurons were the input "organs," by which data and instructions were inserted into the machine. The analogues of motor neurons were the output "organs," by which results of the computations were given by the machine. With the input and output "organs," the functional description of the computer was complete.

Von Neumann pointed out that the medium for input and output (e.g., a Turing Machine tape) had the properties of a memory, and it was used as such for most purposes by existing computing machines (e.g., the Harvard Mark 1 Computer). The main goal of the new electronic computers, however, was to compute as fast as possible. A high-speed computer would be limited in its usefulness by having to rely only on the input-output medium as memory. That is why it was so important to have an internal memory "organ" for programs, data, look-up tables, and intermediate results.[39] By making this point, von Neumann emphasized the functional difference between the kind of universal machine originally described by Turing, which relied only on the tape as its memory component, and the stored-program principle employed in the new electronic machines.[40]

Finally, von Neumann went into more details about the "elements" out of which computers were built. He didn't describe the physical properties of the "elements." What mattered, he said, was that the elements' behavior be "digital" relative to the functioning of the machine:

> Every digital computing device contains certain relay like elements, with discrete equilibria. Such an element has two or more distinct states in which it can exist indefinitely. These may be perfect equilibria, in each of which the element will remain without any outside support, while appropriate outside stimuli will transfer it from one equilibrium into another. Or, alternatively, there may be two states, one of which is an equilibrium which exists when there is no outside support, while the other depends for its existence upon the presence of an outside stimulus.[41]

Von Neumann pointed out that different mechanical or electrical devices could be used as "elements" for a digital computer.

---

[39] Ibid., p. 386.
[40] This point is important in light of the tendency among some historians to attribute to Turing the invention of the stored-program computer (e.g., Aspray 1990, p. 176; Copeland 2000a). This point is discussed in more detail in Chapter 10.
[41] Von Neumann 1945, pp. 387-388.

A crucial property of a computer was that the activity of the elements must be synchronized. Synchronization may be either an effect of the way the elements reacted to one another, or the effect of a master clock sending signals to the elements at fixed time intervals:

> Any such device may time itself autonomously, by the successive reaction times of its elements. In this case all stimuli must ultimately originate in the input. Alternatively, they may have their timing impressed by a fixed clock, which provides certain stimuli that are necessary for its functioning at definite periodically recurrent moments.[42]

Von Neumann called a device synchronized by a clock synchronous, and a device that synchronized "autonomously" (by the way its elements interact) asynchronous. Since stored-program computers needed to rely on the results of "several distinct sequences of operations performed simultaneously," whose outputs must all be synchronized, "the clock impressed timing is obviously preferable."[43]

Then, von Neumann argued that neurons were digital "elements":

> It is worth mentioning, that the neurons of the higher animals are definitely elements in the above sense. They have all-or-none character, that is two states: Quiescent and excited. They fulfill the requirements … with an interesting variant: An excited neuron emits the standard stimulus along many lines (axons). Such a line can, however, be connected in two different ways to the next neuron: First: In an excitatory synapsis, so that the stimulus causes the excitation of that neuron. Second: In an inhibitory synapsis, so that the stimulus absolutely prevents the excitation of that neuron by any stimulus on any other (excitatory) synapsis. The neuron also has a definite reaction time, between the reception of a stimulus and the emission of the stimuli caused by it, the synaptic delay.[44]

Von Neumann here was explicitly following McCulloch and Pitts (1943) in ignoring "the more complicated aspects of neuron functioning."[45] The point was that one could use other "all-or-none" elements, such as vacuum tubes, to imitate neurons, thereby getting them to compute the logical functions that McCulloch and Pitts showed neurons could perform:

> It is easily seen, that these simplified neuron functions can be imitated by telegraph relays or by vacuum tubes. Although the nervous system is presumably asynchronous (for the synaptic delays), precise synaptic delays can be obtained by using synchronous setups.[46]

---

[42] Ibid., p. 388.
[43] Ibid., p. 388.
[44] Ibid., pp. 388-389.
[45] Ibid.
[46] Ibid., p. 389.

The designer of digital computers could thus rely on the technique employed by McCulloch and Pitts "for all preliminary, orienting considerations on vacuum tube systems," that is to say, for designing vacuum tube systems to perform desired arithmetical operations in the binary system.[47] Von Neumann had recruited McCulloch and Pitts's theory of the brain to help design digital computers. This would further encourage people like McCulloch, Wiener, and Pitts to think of the brain itself as a computer.

5.4 A Research Program

Some time in 1944, Pitts got drafted for the war. He received his psychiatric examination by A.A. Brill, the editor of Freud's *Collected Papers*. Pitts—who at the time of the interview wore a goatee—later told friends that Brill asked him, "Young man, why do you wear a beard?" to which Pitts replied, "Old man, why do you?" Pitts was judged "pre-psychotic," given a top-security clearance, and assigned to work at the Kellex Corporation in New York City. At Kellex, which was the Manhattan branch of the Manhattan Project, Pitts calculated breakdown times of uranium.[48]

Pitts detested General Groves, the head of the Manhattan Project, and according to Wiener, in New York he was "lonesome."[49] In October 1944, Wiener visited Pitts in New York, "partly for business and partly to cheer him up."[50] As to the friendly part of the trip, Wiener wrote Pitts saying, "I have a lot to talk over with you … concerning computing machine theory."[51] As to the business part of the trip, Wiener wrote von Neumann:

> I saw Aiken the other day and I am much impressed by the coincidence of his point of view with mine on the future of computing machines. We are thinking of trying to get some meeting of the American Society of the Advancement of Science after the war is over for the discussion of the whole complex problems relating to computing machines, communication engineering, prediction theory, and control engineering. I would like to get your point of view on that.[52]

---

[47] Ibid., p. 389.
[48] These events are narrated by Smalheiser 2000, p. 221, based on interviews with Pitts's friends. See also Lettvin 1949d, p. 516.
[49] Letter by Wiener to Rosenblueth, dated October 19, 1944. Norbert Wiener Papers, Box 4, ff. 66.
[50] Letter by Wiener to Rosenblueth, dated October 19, 1944. Norbert Wiener Papers, Box 4, ff. 66.
[51] Letter by Wiener to Pitts, dated October 17, 1944. Norbert Wiener Papers Box 4, ff 66.
[52] Letter by Wiener to von Neumann, dated October 17, 1944. Norbert Wiener Papers, Box 4, ff 66.

To get von Neumann's point of view, Wiener asked if he and von Neumann could meet while he was visiting Pitts in New York.

Two days after writing von Neumann, Wiener wrote Rosenblueth to inform him of the new tentative project. He told Rosenblueth of his discussion with Aiken and the proposed meeting of the American Society of the Advancement of Science on "the complex of subjects in which you and I are interested." Wiener continued:

> If we do that we shall try to make it a matter of invited addresses of perhaps a half an hour each by a mathematician, physiologist, electrical engineer, statistician, etc. Certainly, this wants to be done in such a way that you can participate in it.[53]

Wiener considered Rosenblueth an essential collaborator. At that time, he was also trying to convince MIT to give him enough flexibility that he could travel to Mexico to work with Rosenblueth, so that the two could continue to work closely together.

After Wiener and von Neumann met, von Neumann got on board. They must have felt they shouldn't wait for a meeting of the American Society of the Advancement of Science. For on December 4, a letter signed by Aiken, von Neumann, and Wiener was inviting McCulloch and others to a meeting of engineers, physiologists, and mathematicians:

> A group of people interested in communication engineering, the engineering of computing machines, the engineering of control devices, the mathematics of time series in statistics, and the communication and control aspects of the nervous system has come to a tentative conclusion that the relations between these fields of research have developed to a degree of intimacy that makes a get-together meeting between people interested in them highly desirable.
>
> Owing to the war, it is not yet time to call together a completely open meeting on the matter, because so many researches developed in the war effort are concerned, but it seems highly desirable to summon together a small group of those interested to discuss questions of common interest and make plans for the future development of this field of effort, which as yet is not even named.[54]

McCulloch was "eager to come and delighted to have been included."[55]

Besides McCulloch, the tentative list of invited participants comprised Pitts as well as S. S. Wilks of Princeton, E. G. Vestine of the Carnegie Institute, W. E. Deming of the Census Bureau, R. Lorente de

[53] Letter by Wiener to Rosenblueth, dated October 19, 1944. Norbert Wiener Papers, Box 4, ff. 66.
[54] Letter by Aiken, von Neumann, and Wiener to McCulloch, dated December 4, 1944. Warren S. McCulloch Papers. A draft of the letter (with recipient left blank) is found in Norbert Wiener Papers, Box 4, ff. 66.
[55] Letter by McCulloch to von Neumann, dated December 20, 1944. Warren S. McCulloch Papers.

Nó of the Rockefeller Institute, and L. E. Cunningham of Aberdeen Proving Ground. Wiener wished Rosenblueth could have been involved, and von Neumann asked Wiener and Aiken to add Goldstine to the list.[56]

The next letter to the group contained a proposal, attributed to Wiener and Aiken, to call the group the Teleological Society. The proposal was briefly justified and followed by an ambitious plan:

> Teleology is the study of purpose of conduct, and it seems that a large part of our interests are devoted on the one hand to the study of how purpose is realized in human and animal conduct and on the other hand how purpose can be imitated by mechanical and electrical means. If this suggestion should meet the approval of the group and if at any stage it appears desirable for us to sponsor a periodical, then an appropriate name might be Teleology or we are contemplating an international Teleologia. This suggestion is made without any desire to force the hands of the group and simply to give the members something to think about.
>
> In addition to the name of the science, possible publication of the Journal, and similar matters, possible agenda include the question of what steps should be taken to form a center for this sort of research and to approach various foundations for its support; what policy should be adopted about patents and inventions and other social relations of the group; what measures should be taken to bring our ideas to general scientific attention either at a meeting of the American Association for the Advancement of Science or elsewhere as may seem suitable; and how to protect the researches of the group from dangerous and sensational publicity. It is likely many other questions of similar importance will arise at the meeting, and it is suggested that you be prepared for any issues of the sort that are of interest to you. Of course, the main function of the meeting is the personal get-together and discussion of purely scientific issues.[57]

It may appear hubristic that before their first meeting, the organizers were already planning a new journal and a center of research. Nevertheless, in the years to come their ideas would, indeed, attract "sensational publicity."

The meeting took place in Princeton on January 6-7, 1945.[58] Although Aiken was unable to attend,[59] the meeting was experienced as "a great success."[60] On the first day, von Neumann talked about computing machines and Wiener on communication engineering. On the second day, McCulloch and

[56] Wiener 1948, p. 23. Letter by von Neumann to Wiener, dated December 16, 1944. Norbert Wiener Papers Box 4, ff. 66.
[57] Letter by Wiener (for Aiken, von Neumann, and Wiener) to Goldstine (and others), dated December 28, 1944. Norbert Wiener Papers, Box 4, ff. 66.
[58] Ever since Wiener described it as taking place in the "winter 1943-1944" in his book *Cybernetics* (Wiener 1948), the meeting of the Teleological Society has been usually misdated by one year in the literature.
[59] Wiener 1948, p. 23.
[60] Letter by Wiener to Rosenblueth, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.

Lorente de Nó described "the present status of the problem of the organization of the brain."[61] This was the first time in which McCulloch and von Neumann met. McCulloch remembered the event as colorful:

> Lorente de Nó and I, as physiologists, were asked to consider the second of two hypothetical black boxes that the allies had liberated from the Germans. No one knew what they were supposed to do or how they were to do it. The first box had been opened and exploded. Both had inputs and outputs, so labelled. The question was phrased unforgettably: "This is the enemy's machine. You always have to find out what it does and how it does it. What shall we do?" By the time the question had become that well defined, Norbert [Wiener] was snoring at the top of his lungs and his cigar ashes were falling on his stomach. But when Lorente and I had tried to answer, Norbert rose abruptly and said: "You could of course give it all possible sinusoidal frequencies one after the other and record the output, but it would be better to feed it noise—say white noise—you might call this a Rorschach." Before I could challenge his notion of a Rorschach, many engineers' voices broke in. Then, for the first time, I caught the sparkle in Johnny von Neumann's eye. I had never seen him before and I did not know who he was. He read my face like an open book. He knew that a stimulus for man or machine must be shaped to match nearly some of his feature-filters, and that white noise would not do. There followed a wonderful duel: Norbert with an enormous club chasing Johnny, and Johnny with a rapier waltzing around Norbert—at the end of which they went to lunch arm in arm.[62]

The participants reached consensus that whether they were engineers, mathematicians, or neurophysiologists, they were interested in the same subject matter. They already had a set of common ideas they could use. If they could develop a common vocabulary, they could learn even more from each other. They agreed to meet again in the spring, after drawing up a common "research program."[63]

The task of assigning the first tasks fell on von Neumann, who on January 12th wrote a detailed letter to the participants:

> In conformity with the agreement reached at the end of our meeting on January 6-7 in Princeton, I will state in what follows our decision on a research program. It was decided that the details of such a program should be first worked out essentially by four distinct groups among the participants. These groups are defined by the aspect of the subject in which each is primarily interested. Each group should attempt to formulate in the next few months a draft memorandum containing the motivation and description of the research in which it would be most actively interested. While no group need restrict itself only to what it considers its primary interest, each one may feel free to lay the main emphasis on that particular field. A balanced presentation which distributes the emphases properly over the entire subject will be worked out by all groups together subsequently. After these four preliminary draft memoranda are formulated, they should be circulated between the four groups, and a first process of adjustment should be carried out on the basis of the comments obtained at this stage. After this, another meeting of all groups, similar

[61] Ibid.
[62] McCulloch 1974, p. 40.
[63] Wiener 1948, p. 23. McCulloch 1974, p. 40. Letter by Wiener to Rosenblueth, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.

> to that of January 6-7, should be called, at which the formulation of a joint final memorandum of the research program will be undertaken and a small group put in charge of writing it.[64]

The four groups, which were to prepare four preliminary memoranda, were as follows: (1) on "filtering and prediction problems," Pitts and Wiener; (2) on "application of fast, mechanized computing methods to statistical problems," Deming, Vestine, Wilks, and possibly von Neumann; (3) on "application [of computers] to differential equations," Aiken, Cuningham, Goldstine, and von Neumann; finally, (4) on "connected aspects of neurology," Lorente de Nó, McCulloch, and Pitts.[65] Von Neumann suggested that the preliminary memoranda be prepared within three months, so that the group could reconvene within six months.

Wiener took his part of the project to heart. He started working on his memorandum with Pitts, and wrote Rosenblueth enthusing about the meeting. After describing the recent events, Wiener pointed out that both Henry Allen Moe of the Guggenheim Foundation and Warren Weaver of the Rockefeller Foundation had expressed interest in funding the project. In addition, McCulloch, and especially von Neumann, seemed to possess valuable skills and connections:

> McCulloch and von Neumann are very slick organizers, and I have heard from von Neumann mysterious words concerning a some thirty megacucks [*sic*] which is likely to be available for scientific research. von Neumann is quite confident that he can siphon some of it off.[66]

Wiener asked for Rosenblueth's suggestions and promised to bring him and Bigelow into the group.

Wiener also inquired about creating a center of research in the new field at MIT, where his proposal met a "friendly but not decisive" attitude.[67] Initially, Wiener thought his proposed center of research could naturally seek the support of Harvard's Medical School for the neurophysiological side of the program. Then he realized that this was not a promising venue, because "Harvard Medical … has become completely clinical."[68] But he wasn't deterred. He thought that if they could find some external funding, MIT would agree to develop a "physiological program" on its own, and the MIT program could

---

[64] Letter by von Neumann, dated January 12, 1945. Papers of John von Neumann, Box 12, ff. 1.
[65] Ibid.
[66] Letter by Wiener to Rosenblueth, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff.67.
[67] Letter by Wiener to von Neumann, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.
[68] Ibid.

be designed from the start to fit the interests of the Teleological Society group.[69] A few years later, part of Wiener's plan would eventually be realized, so that McCulloch, Lettvin, and others were hired at MIT.

After writing a first draft of his memorandum with Pitts, Wiener wrote von Neumann announcing that as soon as Pitts had finished going over it, they would send it to von Neumann.[70] In his letter, Wiener raised and lucidly discussed the relationship between computation and control:

> I found one thing missing in your assignment of topics: namely, there was no single place where the problem of transition from the computing machine to the control machine was discussed. I think this is one of the most important aspects of our project and as it is closely related to the prediction and filtering problems, I have assumed that it goes to our subcommittee for a report. The issues that come up here are those of transfer from continued data to counted data; of the final transition from counted data to the motion of a shaft effector; and the sensing of the motion of the effector by feed-back or other quasi-proprioceptor apparatus. I am quite convinced that feel-back method of proprioceptor needs to be supplemented by apparatus which rather reads the load than merely works with a linear computation of the input and the output of the motor apparatus. I have already gone some way towards developing such a theory.
>
> I have also taken the liberty of emphasizing that the same type of proprioceptor arises in purely mechanical controls, purely organical controls, and controls with a mechanical and organical parts combined together. However, you will see this in our report.[71]

Wiener was discussing how to use digital computers for control purposes. The problem was to get sensors and control devices to cooperate proficiently with a digital mechanism devoted to computing the actions that the control had to take at any given time. The problem was that sensors and control devices had to operate with continuous physical variables, whereas a digital mechanism had to operate with discrete digits. A method of effective transduction between continuous and discrete variables had to be created, and Wiener was already busy developing it.

Von Neumann agreed with Wiener's "emphasis on control mechanisms," and was glad Wiener was addressing the topic. He also hoped to talk with Wiener about his own ideas on the subject, and to see Wiener and Pitts's memorandum as soon as it was ready. Von Neumann was ashamed to admit that their own memorandum was "much less far along."[72]

---

[69] Ibid.
[70] Letter by Wiener to von Neumann, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67. Letter by Wiener to von Neumann, dated January 31, 1945. Norbert Wiener Papers, Box 4, ff. 67.
[71] Letter by Wiener to von Neumann, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.
[72] Letter by von Neumann to Wiener, dated February 1, 1945. Norbert Wiener Papers, Box 4, ff. 67.

The Teleological Society group appears to have dissolved shortly thereafter, with Aiken and most of the other members disappearing from sight. Aside from Wiener's letters about his memorandum, I have found no evidence of the preliminary memoranda or of further meetings of the group. The main problem seems to have been that the war was still going on, which meant two things. First, most of the participants were very busy with their "war work." Second, most of the relevant work was classified and under the control of different governmental agencies. Because of this, at the Princeton meeting they had decided not to organize in a permanent Teleological Society, and to postpone the creation of a society, a journal, and a center of research until after the war.[73]

In spite of the brief life of the Teleological Society, the "research program" drawn by von Neumann shows that by January 1945, there was a group of neurophysiologists, engineers, and mathematicians who publicly recognized each other as belonging in the same new scientific field. What they had in common was an interest in information, computation, and control broadly construed, in either biological organisms or machines. Their new field was still unnamed and largely undefined: at the end of his letter on the "research program," von Neumann pointed out he had not described their field, and in a separate letter to Wiener, von Neumann asked for Wiener's opinion on the matter.[74] And yet they were serious enough about their new enterprise that they attempted to coordinate a common scientific effort among several research groups, involving researchers from different fields of origin working at different institutions.

Some of the participants to the Teleological Society did not keep in touch. Others—namely Wiener, McCulloch, von Neumann, and Pitts, plus Rosenblueth from Mexico—maintained their connection and continued to pursue common plans.

[73] Letter by Wiener to Rosenblueth, dated January 24, 1945. Norbert Wiener Papers, Box 4, ff. 67.
[74] Letter by von Neumann, dated January 12, 1945. Papers of John von Neumann, Box 12, ff. 1. Letter by von Neumann to Wiener, dated January 12, 1945. Norbert Wiener Papers, Box 4, ff. 67.

5.5 Preparing for the End of the War

Encouraged by von Neumann, Wiener pursued his plan of creating a center of research for the new field. He dreamt of hiring von Neumann at MIT, and during the summer of 1945, serious negotiations took place between MIT and von Neumann. Wiener felt that if MIT hired von Neumann, their ideas "concerning an organized collaboration between physiological and mathematical subjects will follow as a matter of course."[75] A consequence of that collaboration would be for Rosenblueth to visit MIT during at least part of the year. Wiener talked to Moe of the Guggenheim Foundation about the Rosenblueth plan and about the need to support Pitts's return to MIT after the end of his military service. Moe gave Wiener the sense of being interested in financing the Rosenblueth plan, and he sent Pitts application forms for a Guggenheim Fellowship.[76]

That same summer, Wiener himself went to Mexico to work with Rosenblueth. According to Wiener, they worked "in the study of the applications of the theory of networks of sensitive tissues to the flutter and fibrillation of the heart."[77] Rosenblueth thought their work together "was so successful that it would be a shame not to renew it and continue it. Indeed, I think it is rather urgent that we go on with what we have started and allied topics."[78] Wiener left Mexico around June 19th. On June 26th, Rosenblueth sent Wiener the manuscript they had collaborated on, asking Wiener to review it and to ask Pitts what he thought of it.[79] Although this was not included in the research program drawn by von Neumann in January, it was still one of the first collaborative efforts of their group, and Wiener was anxious to get feedback on it.

The paper attracted everyone's attention. First, Pitts looked at it "thoroughly." Then, in July, Wiener and Pitts went to visit McCulloch in Chicago. McCulloch told Rosenblueth that when Wiener and Pitts visited, they "were full of your experiments and Wiener's calculations on the theory of flutter

[75] Letter by Wiener to Rosenblueth, dated July 11, 1945. Norbert Wiener Papers, Box 44, ff. 68.
[76] Letter by Wiener to Rosenblueth, dated July 11, 1945. Norbert Wiener Papers, Box 44, ff. 68.
[77] Letter by Wiener to de Santillana, dated October 16, 1945. Norbert Wiener Papers, Box 4 ff. 69.
[78] Letter by Rosenblueth to Wiener, dated June 26, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[79] Letter by Rosenblueth to Wiener, dated June 26, 1945. Norbert Wiener Papers, Box 4, ff. 68.

and fibrillation and they showed me swell manuscripts under way."[80] Wiener, in turn, appreciated both McCulloch's hospitality and visiting his lab; he came back "full of ideas for new work and I owe them to you."[81]

McCulloch had recently discussed "findings and theoretical considerations" with Fremont-Smith, who had suggested to organize a Macy-sponsored meeting on them. McCulloch kept Fremont-Smith updated about the work of the group:

> I am still of the opinion that the group which is forming around Pitts and Wiener … stands a better chance of contributing to breeching the woeful breeches between physiology and psychology and physiology and sociology than any lone worker anywhere. Nevertheless, I am plowing ahead on theory, usually at night, and I believe I have just begun to see the light with respect to the origin of hypotheses which may be wrong even though the nervous system enunciates no false propositions. I will try it out on Rashevsky's gang.[82]

Fremont-Smith was interested in physiological approaches to psychiatry and psychology, and he had probably added sociology in his discussion with McCulloch, because McCulloch felt compelled to mention sociology in his letter even though neither he nor any other member of the group had ever worked in it. By raising hopes that collaboration between different researchers had great potential for progress in bridging physiology and other human sciences, McCulloch was keeping Fremont-Smith's attention on the group.

McCulloch gave the Wiener-Rosenblueth manuscript on the heart "the once over," then Wiener sent it to von Neumann and got comments from him. Delighted by the "very favorable" response commentators gave to their paper, Wiener sent the revised manuscript back to Rosenblueth, who incorporated most of the new suggestions in the final version.[83]

---

[80] Letter by McCulloch to Rosenblueth, dated September 20, 1945. Warren S. McCulloch Papers, ff. Rosenblueth.
[81] Letter by Wiener to McCulloch, dated December 17, 1945. Norbert Wiener Papers, Box 4, ff. 69.
[82] Letter by McCulloch to Fremont-Smith, dated August 27, 1945. Warren S. McCulloch Papers, ff. Fremont-Smith.
[83] Letter by Rosenblueth to Wiener, dated September 3, 1945. Norbert Wiener Papers, Box 44, ff. 68. The quotes are all from letters by Wiener to Rosenblueth, dated July 11, August 11, and August 20, 1945.

By early September, Pitts's application for a Guggenheim Fellowship was in, and Moe asked Wiener to repeat for the committee what he had told him about Pitts.[84] Wiener wrote a letter of recommendation:

> [W]e consider Pitts not merely as a promising young man but as one of the really significant scientists of the country. His obtaining the Doctor's degree is purely a matter of formality. We are counting on incorporating him permanently in our department and regard him as one of us for the future. These accounts speak louder than any words I could say but if you prefer an expressed verbal estimate of his ability I should like to tell you that he is without question the strongest young scientist whom I have ever met. With his broad basis of knowledge both in the physiological sciences and in mathematics he occupies a place which is not filled by any other person. I should be extremely astonished if he does not prove to be one of the two or three most important scientists of his generation not merely in America but in the world at large.[85]

After this remarkable statement about Pitts's potential as a scientist, Wiener did not resist making what McCulloch would have called a "paternalistic" declaration:

> [Pitts] is a good deal of a youngster. Fundamentally he is objective and intelligent in his relationships with people but there [are] some edges which discipline will smooth out. These do not give me the slightest concern in the long run.[86]

In spite of Wiener's professed optimism about smoothing out Pitts's edges, this remark indicated that already in the summer of 1945, Wiener's attitude towards Pitts was starting to resemble the attitude that Rashevsky had had towards him.

Of course, Pitts got the Guggenheim Fellowship, and "with flying colors."[87] He planned to go down to Mexico to do experiments with Rosenblueth,[88] and he and Wiener continued working on randomly connected networks. In October, Wiener described their work to a friend:

> Walter [Pitts] and I have made substantial progress in the theory of random nets of switching devices and find that we are really working in very essential parts of the theory of state of liquids and gases.[89]

Wiener's mention of the "theory of state of liquids and gases" was his way of connecting random nets and thermodynamics.

---

[84] Letter by Moe to Wiener, dated September 4, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[85] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[86] Letter by Wiener to Moe, dated September 6, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[87] Letter by Wiener to de Santillana, dated October 16, 1945. Norbert Wiener Papers, Box 4 ff. 69.
[88] Letters by Wiener to Rosenblueth, dated June 21 and July 11, 1945. Norbert Wiener Papers, Box 4, ff. 68.
[89] Letter by Wiener to de Santillana, dated October 16, 1945. Norbert Wiener Papers, Box 4, ff. 69.

As to von Neumann's move to MIT, von Neumann visited Wiener over the summer to conduct negotiations, and Wiener thought the appointment was "in the bag."[90] But when Princeton's Institute for Advanced Studied (IAS) promised von Neumann support for building his own computer, von Neumann decided to stay at IAS. According to von Neumann, the proposed IAS computer was conceived "in a purely scientific spirit":

> We propose to construct a very high speed, automatic, electronic computer which will be used by the Institute to explore new techniques to solve partial differential equations, to advance the theories of fluid dynamics, turbulence, certain phases of molecular theory, etc.[91]

Goldstine had moved to IAS to be von Neumann's right hand in designing the computer. The following year, Bigelow also "joined the Electronic Computer Group directed by von Neumann at IAS, where Bigelow was head of the experimental group."[92]

Even without von Neumann's move to MIT, the numerous exchanges and visits between him, McCulloch, Pitts, and Wiener—which took place during 1945—left their mark. By the end of the year, Wiener observed that they agreed brains and computers were similar:

> [I]n conjunction with Walter and von Neumann, I have very definite ideas as to the relation between computing machines and the brain, mainly that both are in essence switching devices where the conjunction of the particular complexities of open and closed channels offers new channels to be open or closed as the case may be.[93]

Their claim was not yet that the brain is a computer, though they later put it that way. After all, it was McCulloch-Pitts nets—which were developed as a theory of mind in terms of neural-like mechanisms—that had provided a crucial tool for understanding and designing the circuits of digital computers.

The principal claim of Wiener and friends was that to understand the mental functions of the brain as well as the computations of the computer, both devices had to be described by the same conceptual and mathematical tools. Both employed the "binary system." Both were functionally organized into input "organs," associative "organs," and output "organs." Both were made out of

---

[90] Letters by Wiener to Rosenblueth, dated June 21, July 11, and August 11, 1945. Norbert Wiener Papers Box 4, ff. 68.
[91] Letter by Von Neumann to Hildebrandt, dated November 27, 1945, Papers of John von Neumann, Box 4, folder 1.
[92] Julian Bigelow's Vita, undated, Papers of John von Neumann, Box 2 ff. 7.
[93] Letter by Wiener to de Santillana, dated October 16, 1945. Norbert Wiener Papers, Box 4, ff. 69.

switching elements whose activities could be described either by Boolean algebra (for small circuits of known connectivity) or by statistical mechanical methods (for large circuits of randomly connected units). Much remained to be done to understand brains and computers, but these authors believed that the sciences of brains and computers belonged together.

6 THE NEW SCIENCE OF BRAINS AND MACHINES, 1946

> [W]orking in our shirt sleeves for days on end at every meeting, morning, lunch, afternoon, cocktails, supper and evening, we were unable to behave in a familiar, friendly, or even civil manner. The first five meetings were intolerable. Some participants left in tears, never to return. We tried some sessions with and some without recording, but nothing was printable. The smoke, the noise, the smell of battle were not printable (McCulloch 1974, pp. 40-41).

6.1 The First Macy Meeting

When McCulloch and Fremont-Smith met in the summer of 1945, they talked about organizing a meeting "on mechanisms underlying purposive behavior," sponsored by the Macy Foundation. They saw this topic as laying at the interface between physiology, psychology, and sociology. After their discussion, McCulloch put together a list of "the group in electrophysiology who would profit most by the meeting."[1] McCulloch and Fremont-Smith met again in the winter 1945-6 and discussed whom to invite. By January 1946, Gregory Bateson had been involved to cover the sociology side. McCulloch and Bateson made a list of social scientists and submitted it to Fremont-Smith.[2] McCulloch later wrote he had a "fairly free hand and excellent advisors in gathering the group."[3] The title of the conference was "Feedback Mechanisms and Circular Causal Systems in Biology and the Social Sciences."

According to McCulloch, he and Fremont-Smith agreed to invite at least two participants from each field: two mathematicians, two neurophysiologists, two neuroanatomists, two psychologists, two engineers, two psychiatrists, etc. This was to insure that at least one member of the audience knew the jargon of the speaker. There should be no more than 25 people, which remained a cap to any future Macy meetings. "The group must be kept small," McCulloch wrote, "for the sake of good discussion."[4] The list had to satisfy two goals: "first, to cover the necessary variety of scientific endeavors, and second, to include men from diverse institutions."[5] The list included Wiener, von Neumann, and Pitts for "Mathematical Engineering"; Rosenblueth, Lorente de Nó, and Ralph Gerard for neurophysiology;

[1] Letter by McCulloch to Fremont-Smith, dated August 27, 1945. Warren S. McCulloch Papers, ff. Fremont-Smith.
[2] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.
[3] McCulloch 1974, pp. 40-41.
[4] Letter by McCulloch to von Neumann, dated February 22, 1946. Warren S. McCulloch Papers.
[5] Letter by McCulloch to Wiener, dated February 20, 1946. Norbert Wiener Papers, Box 5, ff. 70.

McCulloch, Lawrence Kubie, and Hank Bosum for psychiatry; Gregory Bateson, Margaret Mead, and Clyde Kluckholm for sociology; Molly Harrower, Donald Marquis, and Heinrich Kleuver for psychology; and "half a dozen others from various fields not well known to" McCulloch.[6] The meeting would take place on March 8th and 9th, Fremont-Smith would participate, and McCulloch would be the chair.[7]

McCulloch had a clear vision of what the meeting should accomplish and proposed Fremont-Smith an appropriately tailored schedule of topics. First, the "formal, logical, or go-no-go aspect" had to be stated, which could best be done by mathematicians and engineers.[8] In the final schedule, this translated into von Neumann making "as concise a statement on computing machines as will serve our purpose."[9] After the "formal aspect," the "final, or purposive, aspect" could be formulated.[10] This task was assigned to Wiener, who had to introduce "the problem of goal-seeking devices," including "inverse feedback" and "all varieties of servo mechanisms which may be appropriate."[11] After both the formal and final aspects, McCulloch thought there should "obviously" follow a neurophysiologist, who should discuss how the formal and final aspects apply to the problems of nervous activity.[12] The rest of the meeting could expand the use of those notions into other areas:

> If we can get this much [i.e. final aspect, formal aspect, and neurophysiological applications] clear, I believe we will be ready to attempt formulation in wider fields, and I am sure that the remainder of the group will all have profited to this extent, that they will have clearly in mind those ideas which may be appropriate to their own problems.[13]

Fremont-Smith accepted McCulloch's schedule and on February 8th, he wrote a formal invitation to the individuals on the list:

> Developments of the last few years in the field of mathematics and engineering, with special reference to the principles underlying computing machines and target seeking devices, when taken together with advances in the field of neurophysiology, promise to throw light upon the

[6] Letter by McCulloch to Pitts, dated February 2, 1946. Warren S. McCulloch Papers, ff. Pitts.
[7] McCulloch 1974, pp. 40-41.
[8] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.
[9] Letter by McCulloch to Wiener, dated February 8, 1946. Warren S. McCulloch Papers.
[10] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.
[11] Letter by McCulloch to Wiener, dated February 8, 1946. Warren S. McCulloch Papers.
[12] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.
[13] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.

> mechanisms underlying self-correcting and purposeful behavior of individuals and of groups. This frame of reference should be of interest to those working in the fields of psychology, psychiatry, anthropology and sociology.
>
> The conference is called for informal discussion and exchange of ideas with a view to re-examining current viewpoints and in the hope of developing more fruitful concepts.[14]

In his invitation, Fremont-Smith added that at the meeting there would be no formal presentations. McCulloch would circulate the "agenda" in advance of the meeting and designate a member of the group to informally introduce each topic.

Bateson and McCulloch divided the list of participants between them, and wrote to invite those they knew best and explain them the nature of the meeting.[15] Needless to say, McCulloch was in charge of, among others, von Neumann, Wiener, and Pitts.

At that time, Pitts was applying for a Junior Fellowship at Harvard. McCulloch wrote him a letter of recommendation, asked Wiener to do the same, and asked Wiener to ask von Neumann to do the same. They complied. Wiener explained von Neumann that after three years of Harvard Fellowship, "there will be no trouble in finding [Pitts] the academic job he wants. As a matter of fact we have our hooks into him here."[16] In spite of Wiener's endorsement of Pitts's candidacy, McCulloch may have sensed that Wiener was becoming uneasy with Pitts's undisciplined ways. In announcing Wiener and Pitts the upcoming Macy meeting, McCulloch took the opportunity to act, as Rashevsky would have said, as the psychiatrist of the group.

To ease the strain between Pitts and Wiener, McCulloch implemented a subtle plan. He sent Pitts a copy of the letter of recommendation he had written for the Harvard fellowship:

> I have nothing but good to say of him.
>
> I have known him well for four years as independent intellect, collaborator, personal friend and home-guest. Physically he is unimpressive but robust. Socially he has outgrown his intolerance of our relative stupidity and of those technical difficulties which render empirical procedures tedious or ineffectual. Humanly, as well as intellectually, he has become a charming companion. Men of [Bertrand] Russell's age and wit find him congenial. To my children he is a second Lewis Carroll.

---

[14] Letter by Fremont-Smith to McCulloch, dated February 8, 1945. Warren S. McCulloch Papers.
[15] Letter by McCulloch to Fremont-Smith, dated January 23, 1946. Warren S. McCulloch Papers, ff. Fremont-Smith.
[16] Letter by Wiener to von Neumann, dated February 7, 1946. Norbert Wiener Papers, Box 5, ff. 70.

> His ability in mathematics I can not fully gauge. In spite of his extreme youth, the scope and power of his intellect is at once apparent in converse with men like Norbert Wiener and John von Neumann. They still know more mathematics, but obviously regard him as at least an equal.
>
> I speak with more assurance of his ability in neurophysiology. He grasps details rapidly and is frequently first to formulate problems to which he applies mathematics or creates it as required—and this he does without losing us by the wayside. A case in point is his outline of the work he has now in hand, namely, the development of statistical theory to cope with problems of functional organization of cerebral cortex. His prospectus was so clear, even to the non-mathematical minded, that it won instant support from biologists. Its theoretical requirements of the experimenter are so defined and practicable that we are now designing apparatus to make the necessary measurements. No theorist has done so much for us heretofore.
>
> His character is perhaps more important. Come what may, he will prosecute science to the best of his no-mean ability. Certainly money will not seduce, security spoil or honor deceive him and he will not undertake what he can not or will not perform.
>
> Would I had [*sic*] him with me always.[17]

In sending Pitts a copy of the letter, McCulloch assigned Pitts a specific task:

> [The letter] should be shown to Wiener and read carefully by you, not as flattery but because I believe it contains the right and truthful answers to the unasked questions. Note especially, emphasis on increasing humility, exemplified ability with names for cross check if desired, assertion of developed scientific drive, incorruptibility and, above all, will not undertake and fail to perform.[18]

McCulloch said he was being "right and truthful," but he knew his statements had a wishful component. He prodded Pitts to make his statements come true: "Remember years hence, when you must write, to consider these points well."[19] Incidentally, Pitts may have been too unorthodox a candidate for Harvard, for in spite of the powerful backing he had, there is no evidence that he got the Fellowship.

We don't know if Wiener felt reassured by McCulloch's statements about Pitts, but he liked McCulloch's agenda for the Macy meeting. McCulloch had put him and his friend von Neumann in charge of the theoretical foundations. Not only was Wiener to introduce teleological mechanisms to the audience, but McCulloch also added specific requests. He asked Wiener to help make crucial distinctions related to feedback and control, and to explain what he saw as "the more pressing difficulties involved in working out the laws for things of our own size." Wiener was also to discuss the problem of mechanical

[17] Letter by McCulloch to Crane Brinton, dated February 1, 1946. Warren S. McCulloch Papers papers, ff Pitts.
[18] Letter by McCulloch to Pitts, dated February 2, 1946. Warren S. McCulloch Papers, ff Pitts.
[19] Letter by McCulloch to Pitts, dated February 2, 1946. Warren S. McCulloch Papers, ff Pitts.

prediction, at least to the extent of "indicating that there is an optimum prediction to be derived from time series."[20]

In McCulloch's mind, the point about prediction was supposed to counter "the old argument that social sciences are necessarily historical and not experimental," which McCulloch feared might "rear its ugly head." McCulloch was not very hopeful about the contribution of the social scientists:

> On the second day we will, in large part, be listeners to attempts to formulate some rather nebulous problems in several chaotic fields, after which we will do our best to make such order of them as we can. It should be fun.[21]

McCulloch's detailed requests to Wiener were probably partially meant to make him feel indispensable. Wiener was a distinguished and self-centered mathematician, but he also tended to depression and liked to be reassured about the value of his work. McCulloch closed his letter by further cajoling Wiener:

> I know you frequently wonder how useful you and your ideas actually are. Remember that without you this conference would never have occurred, and that it is but one of the many examples of the way in which ideas to which you have been the best contributor are making themselves felt and gaining wide acceptance throughout all fields of scientific endeavour.[22]

McCulloch's letter to Wiener should be read by Pitts too, McCulloch wrote Pitts, so Pitts would be aware of McCulloch's agenda. McCulloch added a specific request to Pitts: "It may be a thankless task but I look principally to you to supply the hydra of pseudo theory, no matter how many heads it sprouts at once."[23]

Wiener wrote von Neumann saying that "this is our great opportunity to present our point of view," so the two of them should get together before the meeting to coordinate their talks.[24] In passing, Wiener noted that "Pitts thinks that he has a way to try out Freudian psychology with some of our ideas. In my opinion it is very promising."[25] Then Wiener replied to McCulloch's agenda by telling McCulloch

[20] Letter by McCulloch to Wiener, dated February 8, 1946. Warren S. McCulloch Papers.
[21] Letter by McCulloch to Wiener, dated February 8, 1946. Warren S. McCulloch Papers.
[22] Letter by McCulloch to Wiener, dated February 8, 1946. Warren S. McCulloch Papers.
[23] Letter by McCulloch to Pitts, dated February 12, 1946. Warren S. McCulloch Papers, ff Pitts.
[24] Letter by Wiener to von Neumann, dated February 5, 1946. Norbert Wiener Papers, Box 5, ff 70.
[25] Lett by Wiener to von Neumann, dated February 5, 1946. Norbert Wiener Papers, Box 5, ff 70.

that "[t]his meeting is going to be a big thing for us and our cause," and promised to coordinate his "part" with those of von Neumann, Pitts, and Rosenblueth.[26]

Von Neumann liked the plan too, and wrote McCulloch to thank him for the invitation. In his letter, von Neumann proposed to invite Kurt Gödel to the Macy Meeting.[27] McCulloch agreed to invite Gödel, and took this opportunity to air a project he was currently working on. He wanted to write a book on the logic of McCulloch-Pitts nets, which he saw as useful to both philosophers and electrical engineers.[28] Curiously, McCulloch wanted to model his book after Wittgenstein's *Tractatus Logico-Philosophicus*:

> [I]t is my feeling that Godel [*sic*] would be … valuable [to have at the Macy Meeting] at the moment on account of the incredible lucidity and simplicity of his "Russelized" mind. Moreover, I have a personal axe to grind with him and I am eager to get to know him personally in the near future. I have in mind to write "Tractatus Neurologico Logicus" with a Witchensteinian [*sic*] flavor. In part to clear up questions of signals and signs for the sake of somatics, but far more to crystallize a simple terminology for an electrical engineer and mathematical philosopher. It is my feeling that the English text should be parallel by a presentation in symbolic logic, and Wiener and I both would like to see it, not translated, but—next best—redone in Russian and Chinese so that in years to come, we will be able to understand one another as easily as the language will permit. It will be up to Wiener to locate the proper man for these two languages and we both want to see Godel join us for the symbolic formulation. It will, of course, be most crucial as, if it proves adequate, we can doubtless all resort to it regardless of our own native tongues.[29]

McCulloch wrote only a few fragments pertaining to his "Tractatus," which can be found in the Warren S. McCulloch Papers. Upon McCulloch's request, Fremont-Smith invited Gödel to the meeting, but Gödel declined to attend.[30]

Von Neumann wrote back to McCulloch to tell him that "[t]he more the date of our meeting approaches the more intently am I looking forward to it. I am completely convinced that you have initiated a very fruitful and interesting thing." Von Neumann also made a memorable remark on

[26] Letter by Wiener to McCulloch, dated February 15, 1946. Warren S. McCulloch Papers.
[27] Letter by von Neumann to McCulloch, dated February 20, 1946. Warren S. McCulloch Papers.
[28] McCulloch wrote only a few fragments pertaining to this project, which can be found in the Warren S. McCulloch Papers.
[29] Letter by McCulloch to von Neumann, dated February 22, 1946. Warren S. McCulloch Papers.
[30] Letter by Fremont-Smith to von Neumann, dated February 28, 1946. Warren S. McCulloch Papers.

McCulloch's project: "Your plan to write a 'Tractatus Neurologico Logicus' strikes me as a very interesting one. In fact, I think that your idea is deeper and sounder than Wittgenstein's."[31]

From a report written by McCulloch, we can reconstruct some of the highlights of the meeting. In the morning of March 8th, Von Neumann started off by introducing computing machines, distinguishing between digital and analog types. He pointed out that digital machines are superior to analog ones because they can extend their precision indefinitely by increasing the number of their components so as to represent more digits. He emphasized the universality (in Turing's sense) of digital computers. Von Neumann also distinguished two types of computer memory, one consisting of what McCulloch called "reverberation in a circuit composed of relays," the other consisting of a "locking of relays into particular positions."[32]

Von Neumann was followed by Lorente de Nó, who "described the properties of neurons as relays in the computing machine of the nervous system." In discussion, Gerard and another participant objected to a purely digital characterization of the nervous system.

In the afternoon, Wiener introduced "goal-seeking devices" by describing automata such as Watts's governor and the notion of positive and negative feedback. According to McCulloch, Wiener described Watts's governor as something that took:

> …cognizance of the world about it and of its own performance and operated so as to reduce the discrepancy between its intended performance and its actual performance. From this he extended the concept to reflexes and to all purposive activity by indicating that one merely needed to supply such a circuit with appropriate receptors and effectors. Thusly they became goal-seeking devices, of which he mentioned several, more particularly those which had built into them computing devices, some of which might so base their action on previous information as to guess the future.[33]

Wiener was followed by Rosenblueth, who used feedback terminology to describe "ordinary reflexes and homeostasis of blood pressure, maintenance of respiration, etc."[34]

---

[31] Letter by von Neumann to McCulloch, dated February 25, 1946. Warren S. McCulloch Papers.
[32] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers.
[33] Ibid.
[34] Ibid.

The rest of the meeting was mostly devoted to group discussions, which were perhaps less smooth than McCulloch had anticipated. The only person who seemed to understand everybody was Pitts. As to the others:

> [E]veryone who tried to speak was challenged again and again for his obscurity. I can still remember Norbert [Wiener] in a loud voice pleading or commanding; "May I finish my sentence?" and hearing his noisy antagonist, who was pointing at me or at Frank [Fremont-Smith], shouting: "Don't stop me when I am interrupting." Margaret Mead records that in the heat of the battle she broke a tooth and did not even notice it until after the meeting. We finally learned that every scientist is a layman outside his discipline and that he must be addressed as such.[35]

In spite of the communication difficulties, most of the participants deemed the meeting a success, and planned to have another one in October of the same year. They also decided to devote a separate meeting to teleological mechanisms in sociology.[36] Rosenblueth did express the concern that the scheme of the Macy Meeting might "peter out because of the lack of concrete material, repetition, and too much general verbiage," but he also told McCulloch that he and Wiener were enthusiastic about it.[37]

6.2 The Next Generation

With a new science in the making, a new generation of scientists needed to be trained. A new entry in the group was a young student of Wiener, Oliver Selfridge. After Lettvin returned from the war, where he was a psychiatrist for the Army, Lettvin went back to Boston. For some time, Lettvin, Selfridge, and Pitts shared the same apartment in Boston.[38]

Pitts was still working with Wiener on "nerve networks and multiple prediction."[39] Wiener expected Pitts to write his Ph.D. thesis on random nets, but Pitts tended to get distracted. Smalheiser tells of an adventure that saw Lettvin and Pitts as aspiring screenwriters in Hollywood, which must have occurred around the spring of 1946:

---

[35] McCulloch 1974, pp. 40-41.
[36] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers.
[37] Letter by Rosenblueth to McCulloch, dated July 8, 1946. Warren S. McCulloch Papers, ff. Rosenblueth.
[38] Smalheiser 2000, p. 221
[39] Letter by Wiener to Rosenblueth, dated January 24, 1946. Norbert Wiener Papers.

> The two had written a play, *The Sixth Kingdom*. They had showed it to W.H. Auden, who wrote them a letter of introduction to Christopher Isherwood. Just before arriving in Los Angeles, Pitts had fallen off a mountain edge and had fractured several vertebrae. He lay in a body cast for eight weeks, while Lettvin got a job as assistant writer at Warner Brothers and was assigned to work on *Rebel Without a Cause*. Pitts and Lettvin suggested putting *Darkness at Noon* into an American gangster setting, and brainstormed the idea until Jack Warner squashed the project.[40]

After recovering, Pitts went to Mexico to work with Rosenblueth.[41] The Hollywood adventure did not diminish McCulloch's faith in Pitts. Soon after Pitts's climbing accident, McCulloch wrote Pitts about work they were doing together. McCulloch believed he had found "arithmetical or algebraic errors" in a statistical method Pitts had developed, and was sending it to Pitts "for correction."[42]

Anyone who wanted to participate in the new science of brains and machines needed serious mathematical skills.[43] For that reason, during 1946 Wiener developed "plans for trying to organize the basis of a collaborative course between Tech [i.e., MIT] and Tufts for training people in the biological end of applied mathematics."[44] By the same token, Lettvin said that Pitts and Wiener persuaded him "to register as a special student in mathematics at M.I.T."[45] Lettvin wrote Wiener a detailed letter about his qualifications and plans. As to his qualifications, Lettvin invoked his college and medical education, his work as a physician, but mainly his association with McCulloch and Pitts:

> My unofficial training … is of rather more importance. I have been connected with Dr McCulloch and his laboratory on and off for the past five years and have been in constant communication with Mr Pitts for the past six years. From these men I have learned in a general way what the great problems are in neurophysiology, especially insofar as treatment of nerve nets is concerned, and also have been following their work in such problems. But to do any original research or to be able to tie in with the work at its present stage I need a good amount of sophistication in mathematics.[46]

As to his educational plan, Lettvin proposed the following:

> a.) Several years' study in the physical sciences, especially mathematics. This is, of course, necessary in neurophysiology and psychiatry in the light of the new work done by Mr Pitts,

---

[40] Smalheiser 2000, pp. 222-223.
[41] Letter by Lettvin to Wiener, dated ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70.
[42] Letter by McCulloch to Pitts, dated April 17, 1946. Warren S. McCulloch Papers, ff Pitts.
[43] For instance, see a letter by McCulloch to Ignatius, dated November 15, 1949, Warren S. McCulloch Papers ff. Ignatius.
[44] Letter by Wiener to Rosenblueth, dated October 1, 1946. Norbert Wiener Papers, Box 5, ff. 72.
[45] Lettvin 1989b, p. 516.
[46] Letter by Lettvin to Wiener, ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70.

Dr McCulloch, and you, and the program for future research as plotted in conversation and letter…

b.) Concomitant with my studies in the physical sciences I am arranging to work with the Veteran's Administration and with Boston City Hospital so that my knowledge of the empirical and of neurology shall not lag too far behind. It will be necessary, I am sure you'll agree, that eventually clinical testing will have to be done to support one or another of the theories being brought forth, and this would best be done by someone who is not only acquainted with the theory but also with the nervous systems the theory has to deal with. I with to work myself into such a position eventually.[47]

Lettvin started his graduate studies in mathematics at MIT in the fall of 1946, but mathematics was not his forte. According to Smalheiser, he quickly "flunked out of MIT."[48]

During 1946, Wiener and Rosenblueth continued their collaboration on flutter and fibrillation of the heart. In April, Wiener wrote a letter to Fremont-Smith to explain their research, including its relevance to the new work that Pitts and McCulloch were carrying out and more generally to the topic of the Macy Meetings:

> We have already carried a series of investigations last summer and during the academic year I have developed these more. As a matter of fact I have developed them to such an extent that Dr. Rosenblueth and myself consider a detailed quantitative experimental program as possible and profitable. He is extremely eager that we do not miss this opportunity of working together as both he and I are counting on it as the subject of our communication to the Macy Institute meeting in October on feed-back and circular processes.
>
> Let me explain that the study of flutter and fibrillation is important for the purposes of our conference-
>
> (a) as an examination of a very significant kind of feed-back processes for its own sake
> (b) the heart with its anastomosing net of muscle fibers is a simple method of the brain net differing in the synapses and in the effect that an impulse coming to a place where two fibers join are above threshold strength.
>
> It is, therefore, an excellent place to try out mathematical and experimental methods which we later intend to apply to the cortex of the brain and which are now being developed for the cortex of the brain by Mr. Pitts and Dr. McCulloch in Chicago. It is certain that no real progress can be made in the study of the feed-backs of the brain and the nervous system unless we can separate from them their random background and determine what contribution this makes. It is, therefore, an essential step in the direct line of progress to carry out the proposed investigations. As I have said they will form the topic of our conference next Fall and without this possibility of cooperative work we will be hard put to it to justify our share in the next meeting.[49]

After his pitch, Wiener asked Fremont-Smith whether the Macy Foundation could support his planned trip to Mexico to work with Rosenblueth.

---

[47] Letter by Lettvin to Wiener, ca. April, 1946. Norbert Wiener Papers, Box 4, ff. 70.
[48] Smalheiser 2000, p. 222.
[49] Letter by Wiener to Fremont-Smith, dated April 25, 1946. Norbert Wiener Papers, Box 5, ff. 70.

Fremont-Smith may not have responded the way Wiener hoped, because two weeks later, Wiener threw a tantrum. He wrote to both McCulloch and Fremont-Smith asking them to withdraw his name from the participants to future Macy Meetings.[50] He alleged he didn't have enough funds to support his participation. This was not very credible, because the Macy Foundation paid for the meetings' expenses. So Wiener argued that participating to the Macy Meetings gave one "prominence," and "prominence" required expenses that Wiener could not afford:

> It is perfectly clear to me that the position of a prominent scholar requires a financial backing which I have not got. Even in the case of the meetings that you pay for there is a good deal of consultation work… the very prominence of being present at one of your meetings subjects one to a good deal of secondary expenses in correspondence, telephone calls, etc. and I am not in a position to take care of this.[51]

We must presume that McCulloch and Fremont-Smith intervened in a way that Wiener found appropriate, for Wiener—for the time being—did not withdraw from the Macy Meetings, and over the summer he went to Mexico to visit Rosenblueth.[52]

From Mexico, Wiener kept in touch with Selfridge, who was searching for "a good subject to work on." Wiener proposed that he work on a problem that he and Pitts had addressed:

> One problem that both Pitts and I have solved after a fashion and not to our satisfaction is the factoring of a positive Hermitian matrix with elements which are functions of a variable into the product of two matrices such that the elements of one have no singularity in the upper half plane while its determinant has no zeros in the upper half plane whereas the other factor satisfies the same conditions with the upper half plane as with the lower half plane. Even results in a 2 x 2 matrix should be interesting as they introduce new methods and both Walter and I are convinced that new methods are there. If you can, run into any place where you can get some of Birkhoff's early work relevant to this subject in a paper published before 1912 in Trans. Am. Math. Soc. We will send you our material as soon as we can.[53]

Selfridge's efforts in mathematics would prove more successful than Lettvin's, and he later became a pioneer of artificial intelligence research.

---

[50] Letters by Wiener to McCulloch and Fremont-Smith, dated May 10, 1946. Norbert Wiener Papers Box 5, ff. 71.
[51] Letter by Wiener to Fremont-Smith, dated May 10, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[52] Letter by Wiener to Rosenblueth, dated October 1, 1946. Norbert Wiener Papers, Box 5, ff. 72.
[53] Letter by Wiener to Oliver [Selfridge], dated June 3, 1946. Norbert Wiener Papers, Box 5, ff. 71.

In June, Pitts was planning to go back to Mexico City and to stay there until the Macy Meeting in October.[54] Pitts did not arrive in Mexico until September 21st, apparently because he had some other accident.[55] Perhaps this was the geology adventure narrated by Smalheiser:

> Pitts became convinced that there should be emeralds in the rocks of Massachusetts, and with Selfridge obtained dynamite to blast at the predicted spot. Pitts broke his arm in the ensuing explosion, though they reportedly did find evidence of emeralds.[56]

Regardless of what accident he had had, when Pitts reached Rosenblueth he was in good shape but not entirely recovered.[57]

Rosenblueth and Pitts immediately discussed plans for work, and Rosenblueth felt they had so many interesting topics that the only difficulty was the choice. For the time being they settled on tackling fibrillation. It was difficult to find a good experimental preparation with which to work, Rosenblueth said, but he had devised a scheme that he hoped would work. Pitts was also "very eager" to learn the mathematical techniques that Wiener had developed in the work on clonus he had done with Rosenblueth.[58] Wiener was "delighted to hear that things are going so well with Pitts."[59]

6.3 More Meetings

Preparations were ongoing for a new season of meetings. Bateson organized the conference on "Teleological Mechanisms in Society" that was agreed upon at the first Macy Meeting. The sociology conference, also sponsored by the Macy Foundation, took place on September 20th 1946, in New York. The goal of the conference was to discuss the bearing of circular causal systems, like those discussed at the first Macy Meeting, on "those branches of the social sciences in which quantitative methods have

[54] Letter by McCulloch to the members of the Macy Conference on Feedback Mechanisms, ca. June, 1946. Norbert Wiener Papers, Box 5, ff 71.
[55] Letter by Rosenblueth to Wiener, dated September 23, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[56] Smalheiser 2000, p. 222.
[57] Letter by Rosenblueth to Wiener, dated September 23, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[58] Letter by Rosenblueth to Wiener, dated September 23, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[59] Letter by Wiener to Rosenblueth, dated October 1, 1946. Norbert Wiener Papers, Box 5, ff. 72.

been used."[60] The program included introductory statements by the usual suspects—McCulloch (chairman), Wiener, and von Neumann—plus Bateson, followed by presentations by some prominent social scientists, namely Paul Lazarfeld, Robert Merton, Talcott Parsons, and Clyde Kluckholn.[61] According to Wiener, "The Macy meeting for the sociologists came off very well" and they planned to have another in April. But to warrant the next conference, Wiener emphasized "the need for specific results."[62]

Bateson, helped by Fremont-Smith, McCulloch, and Wiener, also organized a Conference of the New York Academy of Sciences on "Teleological Mechanisms," which took place on October 21st and 22nd in New York. The program included Wiener on "Self-Correcting and Goal-Seeking Devices and Their Breakdown"; Rosenblueth on "Biological Exemplification of Self-Correcting and Goal-Seeking Mechanisms and Their Pathology"; von Neumann on "The Logic of Signs and Signals in Computing Machines"; and McCulloch on "Summary of Theory and Extension to Gestalt Psychology."[63]

The big event of the season, however, was the second Macy Meeting on "Feedback Mechanisms and Circular Causal Systems," which took place on October 17th and 18th, also in New York. At the time they had decided to meet in October, some members of the group had expressed the fear that by October there would not be enough time for "any new exemplifications of feedback mechanisms or any new theoretical advances worth consideration."[64] Their work "can remain useful," McCulloch wrote, "only so long as it remains a mechanistic hypothesis leading to empirical investigation of postulated processes and to the creation of the mathematics required for quantitative formulation."[65] In writing the invitation to the

[60] Letter by Bateson to the members of the Macy meetings, dated June 19, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[61] Letter by Bateson to the members of the Macy meetings, dated June 19, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[62] Letter by Wiener to Rosenblueth, dated October 1, 1946. Norbert Wiener Papers, Box 5, ff. 72.
[63] Program of Conference on Teleological Mechanisms, October 21 and 22, 1946, New York Academy of Sciences. Warren S. McCulloch Papers.
[64] Letter by McCulloch to the members of the Macy Conference on Feedback Mechanisms, ca. June, 1946. Norbert Wiener Papers, Box 5, ff. 71.
[65] Ibid.

new meeting, McCulloch reassured the participants. McCulloch pointed out that he, Pitts, Wiener, and Rosenblueth had "specific items to present."[66] In other words, the second meeting was justified.

As in the case of the first Macy Meeting, McCulloch's summary tells us something of what happened.[67] In the morning of the 17th, Von Bonin opened by discussing the "Anatomical substrate of learning and motor control," followed by Theodore Schneirla on learning in several varieties of ants. Then, McCulloch gave a formal presentation on "the calculus of signals and signs." He devoted most of the presentation to simple McCulloch-Pitts circuits. When he moved to the role of spontaneously active neurons, a discussion ensued:

> I concluded by pointing out that the introduction of a spontaneously active neuron, which is exactly equivalent to the introduction of a tautology, is sufficient to convert the calculus of signals into the total calculus of signs, i.e. of propositions. And I pointed out that signs or any other things made by circuits which might at any future time be used to start signals going again in old patterns, or—what is exactly equivalent to them—modifications of synapsis by learning, which are surrogates for enduring activity in closed paths, once created or engendered embody tautologies or serve the purpose of perenially [*sic*] spontaneously active neurons. In discussion Wiener and von Neumann agreed that this was all that was necessary in computing machines for them tp [*sic*] indulge in all mathematical operations. But they disagreed inasmuch that von Neumann, seconded by Pitts, thought that some criterion of membership might replace order given to pairs by time in open nets devoid of spontaneous activity, whereas Wiener agreed with me as to the artificiality of such attempts to exclude paradoxes impossible in well-ordered sets of operations.[68]

At the September sociology meeting, the terms "field" and "Gestalt" had been used contentiously, and participants agreed to devote some time to their clarification during the second Macy Meeting. McCulloch had scheduled some time for this purpose on the afternoon of the 17th.[69] The result was another heated discussion:

> I am afraid that the afternoon of that day suffered, as the group voiced it, from too much hormonal or too little formal discourse. The word "Gestalt" came up for clarification and it was

[66] Letter by McCulloch to the members of the Macy Conference on Feedback Mechanisms, ca. June, 1946. Norbert Wiener Papers, Box 5, ff 71.
[67] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers. See also the letter by McCulloch to the members of the Conference on Circular Causation, October 1946. Warren S. McCulloch Papers.
[68] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers.
[69] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers.

> at once apparent that five members of the group thought the remaining twenty of us were abusing the term. Before we got through they had convinced us of our ignorance…
>
> The word "field" had a like freight of frenzy. We seemed uncertain as to whether it was introduced in the sense which it has in physics of a region in which bodies are acted upon, or whether it was merely hortatory, requiring that we observe wholes instead of setting up mechanistic hypotheses as to components.[70]

The clarification of "field" and "Gestalt" was postponed to the next meeting.

After these discussions, Pitts presented "the mathematics for the theory of conduction in a random net."[71] On the following day, Rosenblueth and Wiener discussed their work on "Clonus."

To clarify the notion of "Gestalt," the participants decided to invite to the next Macy Meeting Wolfgang Köhler, one of the founders of the Gestalt movement in psychology. The burden of inviting Köhler fell on Kurt Lewin, but as usual, McCulloch had specific instructions for Lewin. He wrote a synopsis that Lewin should communicate to Köhler to put him up to date on the group's ideas. This is an excellent summary of McCulloch's views at the time, in which computers are referred to as "thinking machines":

> [Y]ou had better inform [Köhler] that the group came together in order to see what biological, psychological and sociological problems might be approached with the theoretical tools which, during the war, have created thinking and purposeful machines. Concerning thinking or computing machines he should know 1) that, like the nervous system, they are composed of relays, each of which, on receipt of proper configurations of contemporaneous signals, emits a signal; 2) that the calculus of these signals is that of "significant" propositions, as Witgenstein [*sic*] uses the term; 3) that since the complexity of states of receptors vastly exceeds the possibilities of transmission, and this, in turn, the potentialities of the effectors, the information must be so coded as to reduce the number of subsequent decisions, all of which entails responses to complexes as single entities; 4) that the mode of excitation of a relay determines that it respond to coincidences, so that the formation of the complex entities is rather a product than a sum; 5) that the statistical treatment is now available for handling random nets of relays, such as are found in the cerebral cortex, most notably in association areas; 6) that such computing machines can make what can be shown to be the best prediction from a temporal series of data, whether it be "mechanical" as in a gun pointer which places its shell where the airplane will be when the shell arrives, or "biological" as in a cat who jumps to the place the mouse will be when the cat lands; 7) that the introduction of reverberating circuits into relay nets gives them a kind of active memory for which the absolute time is no longer significant, thus extending the appropriate calculus so that it is now the full calculus of actual, or temporal, propositions. To such circular activity would correspond recognition of the enduring objects of the perceived world, etc., as opposed to mere prehension of events. Among these enduring objects are those artifacts, words

[70] Report by McCulloch to the members of the Conference on Teleological Mechanisms, October 23 and 24, 1947. Warren S. McCulloch Papers.
[71] Ibid.

> etc., which operate as signs in society, and which, for their maker, serve as equivalents for circular paths wherein activity can reverberate sine diem.
>
> Concerning the other action of closed paths, when the circuit is such as to decrease or terminate that process which gave rise to the activity in the circuit, he should know that 1) each such circuit has its goal, aim or end, i.e., that intensity of the process toward which it returns the system, call it self-regulation or homeostasis; 2) that by changing one component, say the amplification, or summation, within the central nervous system, the intensity sought by the system can be set so as to be now this and now that according to conditions external to the circuit, call it a servo-mechanism; 3) that the principle disturbances, or diseases, of such system arise when the "gain," or amplification, at the frequency imposed or due to the natural period, is too great and, are oscillations in the intensity of the process; 4) that these closed paths exist within the central nervous system, through it and the body, and through both and the external world; 5) that in terms of these we can at present state the general properties of all purposive behavior, whether of animals or of machines.[72]

A synopsis similar to McCulloch's was written by Wiener for his old biologist friend Haldane. In December, Wiener received a visit from Haldane. To prepare him for the visit, Wiener updated Haldane on his biology-related work of the last few years. It's a good window into how Wiener saw his work in relation to the group. Wiener started with the "war work" on mathematical prediction that he had conducted in collaboration with Bigelow:

> In the first place in my war work I have gone into the theory of mathematical prediction... This has led to a new theory of wave filters and much other work on communication engineering and servo-mechanisms. As a result of this Arturo Rosenbluth [*sic*] of Mexico and myself have got into a good deal of work on servo-mechanisms in the nervous system and in animal behavior. I have also been very much interested in the modern ultra-rapid computing machine and the close relationship in its manner of action to the nervous system. There is a group of us who are working over this type of idea among them von Neumann, McCulloch of the University of Illinois Medical School, Rosenbluth and myself. I have a number of first rate students around among them Walter Pitts who has been doing very distinguished work on the statistical mechanics of the nervous system and its interconnections.
>
> I have been down in Mexico twice in two years working jointly with Rosenbluth on problems which have both mathematical and physical implications. I am enclosing the first fruits of our work. The paper needs substantial revision in matters that we have found out since but is I think sound in method and likewise in what we continue to be working on. Oliver Selfridge, one of my students, by the way, one of the London Selfridge's, is making very good headway in revising our work.[73]

[72] Letter by McCulloch to Lewin, dated November 15, 1946. Warren S. McCulloch Papers.
[73] Letter by Wiener to Haldane, dated December 3, 1948. Norbert Wiener Papers, Box 5, ff. 72.

Wiener concluded his synopsis by stating his plan of collaboration with Rosenblueth. Wiener would spend a quarter of his time in Mexico, and Rosenblueth a quarter of his time in Boston. To implement that plan, they obtained funding from the Rockefeller Foundation.[74]

The group's ideas were starting to catch on. Psychologists interested in learning mechanisms invited McCulloch to talk at their meeting on "neural reverberation."[75] And Wiener accepted to give a talk on "Teleology" at the session on "New Fields in Mathematics" of the Princeton Bicentennial Mathematical Meeting, which was going to take place on December 17-19. In asking Wiener for a summary of his talk, von Neumann told Wiener that von Neumann's thoughts on the topics that interested them had been developed in new directions.[76]

6.4 Von Neumann's New Thoughts on Automata

Von Neumann's new thoughts were primarily directed at how small organisms and molecules could contribute to an understanding of their topics. On November 25th, von Neumann wrote Wiener:

> A number of things I have lately learned about very small organisms (viruses and bacteriophages) and about very great molecules (proteins) seem to me exceedingly relevant … possibly even more relevant than the neurological aspect to which we have given our main attention. I think that we ought to talk about these things, and also about the various experimental techniques connected with them, which are very much in the foreground.[77]

Von Neumann added that he was very "anxious" to discuss about "teleology" with Wiener. They were going to meet at the Princeton Bicentennial, which was still almost a month away. Von Neumann apparently was too "anxious" to wait that long.

On November 29th, only four days later, von Neumann wrote Wiener a long letter, detailing his recent thoughts. He started by reviewing the ideas discussed by their group over the previous few years, noting that they mainly focused on neurology, and pointing to "difficulties" that remained:

---

[74] Letter by Wiener to Haldane, dated December 3, 1948. Norbert Wiener Papers, Box 5, ff. 72. Letter by von Neumann to Wiener no.1, dated November 29, 1946. Norbert Wiener Papers, Box 5, ff. 72.
[75] See letters and a memo from November 3, 1948. Warren S. McCulloch Papers, ff. Bunch.
[76] Letter by von Neumann to Wiener, dated November 25, 1946. Norbert Wiener Papers Box 5, ff. 72.
[77] Letter by von Neumann to Wiener, dated November 25, 1946. Norbert Wiener Papers Box 5, ff. 72.

> Our thoughts—I mean yours and Pitts' and mine—were so far mainly focused on the subject of neurology, and more specifically on the human nervous system, and there primarily on the central nervous system. Thus, in trying to understand the function of automata and the general principles governing them, we selected for prompt action the most complicated object under the sun—literally. In spite of its formidable complexity this subject has yielded very interesting information under the pressure of the efforts of Pitts and McCulloch, Pitts, Wiener and Rosenblueth. Our thinking—or at any rate mine—on the entire subject of automata would be much more muddled than it is, if these extremely bold efforts—with which I would like to put on one par the very un-neurological thesis of R. [*sic*] Turing—had not been made. Yet, I think that these successes should not blind us to the difficulties of the subject, difficulties, which, I think, stand out now just as—if not more—forbiddingly as ever.[78]

In writing about "difficulties," von Neumann was referring to what he called the "hopeless generality" of the results of Turing and McCulloch and Pitts:

> The difficulties are almost too obvious to mention: They reside in the exceptional complexity of the human nervous system, and indeed of any nervous system. What seems worth emphasizing to me is, however, that after the great positive contribution of Turing-cum-Pitts-and-McCulloch is assimilated, the situation is rather worse than better than before. Indeed, these authors have demonstrated in absolute and hopeless generality, that anything and everything Browerian can be done by an appropriate mechanism, and specifically by a neural mechanism—and that even one, definite mechanism can be "universal".[79]

By "anything and everything Browerian," von Neumann was referring to effective procedures (see Chapter 1).

Turing had shown that there are mechanisms, i.e. universal Turing Machines, which can execute any effective procedure written on their tape in the appropriate code. McCulloch and Pitts had argued that their nets, provided with a tape, could compute anything that Turing Machines could. Moreover, McCulloch and Pitts had argued that their nets offered an explanation of the mind in terms of neural-like mechanisms. From McCulloch and Pitts's theory and their version of the Church-Turing thesis, it followed that brains, like Turing Machines, performed some computation or another. This might be seen as a limitative conclusion, because it meant brains could not perform any processes other than those of Turing Machines, but von Neumann preferred to see it as the "positive result" that brains could do "anything and everything Browerian."

[78] Letter by von Neumann to Wiener no.1, dated November 29, 1946. Norbert Wiener Papers Box 5, ff. 72.
[79] Ibid.

This was still "hopelessly general" because it didn't tell anything about the specific computations performed by the brain. Von Neumann put his point another way:

> Inverting the argument: Nothing that we may know or learn about the functioning of the organism can give, without "microscopic", cytological work any clues regarding the further details of the neural mechanism. I know that this was well known to Pitts, that the "nothing" is not wholly fair, and that it should be taken with an appropriate dose of salt, but I think that you will feel with me the type of frustration that I am trying to express. (H. N. Russell used to say, or to quote, that if the astrophysicist found a general theory uniformly corroborated, his exclamation should be "Foiled again!" Since no experimenta cruces would emerge.)[80]

Von Neumann seemed to be saying that general mathematical results about computation were no shortcuts for understanding neural mechanisms: one still needed to do detailed neuroanatomical and neurophysiological work.

This is what neuroanatomists and neurophysiologists were doing all along, but to von Neumann they didn't seem to have adequate tools to make fast progress:

> After these devastatingly general and positive results one is therefore thrown back on microwork and cytology—w[h]ere one might have remained in the first place. (This "remaining there" is, of course, highly figurative in my case, who have never been there.) Yet, when we are in that field, the complexity of the subject is overawing. To understand the brain with neurological methods seems to me about as hopeful as to want to understand the ENIAC with no instrument at one's disposal that is smaller than about 2 feet across its critical organs, with no methods of intervention more delicate than playing with a fire hose (although one might fill it with kerosene or nitroglycerine instead of water) or dropping cobblestones into the circuit.[81]

The hopelessness of neurological methods was not the end of the difficulties.

Further complications included the existence of a non-digital ("analogy") hormonal system connected via feedback loops with the digital brain, the enormous number of units that make up the brain, and what von Neumann felt as a general lack of conceptual resources for understanding it:

> Besides the system is not even purely digital (i.e. neural): It is intimately connected to a very complex analogy (i.e. humoral or hormonal) system, and almost every feedback loop goes through both sectors, if not through the "outside" world (i.e. the world outside the epidermis or within the digestive system) as well. And it contains, even in its digital part, a million times more units than the ENIAC. And our intellectual possibilities relatively to it are about as good as somebodies [*sic*] vis-à-vis the ENIAC, if he has never heard of any part of arithmetic. It is true that we know a little about the syndromes of a few selected breakdowns—but that is not much.[82]

---

[80] Ibid.
[81] Ibid.
[82] Ibid.

To address at least the difficulty due to the size of the nervous system, von Neumann considered the possibility of studying smaller organisms, whose nervous systems are relatively simple. But he saw further problems there:

> If we go to lower organisms from man with 10^{10} neurons to ants with 10^6 or to some sub-animal with, say, 10^2 neurons—we lose nearly as much as we gain. As the digital (neural) part simplifies, the analogy (humoral) part gets less accessible, the typical malfunctions less known, the subject less articulate, and our possibilities of communicating with it poorer and poorer in content.[83]

Finally, von Neumann dismissed any hope that Gestalt psychology or any other theory developed independently of a study of brain mechanisms would shed much light on the problem:

> I doubt that "Gestalt" theory, or anybodies [*sic*] verbal theory will help any. The central nervous system is complicated, and therefore its attributes and characteristics have every right to be complicated. Let not our facile familiarity with it, through the medium of the subjective consciousness, fool us into any illusions in this respect.[84]

After this somber assessment of the prospects of understanding the nervous system by the methods presently available, Von Neumann admitted that his description was "intentionally exaggerated and belittling," but asked Wiener whether he didn't think there was "an element of truth in it." Von Neumann used his negative considerations as a lead for his new thoughts. He wrote that he had felt "all these doubts for the better part of a year now, and I did not talk about them, because I had no idea as to what might say in a positive direction." This was where the "very small organisms," which he mentioned in his previous letter to Wiener, became relevant.

Von Neumann felt that they needed to turn their attention to simpler systems, like cells. These systems were sufficiently complex that they were self-contained and able to self-reproduce, but they struck von Neumann as easier to understand than nervous systems. This shows that for von Neumann, the problems he and Wiener were interested in were more general than McCulloch's problem of understanding the mind, in that they covered any complex abilities organisms, such as self-reproduction.

---

[83] Ibid.
[84] Ibid.

In the remainder of his letter, Von Neumann outlined the properties of cells that he thought were relevant to his project, then promised to formulate the problem of self-reproduction in a rigorous, Turing-style framework:

> I did think a good deal about self-reproductive mechanisms. I can formulate the problem rigorously, in about the style in which Turing did it for his mechanisms. I can show that they exist in this system of concepts. I think that I understand some of the main principles that are involved. I want to fill the details and to write up these considerations in the course of the next two months. I hope to learn various things in the course of this literary exercise, in particular the number of components required for self-reproduction. My (rather uninformed) guess is in the high ten thousands or in the hundred thousands, but this is most unsafe. Besides, I am thinking in terms of components based on several arbitrary choices. At any rate, it will be necessary to produce complete write-up before much discussing is possible.[85]

This was one of von Neumann's first statements about what he later called self-reproducing automata.

Von Neumann's letter to Wiener is an important document of the direction his thoughts on automata were taking at the end of 1946. Several points are worth noticing. One is von Neumann's endorsement of McCulloch and Pitts's theory of the brain and their version of the Church-Turing thesis, according to which the brain was essentially a digital mechanism for performing computations. Another is von Neumann's belief that Turing's and McCulloch and Pitts's results did not shed much light on brain function. And a third is von Neumann's hope that by focusing on self-reproduction, it was possible to make progress in understanding automata. All these elements would later be present and further developed in von Neumann's important work on automata (von Neumann 1951, 1956, 1966) and in his book comparing *The Computer and the Brain* (von Neumann 1958).

6.5 Importance of the Computationalist Network

By 1946, the group of investigators centering around McCulloch and Wiener was functioning as a small scientific community, whose original disciplines were either neurophysiology (McCulloch and Rosenblueth), or mathematics (Wiener and von Neumann), or electrical engineering (Bigelow). The members of the group were in constant communication with each other. They visited each other and

[85] Ibid.

collaborated on their research papers, either by writing papers together or by commenting on each other's drafts. They had relatively secure sources of funding in several foundations (Macy, Rockefeller, and Guggenheim). Finally, they were training the first members of a new generation of scientists (Pitts, Lettvin, and Selfridge). In the following decade, this group had a considerable impact on the larger scientific community.

The Macy Meetings continued until 1953 (von Foerster 1950, von Foerster, Mead, and Tauber 1951, 1952, 1953, 1955). In 1948, there was another important interdisciplinary meeting that saw the involvement of McCulloch and von Neumann—the Hixon Symposium (Jeffress 1951). The Macy Meetings and the Hixon Symposium did much to popularize the modern notions of computation, information, and feedback, as well as the view that brains and computers belonged in the same science.

As far as brains were concerned, the new science proceeded in four main directions, which at the time were all intertwined. One was the theory of computation, or as von Neumann called it, automata theory. Of the group, von Neumann was the main contributor to this enterprise. His contributions on self-reproducing automata, reliable computation from unreliable components, and cellular automata were all seminal (von Neumann 1951, 1956, 1966). From this line of work came two early pioneers of artificial intelligence, John McCarthy and Marvin Minsky, both of whom were von Neumann's students at Princeton and started their careers largely as computability theorists. Minsky was then hired at MIT, where he was close to McCulloch's group for many years.

A second direction was the design of specific networks, using idealized neurons a la McCulloch and Pitts 1943, for specific mental functions. The first example was McCulloch's model of how people can have circular sets of preferences, i.e. how they can prefer A to B, B to C, and C to A (McCulloch 1945). A more influential example is McCulloch and Pitts's model of vision (McCulloch and Pitts 1947). This work led to the first wave of research into neural networks, whose most famous representative was Frank Rosenblatt (1958).

A third direction was the development of computer programs that performed mental functions. For example, around 1948 Wiener was discussing strategies for programming computers to play chess

with Claude Shannon.[86] Wiener's student Selfridge became an early champion of this method. His famous Pandemonium, an early computer program for pattern recognition, owed a lot to McCulloch and Pitts's 1947 neural nets model of vision (Selfridge 1955, 1959). According to Allen Newell and Herbert Simon, two early leaders of artificial intelligence, Selfridge's approach—which they learned while Newell was working at RAND Corporation in the mid-1950s—was the initial source of inspiration for their early work on artificial intelligence (McCorduck 1979, pp. 132-135)

Finally, there was experimental neurophysiology. Lettvin's main work was in this field. With Pitts, McCulloch, and Humberto Maturana, Lettvin interpreted signals traveling through the optic nerve of a frog as if it were an implementation of the mechanisms described in Selfridge's Pandemonium (Lettvin, Maturana, McCulloch, and Pitts 1959). Their work was a breakthrough in the interpretation of response properties of neurons, and became a model for much work in neurophysiology over the subsequent decades.

Following these lines of development in detail would be an important contribution to the history of computationalism, and more generally to the history of the sciences of mind, brain, and computation. I hope to accomplish this in future work.

[86] See Norbert Wiener Papers, Box 6, ff. 85.

7 COMPUTATIONALISM AND THE CHURCH-TURING THESIS

7.1 Introduction

In this chapter, I discuss the Church-Turing thesis (CT) and its relevance to computationalism. As we have seen in Chapter 4 and 6, CT has been invoked in discussions of computationalism ever since the origin of computationalism (McCulloch and Pitts 1943, von Neumann 1951). Since then, CT has been used in several arguments in favor of computationalism. At the same time, CT has been extensively discussed in the philosophy of mathematics and philosophy of physics literatures. There is no complete consensus on how CT should be interpreted, whether it's provable or unprovable, and whether it's true or false.

The majority of those who invoke CT in support of computationalism ignore the philosophy of mathematics and philosophy of physics literatures on CT, even though these literatures are relevant to the assessment of the relationship between CT and computationalism. This is because, for example, an argument from CT to computationalism might be based on a version of CT that is known to be false in the literature, which makes the argument unsound. This calls for a clarification of the relevance of CT to computationalism.

In the next section, I'll review the literature on CT as briefly as the present topic allows it. I'll start with what I take to be the canonical view of CT, and then I'll discuss the most important alternatives to the canonical view. I'll conclude with a cautious endorsement of a clarified version of the canonical view. In the following section, I'll discuss how CT applies to the physical world. Finally, I'll discuss the relevance of CT for computationalism, by reformulating computationalism in terms of Turing-computability. I will evaluate *a priori* arguments in favor of computationalism based on versions of CT, and reject them as unsound.

The conclusion that arguments from CT to computationalism are unsound is not new (Sieg 1994; Shapiro 1995, p. 41; Tamburrini 1997; Copeland 2000), but a thorough and systematic analysis of the

relevance of CT to computationalism remains to be given. This chapter is a step towards such an analysis.

7.2 The Church-Turing Thesis

As I pointed out in Chapter 1, the Church-Turing thesis is the following:

(CT) A function is effectively calculable if and only if it is Turing-computable.

CT is obvious and unproblematic from right to left; that is, any Turing-computable function is effectively calculable. This is because at any step of any TM computation of any TM with TM program M, we can consult M, which we can do because M is finite, and execute its next instruction, which exists by definition unless we have reached a terminal snapshot, in which case the computation halts.[1] The rest of this section is devoted to the status of CT from left to right.

7.2.1 The Canonical View: CT is True but Unprovable (Kleene 1952, § 62, § 67)

According to the canonical view, since CT connects an informal notion to a formally defined one, CT is not suitable to rigorous demonstration. However, overwhelming evidence has been accumulated in its favor. What follows is a list of the main pieces of existing evidence:

1) No counterexample has been exhibited. Every function known to be effectively calculable, and every operation for defining a function effectively from other functions, has been shown to be Turing-computable.
2) Diagonalization over TM programs, which might be expected to yield a function that is not Turing-computable, does not lead outside the class of Turing-computable functions.

[1] A few authors have objected to this direction of CT (Péter 1959, Porte 1960), but their objections have been convincingly rebutted (Mendelson 1963). At any rate, objecting to this direction of CT has no consequences pertaining to computationalism.

3) Argument from confluence. A number of authors, working with different primitive notions, defined different mathematical formalisms and proposed that the functions definable within their formalism be identified with the class of effectively calculable functions. Some of the best-known formalisms are: general recursiveness (Gödel 1934), λ-definability (Church 1932, Kleene 1935), computability (Turing 1936-7), and reckonability (Gödel 1936). These notions have been proved to be extensionally equivalent to each other, in the sense that any function that falls within one of these formally defined classes falls within all of them.[2]

4) A Turing Machine seems capable of reproducing any operation that a human being can perform while following an effective procedure (Turing's main argument for CT in his 1936-7 paper).

According to the canonical view, the above evidence is enough to accept CT as true, even though CT remains in principle unprovable.

Several alternatives to the canonical view have been offered in the literature. I'll now discuss the alternative views, which will help clarify both the canonical view and CT itself. I'll start with views that are more optimistic than the canonical one, and then I'll move to a number of more pessimistic ones.

7.2.2 Optimistic View 1: CT is True and Provable (Mendelson 1990)[3]

Mendelson has "challenged" the canonical view of CT as an unprovable thesis (Mendelson 1990, p. 230). He compares CT to a number of other "theses" connecting informal notions and formally defined ones, such as the "thesis" connecting the intuitive notion of function and the set-theoretic definition of function (as a set of ordered couples satisfying the condition that if $(x, y) \in f$ and $(x, z) \in f$, then $y = z$). Mendelson points out that these equivalences between informal and formal notions are often either intuitively perceived to be true without proof, or established by arguments that combine intuitive perception and

[2] The argument from confluence is the most popular in computer science, where CT is often formulated as the thesis that any formalization of the notion of effectively calculable functions will yield the same old class—that of the Turing-computable functions (e.g., Newell 1980).

[3] See also Shapiro 1993.

standard mathematical reasoning. Either way, they are accepted as true and not discussed as unprovable theses. According to Mendelson, so should CT. In CT's case, Mendelson submits that its truth follows from Turing's "analysis of the essential elements involved in computation" (Mendelson 1990, p. 233; Turing's analysis and his related argument for CT is listed as (4) above).

Mendelson's considerations are a useful antidote against numerous attempts to refute CT (see below). Most people who understand CT properly do find some or all the considerations in (1)-(4) above very persuasive. Nevertheless, (1)-(4) remain intuitive considerations and not mathematically rigorous proofs. Even Turing, in presenting his argument cited by Mendelson, pointed out that "all arguments that can be given [for CT] are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically" (Turing 1936-7, p. 135).

Mendelson does not claim that the left-to-right direction of CT has been proved rigorously, nor does he offer any proof. He can be interpreted as claiming that some of the intuitive considerations offered in favor of CT are so convincing that CT should be accepted as true. Under this reading, Mendelson's "challenge" to the canonical view is limited to emphasizing that CT is neither more nor less established than many other mathematical "theses" that connect informal notions to formally defined ones, "theses" whose truth is normally not questioned, and which are not even called "theses." In this respect, Mendelson is correct, and this is precisely the canonical view of CT.

Mendelson also argues that intuitive and formally defined notions can be connected by proofs, and therefore CT is provable. In this respect, Mendelson does disagree with the canonical view of CT, but Mendelson's notion of proof includes informal, intuitive arguments as proofs. This is different from the notion of proof implicit in the canonical view of CT, which accepts only formally rigorous arguments as proofs.[4] So Mendelson's considerations are compatible with the canonical view of CT after all.

[4] For a more detailed discussion and criticism along these lines of Mendelson's claim that CT is provable, see Folina 1998. Mendelson's view is also criticized by Bringsjord 1998.

7.2.3 Optimistic View 2: CT is True Because Entailed by Physical Facts (Deutsch 1985)

According to this view, CT is true in virtue of properties of the physical world. Optimistic View 2 extends CT's scope from computability by effective procedure to computability by physical process. The thesis that any function that is physically computable is Turing-computable is stronger than CT properly so called. This stronger thesis is called Physical CT, and will be discussed in section 1.3. If Physical CT is true, then CT is also true. But Physical CT is much more controversial than CT properly so called. So in this section, I will continue to address CT properly so called on its own ground, without relying on stronger but more controversial theses.

7.2.4 The Gandy-Sieg View (Gandy 1980; Sieg 2000)[5]

The Gandy-Sieg view is neither optimistic nor pessimistic. Following Gandy (1980), Sieg (1994, 1997, 2000) argues that there are two components of CT: one component pertains to computability by humans and says that a function is effectively calculable *by humans* if and only if it is Turing-computable; the other component pertains to computability by machines and says that a function is effectively calculable *by machines* if and only if it is Turing-computable. Gandy and Sieg state that effective calculability by humans and effective calculability by machines are two different notions, which require independent analyses.

Sieg finds the evidence listed in (1) through (3) inconclusive. As to (4), Sieg believes that Turing's main argument for CT pertains only to effective calculability by humans. In several publications (Sieg and Byrnes 1996; Sieg 2000), Sieg has systematized, refined, and generalized Turing's argument. He argues that Turing's argument (suitably refined) convincingly establishes CT with respect to effective calculability by humans. But according to Gandy and Sieg, "Turing's analysis makes no reference whatsoever to calculating machines" (Gandy 1980, p. 83-4; Sieg 1994, p. 92; Sieg 1997; p. 171).

As to computability by machines, Sieg (2001) follows Gandy (1980) in defining a class of systems, called Gandy machines, which are constrained by three conditions: a bound to the speed of their

[5] The Gandy-Sieg view is also endorsed by Copeland 1996; Shapiro 1993; and Shagrir 2002.

signal propagation, a bound to their size, and discreteness of their state space.[6] A computation by a Gandy machine is a sequence of state transitions of a Gandy machine such that the initial state of the machine is the input of the computation and the state of the machine at time *t* is the output of the computation at time *t*. The only essential difference between a Gandy machine and a Turing Machine is that, while a Turing Machine can only perform one operation at a time, a Gandy machine can perform a finite number of operations at a time. Since the number of operations performed by Gandy machines at any time is still finite, a Turing Machine can perform the same operations one at a time over a finite time interval. As a consequence, computations by Gandy machines can be carried out by Turing Machines: the only difference is that Turing Machines will take more steps to complete the process. Recasting Gandy's work more clearly and elegantly, Sieg proves rigorously that any function computable by a Gandy machine is Turing-computable (Sieg and Byrnes 1999; Sieg forthcoming).

To the extent that the notion of Gandy machine is a good notion of machine and to the extent that computation by Gandy machine is a good replacement for our intuitive notion of effective calculability by machine, Gandy and Sieg have proved that a function is effectively calculable by a machine if and only if it is Turing-computable, thus giving support to the part of CT pertaining to effective calculability by machines. Given the generality of Gandy machines, this is a remarkable achievement.

But the extent to which computability by Gandy machines captures our informal notion of effective calculability by machines remains unclear. On the one hand, Gandy machines have a discrete dynamics, which means that machines with continuous dynamics, such as ordinary machines as described by ordinary mathematical physics, are left out of Gandy and Sieg's analysis. And when continuous dynamics are allowed, it is not too difficult to define systems whose "computations" are analogous to computations by Gandy machines and yet are not Turing-computable (see the next section). On the other hand, even within the class of discrete machines, some of the conditions entering the definition of Gandy machines have been questioned on physical grounds (Hogarth 1994; Shagrir 2002; see the next section).

[6] Shagrir 2002 notes a further (implicit) condition on Gandy machines: the number of state transitions employed in any computation by a Gandy machine must be finite. Another implicit condition is that the state transitions must be effective (cf. Giunti 1997). Analogous conditions are also implicit in Turing's main argument for CT.

So if one takes Gandy and Sieg's route of dividing CT into two parts and identifying computations by machines with the state space trajectories of physical systems, CT is hardly settled by Gandy and Sieg's proofs.[7]

The Gandy-Sieg view has other unappealing consequences. One is that it ascribes to Church, Turing, and their immediate colleagues a misunderstanding concerning their grounds for believing CT. According to Gandy and Sieg, computability by machines was never analyzed by Turing, and was first analyzed by Gandy 1980. And yet Church, Turing, and people who knew them directly, such as Stephen Kleene and John von Neumann, all believed that Turing's 1936-7 work convincingly settled the boundaries of what can be computed by machines (for historical details, see Piccinini 2003a, p. 27). Moreover, as Shagrir (2002) points out, the modern field of computer science is predicated on the view that Turing's formalism, together with recursive functions and many other equivalent formalisms, captures the functions that are computable by machines. The Gandy-Sieg view has the revisionist consequence that computability theorists and computer scientists, who believe CT applies to machines without needing the Gandy-Sieg analysis, have misunderstood the warrant provided by Turing and others for CT.

The way out of this conundrum is to stick to the canonical view of CT. As I pointed out in Chapter 1, the original object of Church and Turing's analysis was not effective calculability by a class of devices (either humans or machines), but calculability by an effective procedure or algorithm. In assessing CT, all we need to keep in view is the intuitive notion of effective procedure. Under the canonical view, it's irrelevant what device—whether human or machine—follows or executes the procedure or algorithm. Turing's argument for CT, as well as the rest of the evidence that was listed in

[7] There are ways to maintain that Gandy and Sieg's proofs settle CT, but they are philosophically very unpalatable. One way would be to maintain that continuous systems and discrete systems that violate the definition of Gandy machines are not machines. But this simply invites the reformulation of CT without using the term "machine." Are there functions computable by physical systems that are not Turing-computable? Given some plausible interpretations of this question, as we shall see in section 7.3, the answer is positive. Another way to maintain that the Gandy-Sieg view settles CT is to maintain that CT properly so called does not apply to computability by all machines but only to computability by finite discrete machines, where "Gandy machines" is seen as a precise substitute for "finite discrete machine." But even leaving aside worries that Gandy machines are too restrictive to capture the informal notion of finite discrete machines (Shagrir 2002), this response simply invites the formulation of a new version of CT for other kinds of machines.

section 7.2.1, supports the conclusion that as long as an effective procedure is executed, what is computed is Turing-computable. As to the general notion of computability by machines, in section 7.3.2 I'll argue that none of the putative physical counterexamples to CT deserve to be described as machines that execute effective procedures (i.e., machines that effectively calculate in the original sense under discussion), hence they are not counterexamples to CT properly so called.[8]

7.2.5 Pessimistic View 1: CT is False Because Contradicted by Non-uniform Effective Procedures (Kálmar 1959)

Kálmar (1959) invites us to imagine calculating the values of a function by "arbitrary correct means."[9] The means used need not be the same for all arguments of the function. Then, Kálmar argues, there is no reason to suppose that all functions whose values are calculable in this way will be Turing-computable. Ordinary algorithms are *uniform* methods of calculation, in that the procedure is the same for all arguments of the function whose values it calculates. By contrast, Kálmar's proposed method of calculation is *non-uniform*. Even if a function is not Turing-computable, Kálmar submits, we may be able to find all its values by a non-uniform method. Therefore, he concludes, CT is implausible.

The appropriate response to Kálmar's objection is that Turing and Church were only attempting to analyze the notion of uniform method of calculation (Tamburrini 1988, pp. 53-56). Turing explicitly pointed out that, by adopting a non-uniform method, the values of a function that is not Turing-computable might be obtained (Turing 1936-7, p. 139). This reply to Kálmar's objection leaves open whether the method proposed by Kálmar deserves to be called effective. To answer this question, Kálmar must face what may be called Church's challenge.[10] Kálmar should explain how one is supposed to search for "arbitrary correct means" for calculating a function. In doing so, either Kálmar offers an effective procedure for finding "arbitrary correct means," or he does not. If he does, then we expect

[8] Also, in Chapter 8, section 6, I say more precisely what it means for a physical system to perform computations in a sense that is both nontrivial and doesn't beg the question of CT. In Chapter 10, I offer a detailed account of what it takes for a machine to execute an algorithm or a program.
[9] Kálmar's original argument is more technical and complicated, but I think this simplified formulation retains the essence of what's at stake.
[10] Adapted from Church 1956, § 8.

Kálmar's effective procedure to fall under CT, so Kálmar's method of calculation doesn't go beyond Turing-computability after all. If, instead, he gives no effective procedure, Kálmar is not talking about the notion of effective calculability that CT applies to. Either way, in the absence of any evidence that there are effective procedures for finding "arbitrary correct means," CT is safe.[11]

7.2.6 Pessimistic View 2: CT is False Because Contradicted by Non-mechanical Effective Procedures (Gödel 1965, 1972)[12]

Gödel readily accepts CT in so far as it pertains to what he calls "mechanical" procedures, i.e. procedures that require no understanding of meaning in order to be executed (Gödel 1965).[13] However, he suggests that human beings might be able to follow procedures that deserve to be called effective and yet non-mechanical, in the sense that in order to be followed, these procedures require understanding the meaning of abstract terms (e.g., "set"). Gödel offers no examples of non-mechanical procedures, but he gives two analogies. The first analogy is with the introduction of stronger and stronger axioms of infinity in set theory; the second analogy is with the transfinite iteration of the operation of adding to an axiom system the statement that the system is consistent (Gödel 1972). Both of these processes, if carried out effectively and indefinitely, would allow humans to escape the limitations of incomplete formal systems,[14] hence the limitations of Turing-computability. (Otherwise, if there were a TM program for carrying out the described processes, such a program would constitute a complete formal system, which, by Gödel's incompleteness theorems, is impossible). So, a non-mechanical effective procedure—understood by analogy as an effective way of carrying out the processes in Gödel's examples—would allow humans to compute functions that are not Turing-computable. If one such procedure exists, then CT is false.

---

[11] For a similar reply to Kálmar, see Kleene 1987, pp. 494-495. Another reply to the same effect, based on a somewhat less charitable reading of Kálmar's paper, can be found in Mendelson 1960.

[12] A similar objection is voiced by Kreisel (1972, p. 319; 1974). See Tamburrini 1988, chapter 4, and Tamburrini 1997, p. 241 for discussions and references. See also the discussion in Odifreddi 1989 of how, in general, a broad notion of constructive procedure makes room for this objection.

[13] The notion of "understanding" used by Gödel is an intuitive one, which he doesn't explain further.

[14] That is, formal systems subject to Gödel's incompleteness results (Gödel 1931).

The reply to Gödel's objection parallels the reply to Kálmar's objection. On the one hand, Turing and Church were trying to analyze effective calculability by procedures that can be executed without understanding the meaning of the symbols that are being manipulated, so Gödel's objection doesn't apply to their intended notion. On the other hand, Gödel needs to say more about why his putative "non-mechanical" procedures deserve to be called effective. And then, like Kálmar, he faces Church's challenge. Gödel should explain how one is supposed to carry out the processes required by "non-mechanical" procedures, such as "understanding the meaning of abstract terms." In doing so, either Gödel offers an effective procedure for "understanding the meaning of abstract terms," or he does not. If he does, we expect Gödel's procedures to fall under CT after all. If, instead, he gives no effective procedure, Gödel is not talking about the notion of effective calculability that CT applies to. Either way, in the absence of any evidence that there are effective procedures for "understanding the meaning of abstract terms," CT stands.[15]

7.2.7 Pessimistic View 3: CT is False Because Contradicted by Physical Facts (Hogarth 1994)

According to this view, CT is false in virtue of properties of the physical world. It stands in opposition to Optimistic View 2, according to which CT is *true* because *entailed* by physical facts. Pessimistic View 3 is based on the possibility of hypercomputers, i.e. physical systems that compute functions that are not Turing-computable. Pessimistic View 3 says that since hypercomputers are physically possible, CT is false. As it stands, this is a *non sequitur*. Hypercomputers refute CT properly so called only if they compute by following effective procedures. Hypercomputers will be discussed in section 7.3. It will turn out that all hypercomputers that have been proposed in the literature do not compute by following effective procedures, hence they do not refute CT properly so called. However, if they are physically possible they do refute Physical CT, according to which every function that is physically computable is Turing-computable.

[15] On similar grounds, Kleene argues that Gödel's putative effective but non-mechanical procedure is a "pie in the sky" that will remain "stratospheric" (Kleene 1987, pp. 493-494).

7.2.8 Other Objections to CT

Objecting to CT has become a cottage industry. Above, we discussed some of the most interesting and illuminating objections to (the left-to-right direction of) CT. There are several other objections.

Cleland (1993, 1995, 2001, 2002) contrasts "mundane" or "quotidian" procedures, such as procedures for making Hollandaise sauce or lighting a fire, to the kind of mathematical procedures analyzed by Church and Turing. She suggests that some quotidian procedures may be able to compute functions that are not Turing-computable. She gives no examples.[16]

Bringsjord (1998) considers the human ability to hear a story and tell whether it is "interesting." Bringsjord argues that ordinary computer programs (or Turing Machines) cannot determine, for any story, whether it is "interesting." He also argues that some human beings can determine which stories are interesting, for all stories. He takes this to be a counterexample to CT. He does not explain by what means people determine which stories are interesting, for all stories.

Wegner (1999) argues that although any individual computing mechanism is bound to compute only Turing-computable functions, several computing mechanisms interacting together can compute functions that are not Turing-computable. He gives no instructions on how to get interactive systems to do so.[17]

These objections are imaginative and may teach us something about the proper scope of CT. But as they stand, they do not undermine CT. All the above authors face Church's challenge. They should tell more explicitly how their proposed computing means (quotidian procedures, human abilities to determine which stories are interesting, or interactive systems) manage to compute functions that are not Turing-computable from their relevant inputs. If these authors give an effective procedure, we expect it to be Turing-computable. If they don't, then they are not talking about the notion of effective calculability that CT applies to. In the absence of explicit answers, we have no reason to abandon CT.

[16] For defenses of CT against Cleland, see Israel 2002 and Seligman 2002.
[17] For a reply to Wegner, see Prasse and Rittgen 1998.

In conclusion, with the aforementioned caveats, we should keep the canonical view of CT as an unprovable thesis for which there is overwhelming evidence: a function is effectively calculable if and only if it is Turing-computable.

7.3 Physical CT

A number of authors have interpreted CT as a thesis about physical systems. Some of them believe CT is made true by physical facts (e.g., Deutsch 1985), whereas others believe CT is made false by physical facts (Hogarth 1994). Other authors distinguish between CT properly so called and a separate thesis that relates computability to physical systems. This latter thesis has been called thesis M (by Gandy 1980) and Physical CT (e.g., by Pitowsky 1990). Of those who discuss Physical CT, some believe it's true (e.g., Gandy 1980, Wolfram 1985), whereas others believe it's false (e.g., Stannett 2003).

In keeping with the canonical view of CT, here I will adopt the distinction between CT properly so called, which pertains to calculability by effective procedure, and Physical CT, which pertains to a generalized notion of computability by physical systems. There remains the task of formulating Physical CT sufficiently precisely to warrant its assessment. A certain amount of confusion can be avoided by distinguishing at least two versions of Physical CT, a modest and a bold version, which differ in important respects.

7.3.1 Modest Physical CT

A first version of Physical CT pertains to what functions can be *computed*, in a generalized sense that need not rely on an effective procedure, by a mechanism or machine. In order to formulate it, we need to say what we mean by "computation in a generalized sense" and by "mechanism."

A mechanism in the present sense is something that has proper functions, such as the cooling function of a refrigerator or the pumping function of the heart. A computing mechanism in the present sense is a mechanism whose proper function is to obtain certain output strings of tokens (or symbols)

from certain input strings of tokens (and perhaps internal states), according to a general rule that applies to all inputs and outputs.[18] Turing Machines, digital computers, and certain kinds of neural networks are computing mechanisms in this sense.

Strictly speaking, ordinary analog computers are not computing mechanisms in this sense, because their inputs and outputs are real-valued quantities and not finite strings. In the general case, real-valued quantities cannot even be represented as finite strings. But using an analog computer to perform computations requires preparing the input and measuring the output. Any input and output of an analog computer can only be prepared or measured with finite precision, and values measured with finite precision can be (and normally are) represented by strings of symbols. Hence, by taking only values measured with finite precision as the inputs and outputs of analog computers, which needs to be done in order to use analog computers to perform computations, we can subsume analog computers under the present notion of computing mechanism.

In section 7.2, I argued that CT covers the notion of calculability by effective procedure. If Physical CT is to be broader than CT properly so called, it needs to cover more than computation by effective procedure, namely, it needs to cover any physical process that can be employed for transforming input strings into output strings. For example, Physical CT covers operations on real-valued quantities (as in analog computers), continuous dynamical processes operating on strings (as in certain kinds of connectionist computing and molecular computing), and quantum processes (as in quantum computing).

Using this generalized notion of computation, we can formulate a broader version of CT that applies to all computing mechanisms:

> (Modest Physical CT) A function is computable by a mechanism if and only if it is Turing-computable.

[18] This account of computing mechanisms will be fully justified in Chapter 10. The discussion of Physical CT does not depend on any particular account of computing mechanisms. The only exception is accounts according to which everything is a computing mechanism, under which the distinction between Modest and Bold Physical CT may be difficult to express. The view that everything is a computing mechanism will be thoroughly discussed, and rejected (in its relevant versions), in Chapter 8.

Modest Physical CT is believed by most computability theorists and computer scientists. To assess Modest Physical CT, we need to discuss different categories of computing mechanisms and the functions they can compute.

The processes of ordinary digital computers, calculators, and their components, including digital circuits (which are a kind of connectionist computing mechanism), can be exhaustively described by effective procedures, so they fall under CT properly so called. These mechanisms will be analyzed in some detail in Chapter 10.

(Non-digital) connectionist computing mechanisms have sometimes been proposed as computing mechanisms that may go beyond Turing-computability.[19] But so far as I know, no connectionist computing mechanism known to be physically constructible has been proved to compute non-Turing-computable functions.

Ordinary analog computers have been subject to rigorous mathematical analysis, to the effect that the functions they compute are computable by Turing Machines (Rubel 1989).

In recent years, two new approaches to computing have been developed. Molecular computing exploits the combinatorial power of thousands of molecules, and quantum computing exploits superposition and entanglement. Although both molecular and standard quantum computing promise to enhance the efficiency of certain computations, neither approach goes beyond Turing-computability (Harel 2000).

So there are good reasons to believe Modest Physical CT. All computing mechanisms that have been physically built compute only functions that are Turing-computable and not others. It is important to understand the exact scope of Modest Physical CT. Modest Physical CT does not entail that everything is a computing mechanism. It only says that *if* something is a computing mechanism, *then* the functions it computes are Turing-computable.

[19] Cf.: "connectionist models … may possibly even challenge the strong construal of Church's Thesis as the claim that the class of well-defined computations is exhausted by those of Turing machines" (Smolensky 1988, p. 3).

In recent years, however, several designs for hypercomputation have been proposed. Hypercomputation is computation of non-Turing-computable functions. If hypercomputation turns out to be physically possible, it refutes Modest Physical CT. So we now turn to hypercomputation and its physical possibility.

7.3.2 Hypercomputation

A *hypercomputer* is a physical system that yields the values of a function that is not Turing-computable. Several authors have argued that hypercomputers can be built (Stannett 1990, 2003), or at least that hypercomputers are consistent with current physical theories (Copeland 2000a, 2002a). Indeed, if hypercomputation is defined sufficiently loosely, then it is perfectly consistent with some physical theories.

A trivial example is a genuinely random process, such as atom decay is currently believed to be. Alan Turing already pointed out that a machine with a "random element" can do more than a Turing Machines (Turing 1950, pp. 438-439). Alonzo Church defined a mathematical notion of random sequence, and random sequences are not computable by Turing Machines (Church 1940). If we could use a physical process to generate a random sequence, then we would have physical means that go beyond what is computable by Turing Machines.

If this is all that hypercomputation amounts to, it hardly constitutes a challenge to either CT properly so called or Modest Physical CT. This is because a random process cannot be exploited to *compute* a function in any useful sense. Computing is useful in so far as we can specify in advance the function whose values are being computed for a given argument. Since we can't observe a system for an infinite amount of time, if the system is random, then we have no way to know in advance what function it is going to "compute." This makes a random process useless as a putative computing mechanism.

There are several other difficulties in using a random process as a computing mechanism: (1) we can't specify in advance which argument of f we want a value for and run the system only for that argument, like in an ordinary computing mechanism; rather, we have to wait until that value is reached by

the system's dynamical evolution; (2) we cannot repeat a computation, as we can do with ordinary computing mechanisms; (3) the function *f* "computed" by the system varies if we vary either the time *t* at which we start observing the system or the length of the time interval that we adopt for finding the outputs of the system. By contrast, an ordinary computing mechanism has a fixed starting state, which does not depend on the time or place at which it is started, a fixed clock cycle that determines the length of the relevant time intervals, and a fixed way to establish the function it computes. For example, any Turing Machine computes one and only one function, no matter when or where we start it, and when we measure its output. For these reasons, a random process cannot be considered a computing mechanism in a useful sense.[20]

The weakness of the sense in which a random process is hypercomputational motivates the distinction between a weak and a strong notion of hypercomputation. I will distinguish between spurious and genuine hypercomputers.

A *spurious* hypercomputer is a physical system, like a random process, whose output cannot be generated by a Turing Machine. It's called spurious because it cannot be used by an observer to compute arbitrary values of a previously specified function starting at an arbitrary time and place, like ordinary computing mechanisms can (given enough time and space). In the case of a random sequence, for instance, in order to compute the elements of the sequence, an observer needs both to observe the process generating the sequence after time *t*, and to know what time it is (whereas in the case of ordinary computers one does not need to know what time it is). Since spurious hypercomputers are not computing mechanisms, they are irrelevant to both CT properly so called and to Modest Physical CT. However, they are relevant to Bold Physical CT, which will be discussed in the section 7.3.3.

A *genuine* hypercomputer, instead, is a physical system that (given enough time and space) *can* be used by an observer to compute arbitrary values of at least one previously specified function *f'* that is

[20] This is not to say that a random process cannot be useful for many purposes, including helping in certain computational techniques (cf. Copeland 2002a, pp. 480-481).

not Turing-computable—e.g., the halting function for TMs—starting at an arbitrary time and place. A well-known proposal for a genuine hypercomputer has been made by Hogarth (1994).

Hogarth Machines exploit the properties of a special kind of spacetime, called Malament-Hogarth spacetime, which is physically possible in the sense of constituting a solution to Einstein's field equations for General Relativity. Malament-Hogarth spacetimes contain what may be called spacetime *edges*: regions containing an *infinite* time-like trajectory λ that can be circumvented by a *finite* time-like trajectory g. In other words, λ and g have a common origin, here called a *bifurcation*, and there is a spacetime point p on g such that λ lies entirely in p's chronological past. In addition to the operations performed by Turing Machines (TMs), Hogarth Machines exploit five further primitive operations, described by the following instructions: (1) Position yourself at a bifurcation (where both λ and g start); (2) Launch a TM along λ; (3) Pass the edge (i.e., go to point p, by which time g has circumvented λ); (4) Send a signal (from λ to g); and (5) Receive a signal (coming from λ). With the resources offered by Hogarth Machines, we can define a procedure for solving the halting problem, which is not solvable by TMs (see Chapter 1). The halting problem asks, given a TM t and an input x, does t halt on x? Exploiting the power of Hogarth Machines, a procedure for solving the halting problem can be defined as follows:

1. Prepare t with input x and add to t's instructions the instruction to send a signal (from λ to g) upon halting.
2. Position yourself at a bifurcation.
3. Launch t along λ.
4. Pass the edge while receiving a signal if there is one coming from λ.

In the finite time it takes a machine to travel through g, this procedure determines whether t halts on x. This is because by the time g has circumvented λ, which it will do by the definition of Malament-Hogarth

spacetime, the machine traveling through g will have received a signal if and only if t halts on x. Hence, the above procedure solves the halting problem for TMs.

Because of this, Hogarth (1994) claims that Hogarth Machines falsify CT. To determine whether they falsify CT properly so called, we need to determine whether the procedures followed by Hogarth Machines are *effective*, in which case Hogarth Machines would constitute a new analysis of the informal notion of effective calculability, more powerful than TMs and other standard kinds of formalisms.

Strictly speaking, Hogarth Machines do not satisfy our definition of genuine hypercomputer, because they cannot be started at an arbitrary place and time. In order to exploit the special properties of Malament-Hogarth spacetimes, Hogarth Machines need to be started at the beginning of a bifurcation. So in order to work with a Hogarth Machine, unlike an ordinary computing mechanism, one needs to know where one is and what time it is. This difficulty can be overcome by assuming for the sake of the argument that discovering where one is and what time it is are effective operations.

Hogarth Machines exploit TMs that are allowed to perform infinitely many steps along λ before terminating their computation, which may require an actually infinite memory. In so far as Hogarth Machines are like TMs, they rely on, rather than surpass, Turing's analysis of effective calculability. In so far as Hogarth Machines are allowed to perform infinitely many operations on an actually infinite memory before terminating their computation, they stand in clear violation to the intuitive notion of effective procedure that Church and Turing were trying to analyze (which, we saw in Chapter 1, requires that only finitely many steps be performed before halting). Moreover, running a Hogarth Machine requires special operations such as positioning oneself at a bifurcation, launching a TM along λ, and passing a spacetime edge. These are hardly effective operations in the sense in which writing or erasing symbols on paper are effective. But the main difference between these operations and genuine effective operations is that they cannot be replicated as often as needed; instead, they require very special spatiotemporal conditions in order to be performed. By this measure, Hogarth Machines do not constitute

a new analysis of the notion of effective calculability, and as such they do not falsify CT properly so called.[21]

Similar points apply to other proposed hypercomputers, such as infinitely accelerating TMs (Copeland 2002b), infinite time TMs (Hamkins 2002), infinitely shrinking computing mechanisms (Davies 2001), quantum computers relying on Hamiltonians with infinitely many energy levels (Kieu 2002), and machines with real-valued constants that encode infinite strings (Siegelmann 1999). All of these proposals are genuine hypercomputers, but their hypercomputational power derives from their relying on infinite means, which violate the intuitive notion of calculability by effective procedure that Church and Turing were trying to analyze. So genuine hypercomputers are consistent with CT properly so called.[22]

Genuine hypercomputers, however, are in conflict with Modest Physical CT. Modest Physical CT pertains to any physical means of generating output strings from input strings, whether or not they can be described as effective procedures. So to the extent that hypercomputers are genuine physical possibilities, they falsify Modest Physical CT.

The extent to which hypercomputers are physically possible depends on their specific design and on which notion of physical possibility is considered to be relevant. In the context of foundations of physics, consistency with a fundamental physical theory, such as General Relativity, is the criterion for physical possibility. In this respect, if we assume that an actually infinite memory is consistent with General Relativity (which is doubtful, but see Shagrir and Pitowsky 2003), then Hogarth Machines are consistent with General Relativity, hence they are physically possible and falsify Modest Physical CT. Nevertheless, even in the context of foundations of physics Hogarth Machines may not be physically constructible, in the sense that the component that is traveling through g and trying to receive a signal from the component that is traveling on λ might be destroyed by the magnitude of the signal (Earman and

[21] For a more detailed assessment of Hogarth Machines and their relation to CT, with some points of agreement with the present assessment, see Shagrir and Pitowsky 2003.
[22] This is recognized by at least some commentators (Copeland 1996, Shagrir and Pitowsky 2003).

Norton 1993). If Hogarth Machines are not physically constructible, then they don't falsify Modest Physical CT even within the context of foundations of physics.

Outside foundations of physics, Modest Physical CT is mainly relevant to computer science and computer engineering. In this context, the relevant notion of physical possibility depends on whether human engineers can construct a machine and exploit it for performing computations in the actual world. Hogarth Machines rely on such exotic properties of spacetime that, even if those properties were present in our universe (which at present we have no evidence for), it is unlikely that they could be exploited to perform computations. At the very least, they could be exploited only when they present themselves, which may happen rarely or never in the lifetime of the solar system. So Hogarth Machines are not a concern for the version of Modest Physical CT that computer scientists and engineers are interested in.

The same point applies to all designs for hypercomputers that have been proposed so far. Infinitely shrinking TMs may be consistent with Newtonian Mechanics, but they are not even consistent with current physical theories (Davies 2001). We know no method for physically constructing real-valued constants that encode infinite strings, which is necessary for certain kinds of hypercomputers (Siegelmann 1999). Similar considerations apply to other proposed hypercomputers.[23] Present proposals for hypercomputeres will not falsify the computer scientist's version of Modest Physical CT until they are shown to be physically constructible. Whether any hypercomputer is physically constructible remains an open question, and so does Modest Physical CT.

To summarize, CT properly so called does not rule out the possibility that we construct a genuine hypercomputer. It merely rules out the possibility that a hypercomputer computes a function that is not Turing-computable by following an effective procedure. By contrast, Modest Physical CT is true if and only if genuine hypercomputers are impossible. There is as yet little reason to believe that Modest Physical CT is false (at least in the sense that concerns computer scientists and engineers), but the question remains open.

[23] For a more detailed review of the physical implausibility of contemporary proposals for hypercomputation, see Cotogno 2003. For a more hopeful review of the hypercomputation literature, see Copeland 2002a.

7.3.3 Bold Physical CT

A second version of Physical CT pertains to the computational properties of all physical systems, whether or not they are computing mechanisms. As a first approximation, it can be formulated as follows:

> (Bold Physical CT) Any function whose values are generated by a physical system is Turing-computable.

Evaluating Bold Physical CT requires that we make it more precise, by saying how the functions whose values are generated by physical systems are to be identified. Once one moves away from mechanisms operating on strings into the realm of physical processes in general, it is quite difficult to formulate Bold Physical CT in a clear and meaningful way.

Most physical systems are usually mathematically represented by systems of differential equations, which give rise to a state space and a set of trajectories through that space. For any initial condition, the equations pick out the state space trajectory that the system goes through. TMs are dynamical systems in the same sense. They have a fixed starting state, but their different inputs may be used to define different initial conditions, and they have a dynamical evolution that can be represented as a trajectory through a state space.

Given this, a natural explication of Bold Physical CT is that for any physical system, there is a TM whose state transitions map onto the state transitions of that system under its ordinary dynamical description.[24] This proposal faces two related problems. First, state-variables of ordinary physical systems are generally characterized by real numbers. Hence, there are uncountably many initial conditions of a physical system. But TM tapes contain discrete strings of tokens, of which there are only countably many. Therefore, there are too few initial conditions of TMs for them to map onto initial

[24] A proposal along these lines is made by Putnam 1988.

conditions of ordinary physical systems.[25] The second problem is a direct consequence of the first. Since every initial condition gives rise to a different state space trajectory, ordinary physical systems have uncountably many state space trajectories, whereas TMs have only countably many state space trajectories. Therefore, there are too few state space trajectories of TMs for them to map onto state space trajectories of ordinary physical systems.

A way out of this problem is to individuate in any physical system some special variables called "inputs" and "outputs," which can be compared with the inputs and outputs of a Turing-computable function. Bold Physical CT can then be explicated as follows:

> For any physical system S there is a Turing-computable function f: $A^* \rightarrow B^*$ such that for all inputs/output pairs of S <i, o>, there are strings $a \in A^*$ and $b \in B^*$ such that i is of type a, o is of type b, and f(a) = b.

For this proposal to make sense, we need to explicate what it means for a physical system in general to have inputs and outputs.

In formulating Physical CT, Pitowsky uses integral values of time as the system's input, and the integral value of a system's observable at any given integral time as the output for that input (Pitowsky 1990, p. 85). Since Pitowsky is appealing here to integral values, it is natural to restrict the relevant Turing-computable functions to functions on the natural numbers (without loss of generality). Under Pitowsky's proposal, Bold Physical CT can be reformulated as follows:

> For any physical system S and observable W, there is a Turing-computable function f: $\mathbf{N} \rightarrow \mathbf{N}$ such that for all $t \in \mathbf{N}$, f(t)=W(t).

[25] A similar point is made by Cleland 1993.

This version of Bold Physical CT is falsified by any spurious hypercomputer, such as a genuinely random process (see section 7.3.2).[26]

So for Pitowsky's version of Bold Physical CT to be plausible, it should be restricted to deterministic systems. Even so, this version of Bold Physical CT raises two serious problems: First, by focusing on integral values of time and observables, this version of Bold Physical CT rules out by definitional *fiat* what many believe to be one of the most promising sources of physical hypercomputation (including spurious hypercomputation), i.e. real-valued quantities (Cleland 1993; Stannett 1990, 2003). This suggests that this version of Bold Physical CT may be too restrictive to capture relevant mathematical properties of physical phenomena. Second, under this proposal the function whose values are generated by a system varies if we vary either the time at which we start observing the system, or the time scale at which we measure the system, or the scale at which we measure our observable W. This gives the proposal a heavily *ad hoc* flavor. It seems hard to attribute any physical or philosophical significance to a function that varies so whimsically.

Another restricted version of Bold Physical CT is addressed by Pour-El and Richards. These authors raise the question of whether, when a deterministic system's initial conditions are defined by a computable real number (i.e. a real number that can be printed out by a TM), the system's dynamical evolution will always lead to states defined by computable real numbers. They show that there are field equations for which the answer is negative, though these equations seem to have little relevance to physics (Pour-El and Richards 1989).

Another possibility for explicating Bold Physical CT is to extend the classical notion of computability by defining primitive computational operations that either manipulate real-valued quantities instead of strings (Blum et al 1998), or operate on strings but rely on the exact values of real-valued constants (Siegelmann 1999). We may call functions that are computable under these extensions of computability theory *real-computable* functions. Under this proposal, Bold Physical CT goes as follows:

[26] A similar point is made by Copeland 2000, p. 28, and Stannett 2003.

Any string function that is real-computable is Turing-computable.

But under both extensions of computability theory proposed by Blum and collaborators (Blum et al. 1998) and Siegelmann (1999), *all* string functions—including all those that are not Turing-computable—are real-computable. So this formulation of Bold Physical CT is blatantly false for rigorous mathematical reasons.

In conclusion, Bold Physical CT is very problematic. It is hard to make it clear and precise in a way that has philosophical significance, and to the extent that it can be formulated clearly and precisely, it is false for at least some important classes of physical systems.

Modest and Bold Physical CT do not exhaust the questions pertaining to the relationship between computability and physical systems. We now briefly turn to other such questions.

7.3.4 Between Modest and Bold Physical CT

In the study of physical systems and the properties of their mathematical descriptions, computability is relevant in several ways that do not fit comfortably within discussions of Physical CT. I will briefly review some of them, which I think are more relevant to the philosophy of science, and more fruitful to discuss, than Bold Physical CT.

7.3.4.1 Mathematical Tractability

Given a system of equations describing a physical system, an analytic solution is a formula such that, given any initial condition of the system and any subsequent time t, the formula yields the state of the system at time t. The question of what systems can be solved analytically is not directly relevant to computability, but its answer leads to the important issue of computational approximation of physical systems, which is relevant to computability. It is well known that most systems of equations have no analytic solutions. In particular, the majority of nonlinear systems, which make up the majority of systems of equations, are not solvable analytically.

In order to study systems that are not analytically solvable, a geometrical, qualitative approach has been developed by mathematicians.[27] This approach allows mathematicians to identify important qualitative features of a system's state space (e.g., its fixed points) without solving the system analytically. Unfortunately, the geometrical approach is suitable only for relatively simple systems, in which the number of state variables can be reduced to (at most) three, one per axis of a three-dimensional space. This limitation is mainly due to the fact that (ordinary) humans are unable to visualize a space with more than three-dimensions, and hence to apply this geometrical approach to systems whose state variables cannot be reduced to less than four. Nevertheless, the development of these geometrical techniques remains a fertile area of mathematical investigation.

Overcoming the limitations of current methods for studying complex nonlinear dynamical systems, either by extending existing methods or by inventing new methods, is a current research project of many mathematicians.[28]

There is yet another way to tackle dynamical systems, whether solvable or unsolvable analytically, simple or complex. It is the use of computational methods for approximating the dynamical evolution of physical systems. The modern study of dynamical systems has exploded over the last half-century, to a large extent, thanks to the advent of digital computers. This is because computers, by offering larger and larger amounts of memory and computation speed, allow scientists to develop methods of approximation for systems of equations that are not analytically solvable, so as to study their behavior based on those approximation. The tool of computational approximation, which is one of the crucial tools in contemporary science, is what we now turn to.

7.3.4.2 Computational Approximation

There are at least two importantly different ways to approximate the behavior of a system computationally. One relies on the equations describing the dynamics of the system and on numerical

[27] For an introduction, see Strogatz 1994.
[28] Two popular books that introduce some recent, groundbreaking work in this area are Barabási 2002 and Strogatz 2003.

methods for finding successive states of the system from those equations. The other does not rely on equations but treats the dynamics of the physical system itself as discrete. I will now briefly discuss these two methods.

Given a system of equations describing a physical system, whether or not the system is analytically solvable, it may be possible to develop methods for computing approximations to the behavior of the system. Working out specific numerical methods for specific sets of equations and showing that the resulting approximations are accurate within certain error bounds is another fertile area of mathematical investigation. These numerical methods, in turn, are behind the now widespread use of most computational models in science. These models are computer programs that exploit appropriate numerical methods to compute representations of subsequent states of a system on the basis of both the equations representing the system's dynamical evolution and data representing the system's initial conditions. Models of this kind can be constructed for any system whose behavior is described by known systems of equations that can be approximated by known numerical methods. This kind of computational approximation is perhaps the most popular form of contemporary scientific modeling.[29]

A different method of computational approximation relies on a computational formalism called cellular automata. Cellular automata are lattices of cells, each of which can take a finite number of discrete states and changes state in discrete time. At any given time, the state of each cell is updated based on the state its neighboring cells have at that time. Different updating rules give rise to different cellular automata. In modeling a physical system using cellular automata, the system is spatially discretized in the sense that distinct spatial regions of the system are represented by distinct cells of a cellular automaton, and the dynamics of the system is temporally discretized in the sense that the state changes of the system's spatial regions are represented by the updating of the cells' states. The pattern generated by the cellular automaton can then be compared with the observations of subsequent states of the system, so as to evaluate the accuracy of the approximation.[30]

[29] For more on computational models in science, see Humphreys 1990 and Rohrlich 1990.
[30] For more on cellular automata as a modeling tool, see Rohrlich 1990, Hughes 1999.

The popularity and usefulness of computational approximations of physical systems, not only in physics but in many other sciences, may have been a motivating factor behind Bold Physical CT. Some authors state forms of CT according to which every physical system can be "simulated," by which they presumably mean computationally approximated in the present sense, by TMs.[31] But the question of whether every system can be computationally approximated is only superficially similar to Bold Physical CT.

The importance of computational approximation is not that it embodies some thesis about physical systems, but that it is the most flexible and powerful tool ever created for scientific modeling. An approximation may be closer or farther away from what it approximates. The way in which and the degree to which an approximation should mimic the system it models is largely a pragmatic factor, which depends on the goals of the investigators who are building the model.

If one allows computational approximations to be arbitrarily distant from the dynamical evolution of the system being approximated, then the thesis that every physical system can be computationally approximated becomes trivially true. If one is stricter about what approximations are acceptable, then that same thesis becomes nontrivial but much harder to evaluate. Formulating stricter criteria for acceptable approximations and evaluating what systems can be approximated to what degree of precision is a difficult question, which would be worthy of systematic investigation. Here, I can only make a few obvious points.

First, strictly speaking, unpredictable (e.g., non-deterministic) systems cannot be computationally approximated. A computational approximation can only indicate the possible dynamical evolutions of such systems, without indicating which path will be followed by any given system.

Second, if there are any (deterministic or non-deterministic) physical systems whose state transitions are not Turing-computable, e.g. if genuine hypercomputers are possible, then there is a strict sense in which those systems cannot be computationally approximated.

---

[31] E.g.: "Church's thesis [states] that anything that can be given a precise enough characterization as a set of steps can be simulated on a digital computer" (Searle 1992, p. 200).

Finally, as soon as the state-variables of a system are more than two and they interact nonlinearly in a sufficiently complex way, the system may exhibit chaos (in the mathematical sense). As is well known, chaotic systems are so sensitive to initial conditions that their dynamical evolution can only be computationally approximated for a relatively short time before diverging exponentially from the observed behavior of the system.

In conclusion, the extent to which physical systems can be computationally approximated depends both on the properties of physical systems and their mathematical descriptions, and on the criteria that are adopted for adequate approximation. The same computational model may count as producing adequate approximations for some modeling purposes but not for others. At any rate, on any nontrivial criteria for adequate approximation, it is far from true that every physical system can be computationally approximated.

7.4 Computationalism and CT

The ground is now clear to formulate computationalism in terms of Turing-computability and discuss the relevance of CT and Physical CT to it. Appealing to the notion of Turing-computability, computationalism can be reformulated as follows:

(C) The functions whose values are generated by brains are Turing-computable.

Given this formulation of computationalism, CT and Physical CT are potentially relevant to it, for CT and Physical CT state that certain classes of functions are Turing-computable. If the functions whose values are generated by brains belong to the relevant classes of functions, then by either CT or Physical CT, (C) follows.

Some appeals to CT are so unwarranted that they deserve to be mentioned as a sign of the pervasive confusion in the philosophical literature on the topic of computability. For instance:

> [A] standard digital computer, given only the right program, a large enough memory and sufficient time, can compute any rule-governed input-output function. That is, it can display any systematic pattern of responses to the environment whatsoever (Churchland and Churchland 1990, p. 26).[32]

If we interpret the Churchlands's "rule-governed functions" and "systematic patterns of responses" as referring to functions that are well defined within computability theory, then their conclusion that a digital computer can compute any such function is obviously false. We know from Turing's results and from standard computability theory that only countably many functions defined over strings (out of uncountably many such functions) are computable by standard digital computers. So the Churchlands's statement cannot be taken seriously.[33]

There are three ways in which serious arguments from CT to computationalism have been run. They are addressed in turn.

7.4.1 By Physical CT

Several authors have used Physical CT to argue for (C). According to Physical CT, all physically computable functions are Turing-computable. Since brains are physical systems, by Physical CT, the functions physically computed by brains are Turing-computable.[34] As I argued in section 7.3, we should distinguish between at least two importantly different versions of Physical CT. Accordingly, we need to examine two versions of the argument by Physical CT.

---

[32] Cf. also Guttenplan 1994, p. 595.

[33] For a similar dismissal of Churchland and Churchland's statement, see Copeland 1996 and 2000.

[34] Cf:

> Human beings are products of nature. They are finite systems whose behavioral responses to environmental stimuli are produced by the mechanical operation of natural forces. Thus, according to Church's thesis, human behavior ought to be simulable by Turing machine (McGee 1991, p. 118)

> [W]e have good reason to believe that the laws of physics are computable, so that we at least ought to be able to *simulate* human behavior computationally (Chalmers 1996a, p. 313).

7.4.1.1 By Modest Physical CT

Modest Physical CT does not apply to all physical systems, but only to mechanisms that perform computations. It says that, if a physical system computes, then it computes a Turing-computable function. Modest Physical CT is true if and only if genuine hypercomputers are physically impossible, which at present remains an open question. So Modest Physical CT is relatively controversial. But I argued that at the present state of our knowledge, and especially in so far as it concerns computer scientists, Modest Physical CT is at least plausible.

The version of Modest Physical CT that concerns computer scientists is also the one that concerns neuroscientists. For neuroscientists are interested in neural mechanisms, which are relatively small physical systems confined within relatively small spatiotemporal regions. There is little reason to believe that neural mechanisms have access to the exotic physical resources, such as spacetime edges and bifurcations, that are exploited in designs for hypercomputers.

If Modest Physical CT applies to brains, the resulting version of computationalism is not trivial. For Modest Physical CT applies to the functions *computed* by physical systems, hence it entails that those systems are genuine computing mechanisms, whose behaviors are computations—as opposed to non-computing mechanisms, whose behaviors are not computations. In this respect, if we could conclude that Modest Physical CT applies to brains, we would learn something substantive about them.

But Modest Physical CT says nothing about whether any particular physical system is a computing mechanism. It leaves open whether the solar system, the weather, or your brain is a computing mechanism. Whether the brain or any other physical system is a computing mechanism must be established by means other than Modest Physical CT. If we can establish that brains are computing mechanisms by other means, then Modest Physical CT applies to them, and if Modest Physical CT is true, then the functions brains compute are Turing-computable. But if we can establish that brains are computing mechanisms by other means, then we would already have established computationalism by those other means, and whether Modest Physical CT is true or not would make no difference. What

Modest Physical CT establishes (if true) is only that if brains are computing mechanisms, they are not genuine hypercomputers.

7.4.1.2 By Bold Physical CT

Bold Physical CT applies to all physical systems, and says that the functions whose values are generated (by any means) by physical systems are Turing-computable. Hence, assuming that brains are physical, Bold Physical CT does entail that the functions whose values are generated by brains are Turing-computable, which establishes (C). But this is a Pyrrhic victory. All versions of Bold Physical CT that we have examined are false for at least some important classes of physical systems. So the argument from Bold Physical CT to (C) is valid but unsound.

Even if there were a true version of Bold Physical CT that could be used to entail (C), however, this would yield only cold comfort to the computationalist. The main price of using Bold Physical CT to support computationalism is that computationalism is thereby trivialized. The original motivation for computationalism is that the notion of computation can be used to *distinguish* mind-brains from other systems (e.g., Fodor 1998). But Bold Physical CT cannot do this, because it applies indifferently to brains as well as other physical systems by virtue of their being physical. Any view of the brain derived from Bold Physical CT is not a genuine form of computationalism, according to which brains are a computing mechanism and not some other kind of mechanism. In theorizing about brains we are looking for specific explanations of neural processes and mechanisms. If computation is used in a sense that applies to any physical process, then it cannot be the basis for a specific explanation of neural processes. So anyone who wants to claim that brains are computing mechanisms in a nontrivial sense, even if she believes Bold Physical CT, must look for a more stringent version of computationalism and support it independently of Bold Physical CT. This point will be discussed at length in Chapter 8.

In conclusion, neither version of Physical CT helps the supporter of computationalism.

7.4.2 Cognition as an Effective Procedure

A straightforward way of arguing from CT to (C) is to show that cognitive processes are effective in the sense analyzed by Church and Turing. If cognition is effective in this sense, then the functions whose values are generated by brains when they perform cognitive processes fall under CT properly so called. Then, by CT itself, (C) follows.[35] This section evaluates the thesis that cognition is an effective procedure.

Several authors believe that the claim that cognition is an effective procedure is an empirical hypothesis, to be supported on *a posteriori* grounds such as the successes of AI, psychology, or linguistics.[36] This view does not concern us here. Here, I only discuss *a priori* arguments to the effect that cognition is an effective procedure.

The most explicit *a priori* argument that I know of in favor of the thesis that cognition is an effective procedure is by Judson Webb (1980, p. 236ff.). Webb introduces his argument as if it were an explication of Turing's argument for CT (ib., p. 220ff.). Webb is not alone in attributing Turing the thesis that cognition is an effective procedure,[37] but this is a misunderstanding of Turing. Turing expressed a view that is almost the opposite:

> If the untrained infant's mind is to become an intelligent one, it must acquire both discipline and initiative. So far [i.e., by discussing effective procedures] we have been considering only discipline… But discipline is certainly not enough in itself to produce intelligence. That which is required in addition we call initiative… Our task is to discover the nature of this residue as it occurs in man, and try to copy it in machines (Turing 1948, p. 21).

There is no room here for a careful reconstruction of Turing's thought on intelligence and cognition. Suffice it to say that the most recent scholarship on Turing rejects the view that he believed cognition to be an effective procedure.[38]

Regardless of what Turing thought about this matter, here is what Webb says:

---

[35] Cf: "Human cognitive processes are effective; by the [Church-Turing] thesis it follows they are recursive relations" (Nelson 1987, p. 581).
[36] Cf. Haugeland: AI is "a fairly coherent research tradition, based on a single basic idea: thinking as internal symbol manipulation" (Haugeland 1997, 25).
[37] Cleland 1993, p. 284; Shanker 1995, p. 55; Fodor 1998.
[38] See Sieg 1994, Copeland 2000. I have argued at length against this misinterpretation of Turing in Piccinini 2003a.

> To show that man is not an abstract (universal) Turing machine it would be sufficient to show that at least one of the following conditions is false:
> (i) Man is capable of only a finite number of internal (mental or physical) states $q_i \in Q$.
> (ii) Man is capable of discriminating only a finite number of external environmental states $s_i \in S$.
> (iii) Man's memory is described by a function f from pairs $<q_i, s_i>$ to Q.
> (iv) There is a finite set B of atomic human behaviors, including some which may be identified with states s_i of S, which they effect (as in printing a symbol), and each $<q_i, s_i>$ determines a unique element of B. A human (molar) behavior is comprised of a finite sequence of atomic behaviors of B, some of which have neural dimensions, while others may be molar behaviors in their own right (Webb 1980, p. 236).

Webb seems to believe these conditions obtain. He should add that for his argument to go through, at least two further implicit conditions must obtain. First, the function from pairs $<q_i, s_i>$ to B implicit in (iv) must be Turing-computable. Second, humans must be capable of going through at most finitely many memory states and atomic behaviors in a finite time. It is well known that systems that do not satisfy these further conditions can compute non-Turing-computable functions (Giunti 1997, Pitowsky and Shagrir 2002). If Webb's explicit and implicit conditions are satisfied by human brains, then their behaviors are Turing-computable.

As to (i), Webb appeals to Turing's argument that a human who is performing calculations is capable of only finitely many internal states, otherwise some of them would be arbitrarily close and would be confused (Webb 1980, p. 221). This is justified in an analysis of human calculation, where the internal states must in principle be unambiguously identified by the effective procedure followed by computing humans. In his argument, Turing makes it clear that his "internal states" should be replaceable by explicit instructions.[39] Since the instructions have finite length, they can only distinguish between finitely many states.

But this says nothing about the number of internal states humans are capable of outside the context of calculation. Ordinary mathematical descriptions of physical systems ascribe to them uncountably many states. There is no *a priori* reason to suppose that humans are different from other physical systems in this respect. In fact, theoretical neuroscientists make extensive use of ordinary

[39] This point has been emphasized by Sieg (e.g., 2000).

mathematical descriptions, which ascribe to neural mechanisms uncountably many states (Dayan and Abbott 2001).

As to (ii), Webb also attributes it to Turing. But again, Turing was only concerned with effective calculability by humans, an activity that must be describable by an effective procedure. Since effective procedures are finite, it seems plausible that they can only be used to discriminate between finitely many "environmental states" (i.e., symbols). Again, this says nothing about how many environmental states humans can discriminate outside the context of calculation.

Webb adds that (ii) "would follow from a finiteness condition in physics itself to the effect that there *were* only finitely many states of the environment there to be discriminated" (Webb 1980, p. 236). Webb, however, gives no reason to believe that such a physical condition obtains.

As to (iii), Webb makes it clear that the function f from pairs $<q_i, s_i>$ to Q should be Turing-computable. He admits the possibility that f be nondeterministic, but submits that that could be taken care of by a nondeterministic TM. This is true only if there are at most finitely many q_i and s_i, i.e. if (i) and (ii) obtains, but we've already seen that (i) and (ii) are unjustified. Webb is also skeptical that it could be "shown effectively" that there is no such Turing-computable f. If we assume that by "showing something effectively," Webb means proving something rigorously, Webb's statement is surprising. For one of Turing's greatest achievements was precisely to prove rigorously that there is no Turing-computable f for solving uncountably many problems, including the halting problem for TMs. Given CT, there is no principled difficulty in showing that a function is not Turing-computable.

The main difficulty with (iii), however, is not that f may not be Turing-computable, which there is no evidence for. The main difficulty is that Turing-computability may be simply irrelevant to "describing" human memory. The relationship between human memory mechanisms and Turing-computability can be divided into two sets of issues. One set of issues belongs with a general analysis of the relationship between computability and physical systems, which was reviewed in section 7.3. This set of issues has nothing to do with whether cognition in particular is an effective procedure, and its relevance to computationalism was already covered in section 7.4.1. The other set of issues belongs with

assessing the empirical hypothesis that the brain is a computing mechanism, which will be analyzed in the following chapters. That empirical hypothesis is not something that can be settled *a priori*.

As to (iv), Webb says, "it is … hard to imagine anything but Descartes' mechanical organisms existing at the dawn of evolution" (ib., 236), where the context makes clear that "Descartes' mechanical organisms" satisfy condition (iv). But the fact that something is "hard to imagine" hardly seems conclusive evidence for (iv).

As to the further conditions implicit in Webb's argument, they seem no more *a priori* true than those explicitly stated by Webb. In the end, Webb has offered little support for his view that cognition is an effective procedure. This, of course, is not to say that cognition is a non-effective procedure, or a non-Turing-computable procedure. It may be that procedures are just irrelevant to scientific theories of cognition. To determine the relevance of procedures, whether effective or not, to neural or psychological theories, it seems more fruitful to develop and examine empirical theories of cognition rather than arguing *a priori* about these matters.

7.4.3 Effective Procedures as a Methodological Constraint on Psychological Theories

A final argument from CT to (C) invokes a methodological constraint on psychological theories, to the effect that psychological theories should only be formulated in terms of effective procedures. If this is the case, then by CT, (C) follows. This argument is originally due to Dennett, and it has found many followers.[40] Here is Dennett's original:

> [C]larity is ensured for anything expressible in a programming language of some level. Anything thus expressible is clear; what about the converse? Is anything clear thus expressible? The AI programmer believes it, but it is not something subject to proof; *it is, or it boils down to, some version of Church's Thesis* (e.g., anything computable is Turing-machine computable). But now we can see that the supposition that there might be a non-question-begging non-mechanistic psychology gets you nothing, unless accompanied by the supposition that Church's Thesis is false. For a non-question-begging psychology will be a psychology that makes no ultimate appeals to unexplained intelligence, and that condition can be reformulated as the condition that whatever functional parts a psychology breaks its subjects into, the smallest, or most fundamental, or least sophisticated parts *must not be supposed to perform tasks or follow*

[40] Webb 1980, p. 220; Haugeland 1981, p. 2; Fodor 1981a, pp. 13-15; Pylyshyn 1984, p. 52, 109; Boden 1988, p. 259. A discussion of this argument can be found in Tamburrini 1997.

> *procedures requiring intelligence*. That condition in turn is surely strong enough to ensure that *any procedure admissible as an "ultimate" procedure in a psychological theory falls well within the intuitive boundaries of the "computable" or "effective" as these terms are presumed to be used in Church's Thesis*. The intuitively computable functions mentioned in Church's Thesis are those that "any fool can do," while the admissible atomic functions of a psychological theory are those that "presuppose *no* intelligence." If Church's Thesis is correct, then the constraints on mechanism are no more severe than the constraints against begging the question in psychology, for any psychology that stipulated atomic tasks that were "too difficult" to fall under Church's Thesis would be a theory with undischarged homunculi [fn. omitted]. So our first premise, that AI is the study of all possible modes of intelligence, is supported as much as it could be, which is *not quite* total support, in two regards. The first premise depends on two unprovable but very reasonable assumptions: that Church's Thesis is true, and that there can be, in principle, an adequate and complete psychology (Dennett 1978a, p.83; italics added).

Dennett affirms that CT yields a methodological constraint on the content of psychological theories. This is because, he says, any theory that postulates operations or procedures that are "too difficult" to fall under CT is postulating an "undischarged homunculus," that is, an unexplained intelligent process. And any psychological theory that postulates undischarged homunculi should be rejected on the grounds that it begs the question of explaining intelligence.[41]

Dennett's reference to "procedures admissible as ultimate procedures in a psychological theory" suggests that psychological theories are formulated in terms of procedures. Since what fall under CT are *effective* procedures, Dennett's argument entails that the only ingredients of psychological theories are effective procedures. As a matter of fact, with the rise of cognitive psychology, some psychologists did propose that psychological theories be formulated as effective procedures, or computer programs, for executing the behavioral tasks explained by the theories (Miller, Galanter, and Pribram 1960). This view was elaborated philosophically by Fodor (1968b), which is one of the works referred to by Dennett (1978a).

To the extent that psychological theories are or should be formulated in terms of effective procedures, Dennett's appeal to CT is well motivated. For suppose that a psychologist offered an explanation of a behavior that appeals to a non-mechanical procedure of the kind imagined by Gödel

[41] For more by Dennett on homunculi and question begging in psychology, cf. Dennett 1978c, pp. 57-9, and Dennett 1978d, 119ff. For more by Dennett on AI as study of all mechanisms (as opposed to computing mechanisms) for intelligence, see Dennett 1978b, esp. p. 112.

(1965), or to arbitrary correct means of proof like those hypothesized by Kálmar (1959). Suppose that this psychologist refused to give effective instructions for these putative procedures. Then Dennett would be justified in concluding that these putative procedures are undischarged homunculi. Such a psychological theory purports to explain a behavior by postulating an unexplained intelligent process, which begins an infinite regress of homunculi within homunculi. Gödel and Kálmar's proposals may have a legitimate role to play in philosophy of mathematics, but not in a naturalistic explanation of behavior, as a non-question-begging psychological explanation should be. In other words, any psychologist who wants to postulate effective procedures that do not fall under CT should give a mechanical explanation of how they can be followed by people. By so doing, this psychologist would falsify CT. So far, Dennett's argument is sound.

It remains to be seen the extent to which psychological theories are or should be formulated in terms of effective procedures. In Chapter 8, I argue that explaining a behavior by postulating an effective procedure (or computer program) is only one species of a larger genus. This genus is explanation of behavior by functional analysis, which consists of postulating a set of components and ascribing proper functions to those components. I will also argue that in the philosophy of psychology tradition that goes from Fodor to Dennett and beyond, explanation by appeal to effective procedure and explanation by functional analysis are conflated. One of the effects of this conflation is Dennett's conclusion that CT constitutes a methodological restriction on all psychological explanation rather than only on explanations that appeal to effective procedures (or computer programs).

Functional analyses in psychology face the same constraint against begging the question of explaining intelligence that Dennett exploits in his argument. That is to say, the components postulated by a psychological functional analysis should not contain undischarged homunculi. If a component is ascribed intelligence (or other high level cognitive abilities), this intelligence should be discharged by the lesser intelligence (or lesser degree of other high level cognitive abilities) of its components, until a level of analysis whose components have no intelligence (or other cognitive abilities) is reached. This methodological restriction on functional analyses was already formulated by Attneave (1961), who did

not mention either effective procedures or CT. He did not mention them because to the extent that psychological theories are formulated without postulating effective procedures, CT is irrelevant to them.

CT is only relevant to functional analyses that postulate effective procedures, not to other kinds of functional analyses. And *vice versa*: a psychological theory that explains behavior without postulating effective procedures does nothing by itself to falsify CT. Modest Physical CT, however, is relevant to any functional analysis that postulates a process of computation. If a psychological theory postulates a genuine hypercomputation as a psychological process, then it falsifies Modest Physical CT. If it postulates no computations at all, it is irrelevant to Modest Physical CT too. Finally, the issues of computational approximation discussed in section 7.3.4 (Between Modest and Bold Physical CT) are relevant to any psychological functional analysis. They are relevant because they are relevant to any dynamical description—they have nothing in particular to do with psychological theories. In conclusion, CT poses a methodological constraint on a species of psychological theories—those that postulate effective procedures—but poses no general constraint on psychological theories.

7.5 Conclusion

This chapter started with the Church-Turing thesis (CT) and its main interpretations and objections, and defended a clarified version of the canonical view of CT. According to the canonical view, CT connects the informal notion of effective procedure with the formal one of Turing-computability. There can be no rigorous proof of CT, but there is overwhelming evidence in its favor. I then briefly discussed the relationship between computability and physical systems.

This led to a discussion of three attempts to support computationalism *a priori* on the grounds of CT. I argued that given a proper understanding of CT, all those arguments can be seen to be unsound. CT does entail that *if* the brain follows an effective procedure, *then* that procedure is Turing-computable. And Modest Physical CT does entail that *if* the brain performs computations, *then* those computations are

Turing-computable. But neither CT nor Modest Physical CT is of any use in determining whether the brain follows effective procedures or more generally, whether it performs computations.

In this chapter, I also identified other ways in which computability is relevant to brain theory. Neural mechanisms are complex nonlinear dynamical systems *par excellance*, which lie at the frontier of what is mathematically analyzable by dynamical systems theory (see Barabási 2002; Strogatz 2003; Dayan and Abbott 2001). Because of this, methods of computational modeling are crucial to their scientific study. This is independent of whether neural mechanisms are computing mechanisms in any nontrivial sense.

Besides appeals to CT, there is another line of *a priori* argument for computationalism. According to this line, the brain is a computing mechanism because the mind is a computer program running on the brain. That the mind is a computer program is presented as a metaphysical thesis, namely as the most promising solution to the mind-body problem (Putnam 1967b). This line of argument is addressed in the next chapter.

8 COMPUTATIONAL FUNCTIONALISM

8.1 Introduction

This chapter examines the doctrine of *computational functionalism* about the mind, according to which the mind is a program that runs on the brain in the sense in which computer software runs on computer hardware. I formulate computational functionalism as a substantive, empirical thesis about neural mechanisms and their functional organization. Then I defend my formulation of computational functionalism from potential objections. While doing so, I offer the first detailed study of the history of functionalism to date. I argue that the literature on functionalism is permeated by a number of confusions, which my formulation of computational functionalism avoids.

Functionalism is usually formulated in terms of the states of a system. A state *S* of a system is said to be a functional state if and only if *S* is identified by the causal relationships that *S* bears to inputs, outputs, and other states of the system. Being in a functional state of type *S* is the same as being in a state that is caused by certain inputs of types I_1, … I_i together with other states of types S_1, … S_n and causes certain outputs of types I_{i+1}, … I_j and certain other states of types S_{n+1}, … S_m. Functionalism was originally developed by Putnam in a series of papers in the 1960s (1960, 1963, 1967a, 1967b). In those papers, Putnam applied functionalism to "mental states," which he assumed to be the reference of scientific psychological descriptions. The resulting doctrine, that mental states are identified by their mutual causal relationships and by their relationships with their inputs and outputs, is called functionalism about minds.

Functionalism has also been applied in domains other than psychology, e.g. in biology (Rosenberg 1978, 1985) and the social sciences (Schmaus 1999). From now on, we'll restrict our attention to functionalism about minds; when I write functionalism *tout court*, functionalism about minds is intended.

For our purposes, functionalism must be kept as general as possible. It is particularly important that, in its most general formulation, functionalism does not restrict the type of description used to identify a system. For example, functional descriptions might be given in the form of box-and-arrow diagrams that are popular in many branches of psychology, or circuit diagrams that are popular in the connectionist literature, or systems of differential equations (which determine a space of states connected by state-space trajectories), or computer programs, or any other description that scientists of the mind find useful. Moreover, functional descriptions need not be restricted to covering only one level of organization of a system (e.g., neurons, or networks, or neural systems, or organisms): they may span all levels or organization.[1]

In contrast to this liberal use of functional descriptions, Putnam (1967a) proposed that the functional descriptions of mental states be given in terms of Turing Machine (TM) programs. Since then, functionalism about the mind has often been presented as the thesis that mental states are functional states defined by a TM program (e.g., Kim 1996, Rey 1997). This specific version of functionalism, from now on called computational functionalism, entails the computationalist thesis that the brain is a computing mechanism.[2] If computational functionalism is correct, then there is a sense in which the mind-brain is a computing mechanism: the mind is to the brain as the TM program is to its hardware realization, and *vice versa*.[3] This chapter discusses both the relationship between the general doctrine of functionalism and the specific doctrine of computational functionalism.

In discussing functionalism about minds, two theses must be kept distinct. First, the functionalist thesis that mental states are functional states. Second, the functional-computational thesis that functional states corresponding to mental states are states of certain TM programs. The combination of these two

[1] There is a large literature on descriptions of mechanisms in the psychological and biological sciences, including multi-level descriptions; e.g., see Bechtel and Richardson 1993, Machamer *et al*. 2000, Bechtel 2001, Craver 2001.
[2] That is, assuming a minimal materialist commitment, such as token identity between mental and neural states.
[3] In the functionalist literature, the relationship between an ordinary computer program and the hardware on which it runs is usually assimilated to the relationship between a TM program and its hardware realization (see section 8.4 below). For now, I am going along with this assumption, but in Chapter 10 I will reject both this move and the account of the identity conditions of computing mechanisms that results from it. In section 3 of this chapter, when formulating computational functionalism, I do so in terms of stored-program computers, not TMs.

theses amounts to computational functionalism. The first goal of this chapter is to elucidate the two components of computational functionalism and their mutual relations. The general doctrine of functionalism is logically weaker then its computational variety. That is, one can maintain functionalism about the mind while rejecting the thesis that the mind is a TM program. The arguments for functionalism that are found in the literature will be discussed, finding that they provide no reason to endorse the functional-computational thesis. Hence, endorsing functionalism is no reason to endorse *computational* functionalism.

At the same time, I'll argue that in the literature, computational functionalism has become improperly entangled with two other theses, namely the thesis that psychological theories are computer programs and the thesis that everything can be described as a TM. These other theses, which are problematic in their own right, gave many philosophers the false impression that computational functionalism was naturally supported by general considerations about the form of psychological descriptions.

Before moving on to my task, I should make clear what this chapter is *not* about.

When Putnam proposed functionalism about minds, he had in mind states ascribed to individuals by *scientific* theories of mind. Functionalism has also been applied to states ascribed to individuals by so-called *folk* psychological theories, that is, psychological descriptions commonly used and applied by people without training in scientific psychology (Lewis 1966, 1972, 1980; Armstrong 1970), or states ascribed to individuals by alleged analytical or conceptual truths about the mental (Shoemaker 1984). These functionalisms are irrelevant to this dissertation, which is only concerned with the sciences of minds and brains, not with folk theories or analytical truths about minds. Moreover, the issue of folk psychology and its relationship with scientific psychology is irrelevant to the question of the relationship between functionalism and computationalism. Hence, in this chapter we will only be concerned with functionalism with respect to states ascribed to individuals by scientific theories, not by folk theories or analytic truths.

Functionalism has also been developed as a doctrine about the *content* of mental states (e.g., Sellars 1954; Harman 1973, 1999; Block 1986). This chapter is not concerned with mental content, and whether some version of functionalism is sufficient to account for mental content is irrelevant to the relationship between functionalism and computationalism. So I will ignore the relationship between functionalism and content. Functionalism about mental content and its relation with computationalism are discussed in Chapter 9.

The doctrine of functionalism was proposed by Putnam as an alternative to other metaphysical theories of mind; his main targets were logical behaviorism and type-identity materialism.[4] Many also hold that functionalism is an anti-reductionist doctrine (e.g., Fodor 1975). While there is consensus that functionalism is incompatible with logical behaviorism, there has been a long and inconclusive controversy concerning the degree to which functionalism is compatible with elements of materialist and reductionist doctrines (e.g., Lewis 1969, Churchland 1982, Enç 1983, Kim 1989, 1992, Fodor 1997, Bickle 1998a, 1998b, Shagrir 1998, Sober 1999, Bechtel and Mundale 1999, Keeley 2000, Bechtel 2001). Like the issues of folk psychology and mental content, determining the exact relationships between functionalism, type-identity materialism, and reductionism is irrelevant to the question of the relationship between functionalism and computationalism. Therefore, these issues will be mostly ignored.

Finally, there is an extensive philosophical literature that attempts to refute functionalism *a priori*, usually based on intuitive judgments about imaginary scenarios (e.g., Block 1978, Searle 1980, Maudlin 1989). This literature usually focuses not on the general formulation of functionalism, but on its computational variety. Philosophers who reject computational functionalism judge some scenarios to be counterexamples to it (e.g., Searle 1980, 1991), whereas philosophers who accept computational functionalism judge the same scenarios not to be counterexamples to it (e.g., see the peer commentaries in Searle 1980, and Fodor 1991). Since philosophers with opposite theoretical commitments tend to judge

[4] The best-known logical behaviorists are Carnap (1932-3/1959) and Hempel (1935/1949). Similar to their position is the analytic behaviorism of Ryle (1949), Wittgenstein (1953), and Strawson (1959). For an overview of behaviorism, see O'Donohue and Kitchener 1999. The best-known type-identity materialists are Feigl (1958), Smart (1959), and Place (1959).

the same putative counterexamples in opposite ways, the discussion of these imaginary scenarios is inconclusive. This chapter contains no attempt to disprove computational functionalism *a priori*.[5]

8.2 Multiple Realizability and Computational Functionalism

In this section, I review the main lines of reasoning offered in the literature in favor of functionalism about the mind, pointing out that they don't support *computational* functionalism. That is, none of the reasons commonly given in support of functionalism are reasons to think that minds are computer programs. Therefore, even if functionalism is accepted, this is no reason to endorse computationalism. Arguments for functionalism come in the form of arguments from multiple realizability.[6]

Multiple realizability 1: Analogy with computers and their programs (Putnam 1960). The mind could be to the brain what a computer program is to a computer. The same computer program can be run on different machines made of physically different components, so states defined by programs are functional states not physical states. If minds are to brains what programs are to computers, then mental states are functional states not physical states. This argument derives functionalism from the premise that the mind is analogous to a computer program. Hence, on pain of circularity, it cannot be used to establish that minds are programs.[7]

---

[5] For a generally skeptical position on the use of imaginary scenarios in philosophy, see Piccinini forthcoming.

[6] Zangwill (1992) and Shagrir (1998) argue that there is no good argument for multiple realizability itself. Here, I assume multiple realizability and argue that it doesn't entail computational functionalism. Zangwill also argues for the stronger thesis that multiple realizability does not entail functionalism. For some multiple realizability arguments, this is correct. In fact, later we will see that multiple realizability was present in Putnam 1960 whereas functionalism was denied. But other multiple realizability arguments do entail functionalism. For example, the strong form of Multiple realizability 1 below is based on the thesis that the mind is a computer program. If the mind is a computer program, then the program itself is the functional organization of the mind in the sense required by functionalism.

[7] There are two variants of this argument from the analogy with computer programs, one stronger than the other. The weaker version assumes that minds are analogous to programs in being multiply realizable, but falls short of assuming that minds *are* programs. This version can be found in Putnam 1960, p. 149 (cited below), and does not entail that there are functional state types that correspond to mental state types. This is why Putnam 1960 could both argue for multiple realizability on the ground of his analogy between minds and programs and deny the thesis of functionalism. The stronger version of this argument goes all the way to assume that minds are programs for brains, and therefore that brains are stored-program computers. This version can be found, for example, in Putnam 1964 (as a conditional statement) and Fodor 1968a. Fodor 1975 runs a similar argument:

Multiple realizability 2: Analogy between human minds and the minds of animals, Martians, robots, and angels (Putnam 1967a, 1967b). An animal, a Martian, a robot, or an angel can be said to be in the same mental state as humans are; for example, in pain. Yet, their brains might be in a different neural state than human brains, or their brains might be made of different materials than human brains, or they might have no brains at all. Therefore, mental state types are not identical to brain states types. Sound as this argument may be, it does not entail the computational thesis that minds are programs any more than it entails that animal, Martian, robot, or angel minds are programs. It does not even entail that types of mental states are functional types (in the sense of functionalism described here).[8]

Multiple realizability 3: Analogy with artifacts (Fodor 1965, 1968a, 1968b). Minds could be like clocks or engines. Clocks and engines can be made of physically different components, so being a clock or an engine is a functional not a physical property, at least in the sense that the identity conditions of clocks and engines depend on what they can be used for (Fodor 1968b, p. 116). If minds are like clocks and engines, then the same is true of minds. However, even if minds are like clocks and engines, this argument does not entail that minds are programs any more than it entails that clocks or engines are programs.

Multiple realizability 4: Analogy with biological traits (Fodor 1968a; Block and Fodor 1972, p. 238). Minds could be like wings or feet. Insects, birds, and bats have very different sorts of wings and feet, yet they are all wings and feet. So, being a wing or a foot is a functional not a physical property. If minds are like wings or feet, then the same is true of minds. However, even if minds are like wings or

---

> [I]t is entirely possible that the nervous system of higher organisms characteristically achieves a given psychological end by a wide variety of neurological means. It is also possible that given neurological structures subserve many different psychological functions at different times, depending upon the character of the activities in which the organism is engaged. [Fn: This would be the case if higher organisms really are interestingly analogous to general purpose computers. Such machines exhibit no detailed structure-to-function correspondence over time; rather, the function subserved by a given structure may change from instant to instant depending upon the character of the program and of the computation being performed.] (Fodor 1975, p. 17).

[8] The case of the robot deserves a comment. One might argue that, in the case of the robot, we are inclined to think that its behavior is driven by a program. Hence, at least the robot's mind is a program. This argument begs the question of computational functionalism. The question is whether minds are programs, not whether the behavior of robots is driven by programs (which, incidentally, it need not be, cf. Brooks 1997). If minds are something other than programs, then any mind of a robot will also be something other than a program.

feet, this argument does not entail that minds are programs any more than it entails that wings or feet are programs.

Multiple realizability 5: Analogy with "high level" physical properties (Lycan 1981).[9] Having a mind could be like being a metal or a vortex. Different chemical substances are metals, and vortices can originate within different physical media. So, being a metal or a vortex is a functional not a (micro-)physical property. If having a mind is like being a metal or a vortex, then the same is true of minds. However, even if having a mind is like being a metal or a vortex, this argument does not entail that minds are programs any more than it entails that metals or vortices are programs.

The different multiple realizability arguments listed do not appeal to the same notion of function (when they do appeal to a notion of function). Argument 1 appeals to the functional properties of a particular type of mechanism, the stored-program computer. Argument 2 appeals to our dispositions to ascribe mental qualities. Arguments 3 and 4 appeal to something like the notion of proper function (discussed below). Argument 5 appeals to "high level" physical properties. The existence of so many multiple realizability arguments, which use so many different notions of function, calls for some clarification. In the next section, I offer a clarified formulation of computational functionalism, which is compatible with the idea that mental states are multiply realizable.

8.3 Multiple Realizability, Functional Analysis, and Program Execution

Computational functionalism is based on three main ideas: multiple realizability, functional analysis, and explanation by program execution.

> *Multiple realizability* about a state or system X under description D1 is the thesis that (i) there is another description D2 of X, (ii) both D1 and D2 have explanatory value within some science of

[9] Lycan doesn't run a multiple realizability argument but he seems to think that being a metal is in some sense a functional property (Lycan 1981, p. 33).

X, and (iii) X under description D2 is a realization of X under description D1, i.e. D1 also applies to classes of systems that D2 does not apply to. For short, I'll say that D2 realizes D1.

For instance, the same object may be described as a piece of metal (D1) or as a piece of iron (D2), and both descriptions can be used in chemical explanations. D1 ("metal") also applies to pieces of aluminum, lead, gold, etc., which are classes of objects that D1 ("iron") does not apply to. So D2 realizes D1, and D1 is multiply realizable.

A *Functional analysis* of a system X is a description of X in terms of spatiotemporal components of X and their proper functions.

The notion of proper function is normative: *x* has proper function *pf* if and only if *x* must perform *pf*—it is right for *x* to do *pf*, wrong to do otherwise. When something fails to perform its proper function, we say that it malfunctions or is defective. I will not attempt to analyze proper functions any further.[10] Paradigmatic examples of things with proper functions are biological organs and artifacts. Paradigmatic examples of things without proper functions are clouds.[11]

Functionalism about a system, as I'll use the term, says that the system can be given a functional analysis, and the functional states of the system are identified by their role within the functional analysis of the system. This version of functionalism is different from the classical one. According to classical functionalism, a system is identified by its inputs, outputs, and functional states, and a functional state is identified by its causal relations with inputs, outputs, and other functional states. According to the current version of functionalism, a system is identified by its component parts, their mutual connections, and their proper functions, and the functional states of the system are identified by their contribution in fulfilling

[10] The term "proper function" is due to Millikan, who emphasizes the normative character of function ascription (1984, 1993). I do not want to be committed to Millikan's account of function ascription. For an alternative account of proper functions, see Cummins 1975, 1983, Bechtel and Richardson 1993, Schlosser 1998, Craver 2001, Mahner and Bunge 2001.

[11] I took these examples from Millikan 1993, chapters 1 and 2.

the system's proper function(s). When a functional analysis of a system is available, states of the system are not only identified by their causal relations to other states, inputs, and outputs, but also by the component to which they belong and the proper function performed by that component when it is in that state. For example, ordinary TM states would be identified not only as causing and being caused by certain inputs, outputs, and other states, but also as being states *of* the tape or *of* the active device, which are the components of the TM.

This non-classical formulation will be motivated by the discussion of the functionalist literature in the rest of this chapter. We'll see that early functionalists formulated functionalism in the classical way, but they gave examples of functional systems that implicitly appealed to functional analysis in order to identify the system, its components, and its states. So their discussion of functionalism was ambiguous between the classical and the present versions. To resolve this ambiguity, a version of functionalism based on the notion of proper function—similar in this respect to the one adopted here—was proposed by Lycan (1981, 1987) and Sober (1984). A full justification for identifying computing mechanisms (along with other mechanisms) by functional analysis will be given in Chapter 10.

> An *explanation by program execution* of a capacity C possessed by a system X is an explanation that postulates a set of instructions I and says that X possesses C because X executes I (in the sense in which computers execute instructions).

The sense in which computers execute instructions will be briefly discussed below, and it will be made more explicit in Chapter 10. The importance of the notion of explanation by program execution for the understanding of computational functionalism will become clearer in section 8.4, when I will investigate the history of computational functionalism, and where I'll argue that some of the above notions have been inappropriately conflated in the literature. For now, the intuitive notion of program execution will suffice for our purposes.

These three ideas are related as follows.

8.3.1 Multiple Realizability and Functional Analysis

Multiple realizability about X says that there are two descriptions D1 and D2 of X, one of which implements the other. This is compatible with either D1, or D2, or both, or neither being functional analyses. For example, multiple realizability about the mind-brain under its psychological and neurological descriptions says that psychological descriptions are implemented by neurological descriptions; however, *both* psychological theories and neurological theories are usually couched in a functional language that attributes functions to components of the system. For instance, a psychological theory of memory may attribute the function of storing memories to a Short Term Memory component, whereas a neurological theory of memory may attribute the function of storing memories to the strength of synaptic connections within certain neurons. On the other hand, multiple realizability can apply to a system under the descriptions *vortex* and *whirlwind*, neither of which is a functional analysis (because neither is a partition of a system into components and an ascription of proper functions to the components). In mathematical physics it is common to encounter classes of equations that describe different classes of systems; for example there is a class of equations that applies both to spin glasses and to what have come to be called Hopfield neural networks (Hopfield 1982). In this case, the differential equations are hardly functional analyses, and although spin glasses are not described by functional analysis, Hopfield networks can be.

8.3.2 Multiple Realizability and Explanation by Program Execution

Some systems execute instructions (e.g., stored-program computers) while others don't (e.g., planetary systems). Whether multiple realizability applies to either of these classes of systems does not depend on whether they execute instructions but on whether they possess two different descriptions and whether one of their descriptions realizes the other. In the examples given in the previous section, perhaps possessing memory involves executing instructions, but being a vortex hardly does. Nonetheless, multiple realizability may apply to both.

An explanation by program execution postulates a set of instructions to explain a system's capacity. In a loose sense, the set of instructions may be considered as a description of the system's capacity, to which the doctrine of multiple realizability applies. In fact, many physically different computers can execute the same set of instructions. So, one may theorize about what instructions are executed by what system and formulate theories that postulate certain sets of instructions to explain certain capacities. In such a case, given the explanation of the system's capacities by program execution, multiple realizability about that system says that there is another description of the system that realizes the program. The delicate question—for the purpose of understanding explanation by program execution—is what that description describes. When the behavior of ordinary stored-program digital computers is explained by appeal to program execution, the program is not just an abstract description of a capacity. The program is also a *physical component* of the system (or a stable state of a component), whose proper function is to generate the relevant capacity of the computer. Programs are physically present within computers, where they have a proper function to perform. Somehow, this simple and straightforward point seems to have been almost entirely missed in the philosophical literature.[12]

8.3.3 Functional Analysis and Program Execution

A system that is subject to functional analysis may or may not execute programs. For example, digital computers are analyzed functionally in terms of various kinds of memories, processors, input devices, and output devices, and their capacities are explained by appeal to the execution of programs. However,

[12] Robert Cummins, one of the few people to discuss this issue explicitly, has even argued against it (1977, 1983). He says that programs are not causes of computations, even in stored-programs computers. The main reason for Cummins's odd doctrine seems to be his conflation of functional analysis and description by a program. Briefly, Cummins thinks that explaining a capacity by program execution is the same as giving a functional analysis of it, and therefore the program is primarily not a part of the computer but a description of it. In later sections, I'll discuss this conflation in more detail and argue that Cummins inherited it from the literature on functionalism, where it originated with Putnam and Fodor's views that functional analyses are given by TM tables, and that the relationship between a computer program and the machine on which it runs is the same as the relationship between a TM table and its hardware implementation. In Chapter 10, I'll criticize Putnam, Fodor, and Cummins's accounts of computing mechanisms in detail, and vindicate the statement made here, that programs are physical components of (some) computing mechanisms.

many biological systems are analyzed functionally (Bechtel and Richardson 1993, Craver and Darden 2001), but their capacities are rarely explained by program execution.[13]

So, explaining a capacity by program execution is not the same as providing a functional analysis of a system. Rather, it is to provide part of a very *peculiar kind* of functional analysis. Computers are subject to explanation by program execution because they are very peculiar kinds of functional systems, and their peculiarity is described by peculiar functional analyses. That is, computers have processors that are capable of a finite number of primitive operations, and memories that can contain instructions. Moreover, computers' processors have the capacity to copy the instructions contained in the memories of the computer and perform the corresponding sequences of operations, sending the results to the output devices. This is a brief functional analysis of computers, which appeals to kinds of components and their proper functions.[14] This particular functional analysis (or better, a detailed and complete version of it) *explains* how computers have the peculiar capacity to execute instructions, that is, it explains by functional analysis why certain capacities of computers can be, in turn, explained by program execution.

Explanation by program execution says that there is a stable state of a part of a system that, properly interpreted, specifies what the system is going to do to its input; in other words, explanation by program execution presupposes that a state of part of the system *functions* as a program; in descriptions of stored-program computers, "program" is used as a function term. The way a program determines what the system is going to do is cashed out in terms of the functional analysis of the rest of the whole system as a program-executing system. So, explanation by program execution presupposes that the system executing the program is a very particular kind of system such that it executes programs in a way that corresponds to what the program says. In the case of ordinary computers, this property of the computer depends on the structure of the circuitry. This is why the appeal to program execution is explanatory for those systems; because it postulates programs executed by a particular kind of mechanism: a program-executing mechanism. Stored-program computers are program-executing mechanisms.

[13] A possible exception is the notion of developmental "program," which would require a separate analysis.
[14] Cf. any standard textbook on computer organization and design, e.g. Patterson and Hennessy 1999. This account of computing mechanisms will be expanded in Chapter 10.

8.3.4 Computational Functionalism Revisited

Now we are ready to formulate computational functionalism about the mind, i.e. a thesis about minds motivated by an appeal to program execution as an explanation for mental capacities:

> *Computational functionalism*: minds are programs running on brains.

Computational functionalism has several consequences that are relevant to the study of minds and brains.

Computational functionalism allows for the use of computation by program execution as an *explanation* for psychological capacities. This is a kind of mechanical explanation, which postulates a mechanism—the stored-program computer—and an initial condition of (part of) the mechanism—the program—which together determine the behavior to be explained. It may surprise some readers that given this version of computational functionalism, there is a sense in which the mind is a physical component of the brain, the same sense in which computer programs are physical components of computers.

Computational functionalism offers a particular functional analysis of the mind. That is, the mind is described as a program, which means that the function of the mind is to specify what sequences of instructions the brain has to execute. This presupposes a particular functional analysis of the brain as a program-executing mechanism, i.e. a mechanism that can store programs and execute them. Whether a functional analysis applies to a system is an empirical question. So whether the brain has the particular functional analysis of a program-executing mechanism is an empirical question, and if the brain were not functionally organized in this way, computational functionalism about the mind would turn out to be false. This shows computational functionalism to be an empirical hypothesis. So, computational functionalism postulates a level of computational theory (by functional analysis) of the brain, which is supposed to explain how brains can execute instructions. This theory can be tested by observing the brain

to determine whether it is organized as a stored-program computer. There remains the task of explicating in more detail what it means to execute a program. This is accomplished in Chapter 10.

Computational functionalism entails that minds are multiply realizable, because the same program can run on physically different pieces of hardware. Because of this, the mental program can also be characterized and studied abstractly, independently of how it is implemented in the brain, in the same way that one can investigate what programs are (or should be) run by digital computers without worrying about how they are physically implemented. Under the computational functionalist hypothesis, this would be the task of psychological theorizing. So, psychological theories would provide abstract descriptions in the form of programs, which would be expected to be implemented by the brain under its description as a stored-program computer. The programs postulated by psychological theories can be seen as offering task analyses (which are a kind of functional analyses) of mental capacities (more on this below).

Following the mainstream literature (to be reviewed in section 8.4 below), I formulated computational functionalism using the notion of explanation by program execution. Other notions of computational explanation are available, which appeal to computing mechanisms that do not execute instructions. The most popular of such computing mechanisms are connectionist networks. To cover connectionist computationalist theories of mind, the present formulation of computational functionalism can be generalized by replacing program execution with other computational processes, such as connectionist computation. According to this generalized computational functionalism, the mind is the computational organization of a computing machine, whether that machine is a stored-program computer, a connectionist computing mechanism, or any other kind of computing machine (e.g., a finite state automaton). Given the generalized formulation, psychological explanations need not invoke the execution of programs—they can invoke either program execution or any kind of computation (connectionist or otherwise) presumed to generate the behavior to be explained. Given this generalization, computational functionalism is fully compatible with connectionist computationalism.

Computational functionalism entails functionalism, namely the view that the mind is subject to functional analysis. However, functionalism is a more general doctrine than computational functionalism. There are many functional analyses that do not involve program execution or any other computational process, and simply stating that the mind is subject to functional analysis is consistent with any non-computational functional analysis applying to the mind. In later sections, we will see that this point is not a consequence of my reformulation of functionalism and computational functionalism. It applies also to the original versions of them formulated by Putnam and Fodor in the 1960s.

Computational functionalism about the mind is relevant to this dissertation because it entails computationalism about the brain, and computationalism is the subject of this dissertation. There is no entailment in the other direction. Computationalism about the brain is compatible with the mind being the computational organization of the brain, but also some non-computational, higher-order functional property of the brain, or even some non-functional property of the brain, such as its physical composition, the speed of its action, its color, or more plausibly, the intentional content or the phenomenal qualities of its states. That is to say, one can be a computationalist about the brain while opposing or being neutral about computational functionalism about the mind.

My formulation of computational functionalism as an empirical hypothesis about the type of functional analysis that applies to the mind-brain may be surprising to those who are used to its traditional formulation as a metaphysical thesis. They may resist it on the grounds that functional analysis is really a form of computational description, so that computational functionalism is a consequence of functionalism after all. The best way to respond to this objection may be to take a close look at the origin of computational functionalism. We shall see how the doctrine that functional analysis is a form of computational description got inserted into and mixed up with computational functionalism: not by argument of careful conceptual analysis, but rather by a series of historical accretions. While responding to this objection, I will offer the first careful historical reconstruction of the origin of computational functionalism, which also shows how the transition from functionalism *tout court* to computational functionalism was not motivated by argument but by the search for a more explicit formulation of

functionalism, in an intellectual environment strongly influenced by computational approaches to the science of mind and brain.

8.4 Origin of Computational Functionalism

In the 1960s, Putnam proposed both functionalism and computational functionalism. In formulating computational functionalism, he was influenced by Fodor's ideas about psychological theories, which in turn were influenced by Putnam's functionalism. Fodor also went on to popularize computational functionalism in the form of a computational theory of mind that explained psychological capacities. Accordingly, a good way to introduce and unpack these doctrines is to look at their origin in the work of Putnam and Fodor.

8.4.1 The Brain as a Turing Machine

Putnam believed that the human *brain* is a probabilistic TM, which he sometimes called probabilistic automaton.[15] He commented on computationalism (about the brain) as early as 1958, in a long paper on reductionism, which he co-authored with Paul Oppenheim before any of his papers on minds and machines. In that paper, Oppenheim and Putnam address the reduction of mental phenomena, such as "learning, intelligence, and perception," to cellular activity, and specifically to the activity of networks of neurons (Oppenheim and Putnam 1958, pp. 18-19).

Oppenheim and Putnam argue that Turing's analysis of computation "naturally" leads to hypothesize that the brain is a TM:

> The logician Alan Turing proposed (and solved) the problem of giving a characterization of *computing machines* in the widest sense—mechanisms for solving problems by effective series of logical operations. This naturally suggests the idea of seeing whether a "Turing machine" could consist of the elements used in neurological theories of the brain; that is, whether it could consist of a network of neurons. Such a nerve network could then serve as a hypothetical model for the brain (Oppenheim and Putnam 1958, p. 19; their italics).

[15] A probabilistic TM is one some of whose instructions share the first two symbols.

Then, Oppenheim and Putnam point to McCulloch and Pitt's theory (McCulloch and Pitts 1943), von Neumann's model of reliable computation from unreliable components (von Neumann 1956), and related work (e.g., Shannon and McCarthy 1956) as theories of the brain that are capable of reducing mental phenomena to neural activity.[16]

In 1961, Putnam writes as follows: "I would suggest that there are many considerations which point to the idea that a Turing machine plus random elements is a reasonable model for the human brain" (Putnam 1961). In that paper, Putnam does not say what those considerations are. But in a later paper, he mentions McCulloch and Pitts' theory and the mechanical mice created by Shannon (1952) in a similar context. On this occasion, Putnam does not explicitly endorse the view that the brain is a probabilistic TM, but he points out that, *if* the human brain is such a device, then any physical realization of the same TM program has the same *psychology* that humans have:

> [I]f the human brain is a 'probabilistic automaton,' then any robot with the same 'machine table' [i.e., TM program] will be psychologically isomorphic to a human being. If the human brain is simply a neural net with a certain program, as in the theory of Pitts and McCulloch, then a robot whose 'brain' was a similar net, only constructed of flip-flops rather than neurons, would have exactly the same psychology as a human (Putnam 1964, pp. 394-395).

This shows that in the early 1960s, Putnam was taking seriously McCulloch and Pitts's theory that the brain is a computing mechanism, and concluded from it that a robot realizing the appropriate computational description would have the same psychology as humans do.[17] The step from that to the conclusion that the mind is the abstract computational organization of the mechanism (as described by its TM program) is relatively short. This suggests that Putnam's computationalism about the mind derived at least in part from the belief that the human brain is a computing mechanism, rather than *vice versa*. As a

[16] They add:

> In terms of such nerve nets it is possible to give hypothetical micro-reductions for *memory*, *exact thinking*, *distinguishing similarity or dissimilarity in stimulus pattern*, *abstracting* of "essential" components of a simulus pattern, recognition of shape regardless of form and of chord regardless of pitch …, *purposeful behavior* as controlled by negative feedback, *adaptive behavior*, and *mental disorders*" (Oppenheim and Putnam 1958, p. 20; their italics).

[17] Curiously, neither Putnam nor subsequent philosophers of mind seem to have commented on McCulloch and Pitts's claim that their theory solved the mind-body problem. Putnam's computational functionalism can be seen as ways of making McCulloch and Pitts's claim explicit.

matter of fact, at this point in time Putnam had not yet committed in print to the thesis that the mind is a TM program, nor had he offered any reason to believe that it is.

8.4.2 The Analogy between Minds and Turing Machines

Much of the initial motivation for Putnam's formulation of functionalism about minds lies in what he calls an "analogy between man and Turing machine" (Putnam 1960, p. 159). According to Putnam, mental states are analogous to the states defined by a TM program, whereas brain states are analogous to the physical states of a hardware realization of a TM program. In Putnam's 1960 paper, the analogy is not yet aimed at proposing a functionalist metaphysics of minds, but at constructing an analogue of the mind-body problem for TMs. The existence of the TM analogue problem, in turn, is meant to show that the mind-body problem is a purely "verbal" or "linguistic" problem, which does not demand a solution any more than its TM analogue demands a solution. Here we won't dwell on Putnam's critique of the mind-body problem, but only on the relationship between Putnam's analogy and the doctrine of functionalism. Throughout his paper, Putnam stresses that the analogy between minds and TMs cannot be stretched too far. He explicitly rejects the following claims: that the mind-body problem literally arises for TMs (ib., p. 138), that machines think or that humans are machines (ib., p. 140), and that machines can be properly said to employ a language (ib., p. 159).

Putnam applies his analogy between men and TMs to psychological theories:

> It is interesting to note that just as there are two possible descriptions of the behavior of a Turing machine—the engineer's structural blueprint and the logician's "machine table" [i.e., TM program]—so there are two possible descriptions of human psychology. The "behavioristic" approach … aims at eventually providing a complete physicalistic [fn: In the sense of Oppenheim and Putnam 1958 …] description of human behavior. This corresponds to the engineer's or physicist's description of a physically realized Turing machine. But it would also be possible to seek a more abstract description of human mental processes, in terms of "mental states" (physical realization, if any, unspecified) and "impressions" (these play the role of symbols on the machine's tapes)—a description which would specify the laws controlling the order in which the states succeed one another, and the relation to verbalization… This description, which would be the analogue of a "machine table," it was in fact the program of classical psychology to provide! (ib., pp. 148-149.)

In this passage, Putnam points out that psychological theories can be formulated in two ways: one describes behavioral dispositions and physiological mechanisms, the other describes "mental states" and "impressions." Then Putnam suggests that *if* it were possible to formulate an "abstract" psychological theory in terms of mental states, then that theory would stand to a psychological theory describing physiological mechanisms in the same relation that TM programs stand to descriptions of their physical realizations. Putnam's suggestion is offered without argument, as an analogy with descriptions of TMs.

After drawing the analogy between psychological theories and TMs, Putnam explicitly *denies* that "abstract" psychological theories can be formulated, because functionalism about minds does not hold:

> Classical psychology is often thought to have failed for methodological reasons; I would suggest, in the light of this analogy, that it failed rather for empirical reasons—the mental states and "impressions" of human beings do not form a causally closed system to the extent to which the "configurations" of a Turing machine do (ib., p. 149).

By hypothesizing that human minds "do not form a causally closed system," Putnam suggests that there can be no theory of minds in terms of "mental states," and *a fortiori* no complete functional description of minds. This view is the denial of functionalism, and it shows that in his first paper on minds and TMs, Putnam is not yet a functionalist. Thus, he rejects the doctrine of functionalism about minds. In that paper, his only goal is to convince the reader that there is enough positive analogy between humans and TMs to show that the mind-body problem is a pseudo-problem.

According to Putnam, the positive analogy between minds and TMs includes an important element, which later became part of the functionalist doctrine. That element is multiple realizability, namely the view that the same mental states can be physically realized in different ways. Putnam says:

> The functional organization (problem solving, thinking) of the human being or machine can be described in terms of the sequences of mental or logical states respectively (and the accompanying verbalizations), without reference to the nature of the "physical realization" of these states (ib., p. 149).

Besides multiple realizability, another element of Putnam's positive analogy was going to play an important role in computationalism, namely the view that just like human thought processes, the computations generated by TM programs are *rational* processes: "In the case of rational thought (or

computing), the 'program' which determines which states follow which, etc., is open to rational criticism" (ib., p. 149). These elements would become integral parts of computational functionalism. We'll come back to them later. For the moment, we focus on the theme of the analysis of psychological theories.

8.4.3 Psychological Theories, Functional Analysis, and Programs

Putnam's theme of psychological theories was picked up by Fodor, and through Fodor's work it became an important motivation for computational functionalism. Fodor offered his analysis of psychological theories in an article that has "a particular indebtedness" to a book by J. Deutsch (1960) and to Putnam's 1960 article (Fodor 1965, p. 161).

Fodor's account of psychological theories, which is largely borrowed from Deutsch (1960, esp. pp. 10-15), goes as follows. Psychological theories are developed in two logically distinct phases. Phase one theories are *functional analyses*, whereas phase two theories are *mechanical analyses* (Fodor's terms). The distinction between functional and mechanical analysis is explicated by the example of internal combustion engines. Functional analysis identifies the functions of engine parts, namely their contribution to the activities of the whole engine. For an internal combustion engine to generate motive power, fuel must enter the cylinders, where it detonates and drives the cylinders. In order to regulate the flux of fuel into the cylinders, functional analysis says there must be valves that are opened by valve lifters. Valve lifters contribute to the activities of the engine by lifting valves that let fuel into the cylinders. Given a functional analysis of an engine, mechanical analysis identifies physical structures that correspond to the functional analysis of the engine. In certain engines, camshafts are identified as physical structures that function as valve lifters, although in other engines the same function may be performed by other physical structures. By the same token, according to Fodor, phase one psychological theories identify psychological functions (functional analysis), whereas phase two psychological theories identify physiological structures that perform those functions (mechanical analysis). From his notion of functional analysis as explicated by his example of engines, Fodor infers that psychological theories have indefinitely many realizations or "models"; different mechanisms can realize a given functional analysis

(ib., p. 174-175). Fodor also argues that the relationship between phase one (functional analysis) and phase two (mechanical analysis) psychological theories is not a relation of reductive "microanalysis" in the sense of Oppenheim and Putnam 1958, in which there is a type-type correspondence between the predicates of the two descriptions (ib., p. 177). In all of this, Fodor is following quite closely the account of psychological theories proposed by Deutsch, who also inferred that there is a "theoretically infinite variety of counterparts" of any type of mechanism postulated by phase one psychological theories (Deutsch 1960, p. 13).

Prima facie, Fodor's analysis of psychological theories is unrelated to the analogy between TMs and psychological descriptions drawn by Putnam in his 1960 article. On the one hand, TM programs are finite sets of quadruples specifying a finite number of *state* types and what state transitions must occur under what conditions, and they do not specify either what component types must make up the system that realizes the TM program or what their proper functions must be. On the other hand, functional analyses—based on Fodor's examples and Deutsch's formulation—are specifications of mechanism types, consisting of different *component* types and their assigned proper functions without specifying the precise state types and state transitions that must occur within the analyzed system. A functional analysis of an engine does not come in the form of a TM program, nor is it obvious how it could be turned into a TM program or whether turning it into a TM program would have any value for explaining the functioning of the engine. Programs can be analyzed into subroutines, and subroutines can be analyzed into sequences of elementary operations, but this is not a functional analysis in Deutsch's sense. Even the fact that both TM programs and functional analyses can be multiply realized seems to originate from different reasons. A functional analysis can be multiply realized because systems with different physical properties can perform the same concrete proper function (e.g., generate motive power), whereas a TM program can be multiply realized because systems with different physical properties, no matter what proper functions they perform, can realize the same abstractly specified state transitions. At the very least, the thesis that the two are related requires analysis and argument. So, *prima facie* there is no reason to think that giving a functional analysis of a system is equivalent to describing that system by using a

TM program. In fact, neither Fodor nor his predecessor Deutsch mention computers or TMs, nor do they infer, from the fact that psychological descriptions are functional analyses, that the mind is a TM program or that the brain is a computer.

However, when Fodor describes phase one psychological theories in general, he departs from his example of the engine and from Deutsch's view. Fodor describes psychological functional analyses as postulations of internal *states*, i.e. as descriptions that closely resemble TM programs: "Phase one explanations purport to account for behavior in terms of internal states" (Fodor 1965, p. 173). This way of describing functional analyses is clearly under the influence of Putnam's 1960 analogy between minds and TMs. In a later work, where Fodor repeats his two-phase analysis of psychological theories, he explicitly attributes his formulation of phase one psychological theories to Putnam's 1960 analogy (Fodor 1968b, p. 108).[18] However, in his 1965 article Fodor does not discuss TM programs explicitly, nor does he explain how his formulation of phase one psychological theories in terms of states squares with his example of the engine. He also does not explain why he adopted Deutsch's analysis of psychological theories but rejected the notion that phase one psychological theories hypothesize the existence of mechanism types and analyze them by postulating component types and their proper functions.

The first to accept Fodor's analysis of psychological theories was probably Putnam himself, in a paper that appeared in print before Fodor's:

> Psychological theories say that an organism has certain states which are not specified in 'physical' terms, but which are taken as primitive. Relations are specified between these states, and between the totality of the states and sensory inputs ('stimuli') and behavior ('responses'). Thus, as Jerry Fodor has remarked (Fodor, 1965), it is part of the 'logic' of psychological theories that (physically) *different* structures may obey (or be 'models' of) the *same* psychological theory (Putnam 1964).

Here, Putnam adopts Fodor's formulation of psychological theories in terms of state types and their relations to inputs and outputs.

Thus, Fodor 1965 introduces in the philosophical literature both the notion that psychological theories are functional analyses and the notion that psychological theories are like TM programs in that

[18] Fodor doesn't mention that Putnam 1960 claimed that there can't be a psychological theory of this type because minds are not causally closed systems; Fodor implicitly disagrees with Putnam on this point.

they are descriptions of transitions between state types. Both themes would be very successful in philosophy of psychology. This happened before the publication by Putnam of his doctrine of functionalism about the mind.

8.4.4 Functionalism

According to Putnam (1963, p.35, fn. 2), he first formulated the doctrine of functionalism about minds in his paper "The Mental Life of Some Machines," delivered to the Wayne State University Symposium in the Philosophy of Mind in 1962, and published as Putnam 1967a. In that paper, Putnam again introduces his analogy between minds and TMs, this time with the purpose of analyzing mentalistic notions like "preferring," "believing," and "feeling." He points out that in many ways the sense in which TMs are said to have beliefs, preferences, or feelings, is different from the sense in which human beings are said to have such states; but he submits that these differences are irrelevant to his argument (Putnam 1967a, p. 177-178). His strategy is to apply mentalistic descriptions to appropriate TMs and argue that for these machines, all traditional metaphysical doctrines about minds (materialism, dualism, and logical behaviorism) are incorrect (ib.). This constitutes a shift from the 1960 article, where the same analogy between minds and TMs was used to argue that the mind-body problem is a purely verbal problem. Here, instead of the traditional metaphysical mind-body doctrines, Putnam offers functionalism about minds:

> It seems that to know for certain that a human being has a particular belief, or preference, or whatever, involves knowing something about the functional organization of the human being. As applied to Turing Machines, the functional organization is given by the machine table [i.e., TM program]. A description of the functional organization of a human being might well be something quite different and more complicated. But the important thing is that descriptions of the functional organization of a system are logically different in kind either from descriptions of its physical-chemical composition or from descriptions of its actual and potential behavior (ib., p. 200).

In this passage, Putnam explicitly proposes that knowing minds is the same as knowing the functional organization of human beings, in analogy with the functional organization of TMs that is given by a TM program. But he immediately adds that the description of the functional organization of a human being might be "something quite different and more complicated" than a TM program. This shows that, even in

Putnam's original formulation, functionalism is logically weaker than computational functionalism. That is, the doctrine that mental states are functional states was initially formulated without assuming that the functional description of minds can or must be given by a TM program.

8.4.5 Computational Functionalism

Putnam takes the step from functionalism to *computational* functionalism in his best-known paper on the subject, published in 1967 (Putnam 1967b). In that paper, he states his functionalist doctrine directly in terms of TM programs:

> A Description of S where S is a system, is any true statement to the effect that S possesses distinct states S_1, S_2 …, S_n which are related to one another and to the motor outputs and sensory inputs by the transition probabilities given in such-and-such a Machine Table [i.e., TM program]. The Machine Table mentioned in the Description will then be called the Functional Organization of S relative to that Description, and the Si such that S is in state Si at a given time will be called the Total State of a system relative to that Description. It should be noted that knowing the Total State of a system [is] relative to a Description. Description involves knowing a good deal about how the system is likely to "behave," given various combinations of sensory inputs, but does not involve knowing the physical realization of the S_i as, e.g., physical-chemical states of the brain. The S_i, to repeat, are specified only implicitly by the Description—i.e., specified only by the set of transition probabilities given in the Machine Table.
>
> The hypothesis that "being in pain is a functional state of the organism" may now be spelled out more exactly as follows:
>
> 1 All organisms capable of feeling pain are Probabilistic Automata.
>
> 2 Every organism capable of feeling pain possesses at least one Description of a certain kind (i.e., being capable of feeling pain is possessing an appropriate kind of Functional Organization).
>
> 3 No organism capable of feeling pain possesses a decomposition into parts which separately possess Descriptions of the kind referred to in (2).
>
> 4 For every Description of the kind referred to in (2), there exists a subset of the sensory inputs such that an organism with that Description is in pain when and only when some of its sensory inputs are in that subset (Putnam 1967b, 30).

Clause 2 says that there is a particular TM program that describes creatures with mental states. Clause 4 says that a creature is in pain if and only if that particular TM program, when applied to the creature, indicates that the creature is in the type of state that corresponds to pain. Since this is supposed to hold for every mental state, mental states can be defined by a TM program. (Clause 3 is an *ad hoc* addition to avoid attributing mental states to collective individuals, e.g. bee swarms (ib., p. 31).)

In the same paper, Putnam argues that his proposal is a more plausible "empirical hypothesis" about mental states than either type-identity materialism or logical behaviorism. Putnam's arguments about both doctrines are as follows.

First, type-identity materialism. For present purposes, type-identity materialism is the thesis that any type of mental state is identical to some type of brain state. Putnam argues that type-identity materialism is committed to finding the same types of brain states in all organisms that realize the same types of mental states, whether they are mammals, reptiles, mollusks, or Martians. He also argues that this is implausible, because these organisms have widely different brains, or perhaps no brains at all. These points are similar to those he made in previous papers (Putnam 1960, 1967a). Unlike type-identity materialism, Putnam's functionalist doctrine is only committed to those organisms having the same functional organization, not to their having the same types of brain states. Hence, Putnam concludes that functionalism is more plausible than type-identity materialism (ib., pp. 31-32). The assumption that the functional organization of minds is given in terms of TM programs plays no role in Putnam's comparison of the plausibility of functionalism and type-identity materialism, and is not even mentioned by Putnam while making that comparison.

Second, logical behaviorism. For present purposes, logical behaviorism is the thesis that mental states are sets of behavioral dispositions. About logical behaviorism, Putnam offers a number of considerations similar to those he offered in previous papers (Putnam 1963, 1967a). The main consideration hinges on the premise that, contrary to logical behaviorism, the same set of behavioral dispositions may correspond to different mental conditions. One of Putnam's examples involves two animals whose motor nerves are cut. These animals have the same set of behavioral dispositions, namely, they are paralyzed. But intuitively, if the first animal has intact sensory nerves whereas the second animal has cut sensory nerves, under appropriate stimulation the first animal will feel pain whereas the second animal won't. So, contrary to logical behaviorism, being in pain is not a behavioral disposition. Putnam's functionalist doctrine escapes this objection because it identifies the different states of the two animals in terms of the effects of sensory inputs (or lack thereof) on their internal states. Hence, Putnam

concludes that functionalism is more plausible than logical behaviorism (ib., pp. 32-33). Again, the assumption that the functional organization of minds is given by TM programs plays no role in Putnam's comparison of functionalism and logical behaviorism, and is not mentioned by Putnam during his discussion.

Let us recapitulate Putnam's path to computational functionalism. In 1960, when Putnam first formulated the analogy between minds and TMs, he denied that the mind was a closed causal system (Putnam 1960). Later, he formulated the doctrine of functionalism, namely, the doctrine that mental states *can* be identified by a functional description (Putnam 1967a), and he adopted Fodor's 1965 view that psychological theories are descriptions of transitions between state types (Putnam 1964). Finally, he added the further *functional-computational* thesis that functional descriptions of minds are TM programs (Putnam 1967b). The transition between functionalism and computational functionalism was made without argument. In arguing in favor of computational functionalism, and against both type-identity materialism and logical behaviorism, Putnam used arguments that make no appeal, either as a premise or as a conclusion, to the functional-computational thesis. This is unsurprising, because those arguments were already formulated in Putnam's previous papers, where the computational thesis was not stated (Putnam 1960, 1963, 1967a).

8.4.6 Functional Analysis and Explanation by Program Execution

Fodor's 1965 description of phase one psychological theories as having the same form as TM programs paved the way for the later *identification* of psychological functional analyses and TM programs. In a paper published a few years later (Fodor 1968b), Fodor repeats his view, already present in Fodor 1965, that psychological theories provide descriptions of psychological functions.

But this time, he adds that psychological theories are canonically expressed as lists of instructions: "the paradigmatic psychological theory is a list of instructions for producing behavior" (ib., p. 630). Although this flies in the face of the history of psychology, which is full of illustrious theories that are not formulated as lists of instructions (e.g., Freud's psychoanalysis, or Skinner's behaviorism),

Fodor does not offer evidence for this statement.[19] Fodor says that each instruction in a psychological theory can be further analyzed in terms of a list of instructions, which can also be analyzed in the same way. This does not lead to infinite regress because for any organism, there is a finite list of elementary instructions in terms of which all psychological theories for that organism must ultimately be analyzed. This type of explanation of behavior, based on lists of instructions, is explicitly modeled by Fodor on the relationship between computers, computer programs, and the elementary instructions in terms of which programs are ultimately formulated.[20]

Fodor does not distinguish between functional analysis and the analysis of capacities in terms of lists of instructions, nor does he discuss the relationship between the two. On the contrary, he discusses the two as if they were the same. However, in light of sections 8.3 and 8.4.2 above, I reserve the term "functional analysis" for an analysis that partitions a system into components and ascribes proper functions to the components. I will call the analysis of capacities in terms of lists of instructions *task analysis*. There are good reasons to keep these notions distinct. A functional analysis postulates a set of component types and their proper functions, which in turn can be given a functional analysis, and—unlike a task analysis—is not committed to analyzing the behavior of the system into sequences of elementary operations of finitely many types. On the other hand, a task analysis explains a capacity of a system as the execution of a sequence of operations of finitely many types by that system, which need not be analyzed into components and their proper functions. When a capacity of a system is explained by appealing to the causal role of a series of instructions constituting a task analysis of that capacity, like in ordinary computers, they are given an explanation by program execution.

Fodor's goal in this paper is to defend intellectualist explanations in psychology against Ryle's (1949) criticisms. In Fodor's formulation, intellectualist explanations are explanations of psychological capacities that appeal to tacit knowledge of rules. In his view, psychological theories formulated as lists

[19] At the time of Fodor's writing, though, some psychologists did propose such a view of psychological theories (Miller, Galanter, and Pribram 1960).

[20] Notice that from this point on in the literature, computer programs tend to replace TM programs in formulations of functionalism, without a distinction being drawn between them. As I pointed out before, they are not the same kind of description, and they presuppose very different functional analyses.

of instructions offer intellectualist explanations of psychological capacities. He argues that, contra Ryle, there is nothing methodologically wrong with intellectualist explanations. Fodor analyzes intellectualist explanations as lists of instructions that are executed in the sense in which digital computer programs are executed, and uses this analysis to argue that intellectualist explanations involve no infinite regress. But in order for these explanations to be properly intellectualistic, that is, different from the kind of causal process that Ryle would accept as an explanation of behavior, Fodor needs to stress that intellectualist explanations are significantly different from explanations in terms of generic causal processes:

> [T]he intellectualist is required to say not just that there are causal interactions in which the organism is unconsciously involved, but also that there are unconscious processes of learning, storing, and applying rules which in some sense "go on" within the organism and contribute to the etiology of its behavior (Fodor 1968b, p. 632).

This distinction between psychological explanations that appeal to mere causal interactions and those that appeal to genuine tacit knowledge, or rule following, or program execution, will be accepted in the literature as what makes computational theories of mind different from theories that are not computational. Without this distinction, computational theories of mind cannot be distinguished from neurophysiological theories that have nothing intellectualistic about them, because they do not appeal to the possession of knowledge, rules, or other "mentalistic" constructs. This distinction is reminiscent of Putnam's 1960 point that the analogy between men and TMs is such that both are open to rational criticism. This can be true only if the sense of program execution that is being used in the analogy is something more than a mere causal happening; it has to be a sense in which program execution is a rational process. Similar distinctions would be used in the literature on computationalism by supporters and critics alike to identify the difference between genuinely computational theories of mind and theories that could not be distinguished from "purely" neurophysiological or functional theories (Fodor 1975, p. 74, n. 15; see also Fodor 1968b, p. 632; Dreyfus 1979, pp. 68, 101-102; Searle 1980, pp. 37-38; Searle 1992, p. 208).

Of course, the distinction between explanation by program execution and other forms of explanation does not entail by itself that minds are programs, nor does Fodor suggest that it does. The

mind would be a program only if all psychological capacities, states, and processes could be correctly explained by appeal to program execution. For Fodor, whether this is the case is presumably an empirical question. On the other hand, Fodor's paper firmly inserts into the philosophical literature a thesis that begs the question of whether all psychological capacities can be explained by program execution: the thesis that psychological theories are canonically formulated as lists of instructions for producing behavior. This leaves no room for alternative explanations to be considered. In particular, the general type of functional analysis identified by Deutsch (1960), which explained mental capacities by postulating types of components and their proper functions, has been transformed, through Fodor's 1965 reinterpretation, into explanation by program execution.

After that, the mongrel of functional analyses and computer programs remained in the literature on functionalism and psychological explanation, where there are many statements to the effect that psychological theories are functional analyses, and that psychological functional analyses (or sometimes all functional analyses) are computer programs (or some other computational description). For example, similar conflations can be found in works by Dennett (1975, 1978c), Cummins (1975, 1983), Marr (1982), and Churchland and Sejnowski (1992).[21]

In summary, Putnam and Fodor formulated their analyses of minds and psychological theories in a way that makes it very natural to think that minds are programs or that mental states are states of a TM program. This is because they construed psychological theories as functional analyses, and functional analyses as TM programs. However, it should be clear by now that none of the considerations offered by Putnam and Fodor in support of their analysis of minds and psychological theories constitute a reason to

[21] Even Harman, who criticizes Fodor and Putnam's construal of functional analysis, follows their lead in this respect. Harman calls Fodor's explanations by functional analysis *narrow* because they attempt to explain an organism only in terms of its internal states, inputs, and outputs, without reference to how the system interacts with its environment. Harman argues that a complete explanation of an organism requires a *wide* functional story, i.e. a story that takes into account the relation between the organism and its environment. Nevertheless, even Harman identifies the narrow functional story about an organism with the description of the organism by a program (Harman 1988, esp. pp. 240-1).

believe that the mind is a computer program or that mental states are states of programs, that is, a reason in favor of computational functionalism.

8.5 Later Developments of Functionalism

Functionalism has been extensively discussed in the philosophical literature. Much of the discussion centers on functionalism in relation to folk psychology and reductionism or on attempts to refute functionalism *a priori*. As said in the introduction, these topics are irrelevant here. This section addresses elaborations of functionalism by philosophers who maintained Putnam's motivation: to give a metaphysics of states that are ascribed to individuals by scientific psychological theories. Later writers discussed difficulties faced by functionalism in dealing with certain aspects or paradoxes of mentality. While weakening or elaborating on Putnam's functionalism, these authors still maintained that their versions of functionalism are computational. But like Putnam, they gave no reason to believe that minds are programs. The goal of this section is to show that their views don't affect the conclusion that functionalism provides no motivation to believe that minds are programs. As examples, the views of Block and Fodor (1972) and Lycan (1981, 1987) are examined.

Block and Fodor (1972) argue for a version of functionalism that is inspired by, but slightly weaker than, Putnam's 1967b version. As we saw in section 8.2.3, in his 1967b paper Putnam identified mental states with states defined by certain TM programs. Block and Fodor discuss a number of problems that arise specifically from identifying mental states with states of TM programs.[22] Moreover,

[22] Block and Fodor enumerate the following problems: (i) a difficulty in drawing the distinction between dispositional mental states (beliefs, desires, inclinations, etc.) and occurrent mental states (sensations, thoughts, feelings, etc.) (ib., pp. 242-243), (ii) a difficulty in representing a set of mental states as occurring simultaneously (ib., pp. 243-244), (iii) a difficulty in ascribing the same mental state to two organisms unless they have identical TM programs (or relevant portions thereof) even though it is natural to ascribe the same mental state to two organisms in the presence of slight differences, say, in their behavioral dispositions associated to that state (ib., pp. 245-6), (iv) the fact that by definition there are only finitely many states of any TM program, but persons can be in infinitely many type-distinct psychological states, hence the two cannot be put in one-to-one correspondence (ib., pp. 246-247), and (v) a difficulty in representing structural relations among mental states (e.g., that believing that *A* is a constituent of believing that *A & B*) (ib., pp. 247-248). Block and Fodor also discuss the problem of qualia, arguing that qualia may not be functionally identifiable states at all. But, they note, if qualia cannot be identified

Block and Fodor seem to be aware that computational functionalism as formulated by Putnam is a version of the more general doctrine of functionalism, according to which mental states are identified by the causal relations they bear to inputs, outputs, and other mental states.[23] They end the paper saying that identity conditions for psychological states will be given by psychological laws when psychologists discover them, and it's a mistake to try to restrict such identity conditions by reference to things like states of TM programs, just as it was a mistake to restrict them to states falling under behavioral or neurophysiological laws. Psychological states should only be identified by psychological laws (ib., pp. 248-249). Nonetheless, in formulating their version of functionalism, far from reverting to a general formulation of functionalism that leaves it open for psychologists to devise their own functional descriptions, Block and Fodor restrict functional descriptions of mental states to what they call "computational states of automata," where a computational state is "any state of the machine which is characterized in terms of its inputs, outputs, and/or machine table states" (ib., p. 247). It is remarkable that, despite their appeal to psychological laws and their rejection of philosophical strictures on the identity of mental states, Block and Fodor still maintain that mental states are computational states, and conclude their paper saying that "[i]t may be both true and important that organisms are probabilistic automata" (ibid., p. 249). They give no reason for this conclusion, and just like Putnam gave no reason for why mental states should be states of TM programs, Block and Fodor give no reason for their restriction of functional descriptions to computational descriptions.

Like Block and Fodor, Lycan (1981, 1987) offers a version of functionalism inspired by Putnam's 1967b version. But unlike Block and Fodor, Lycan is concerned with the relationship between different levels of organization in the mind-brain. He argues that without some restriction on what counts as a realization of a TM program, a system realizes any TM program that is realized by the system's proper

---

functionally, qualia are irrelevant to the proper formulation of functionalism, so they ignore this problem (ib., pp. 244-245).

[23] For example, at some point they refer to "functionalism in the broad sense of that doctrine which holds that the type-identity conditions for psychological states refer only to their relations to inputs, output, and one another" (ib., p. 245).

parts.[24] He finds this putative consequence of Putnam's functionalism unacceptable. The reason is that, according to Lycan, it is conceivable that a proper part of an organism has mental states of its own. These premises, combined with the functionalist doctrine that having mental states is the same as realizing a certain TM program, entail that an organism has the mental states of all of its proper parts.[25] Lycan rejects this conclusion on intuitive grounds, and uses this rejection to motivate a restriction on the notion of realization (Lycan 1987, pp. 28-30). Lycan's restriction, introduced in Chapter 4 of his book, is a teleological requirement that has to do with what a functional state does *for* an organism. In formulating his version of functionalism, Lycan stops mentioning TM programs altogether. Building on the ideas of Attneave (1960), Fodor (1968b), and Dennett (1975a), Lycan argues that the best way to understand minds is to break them down into subsystems, each of which contributes to the activities of the whole by performing some intelligent task, such as solving a problem or analyzing a perceptual representation. Each mental subsystem can be further analyzed into sub-subsystems, and each sub-subsystem can be further analyzed until one reaches components that have no interesting "mentalistic" properties. He calls this picture *teleological* functionalism. Lycan defends his view at length, arguing that it is an appealing metaphysics of the mind and is not affected by his objection to Putnam's version of functionalism. In the rest of the book, he offers an account of intentionality and qualia based on his teleological version of functionalism. In formulating his teleological functionalism, Lycan makes no use of the notion of TM programs or any other computational notion. His discussion is couched in terms of systems and subsystems, and what they contribute to each other's activities.[26] Despite this, and despite the flaws that Lycan finds in Putnam's computational formulations of functionalism, he states that his version of

[24] Putnam's clause 3, cited in section 8.2.5, was designed to block this inference, but Lycan rejects it as unjustified.
[25] Incidentally, Lycan's argument can be modified to apply to Block and Fodor's version of functionalism.
[26] Although Lycan doesn't cite Deutsch (1960), his metaphysics of mind is reminiscent of Deutsch's view of phase one psychological theories. This is not surprising, because Deutsch inspired Fodor 1965, whose ideas carried over in Fodor 1968b, and Fodor 1968b is among the main acknowledged sources of Lycan's view. I argued above that Fodor distorted Deutsch's views to suit Putnam's analogy between minds and TMs; Lycan can be seen as revising Fodor's views in a way that is closer to Deutsch's original formulation.

functionalism is still a computational theory of mind.[27] He does not explain what it is about teleological functionalism that makes it a *computational* theory, or what metaphysical reasons there are to think that mentation has to do with computation.

8.6 Is Everything a TM?

There is one more element of the literature on functionalism that needs to be discussed: the view that everything is a (probabilistic) TM. The thesis that everything can be "described" or "interpreted" as a TM was already present in Putnam's writings on functionalism. For example, he wrote that "everything is a Probabilistic Automaton under some Description" (Putnam 1967b, p. 31).[28] If this is true, then even though functionalism about minds does not entail computationalism, this may not make any difference. If everything is a TM, then minds are TMs as well, and therefore computational functionalism holds without need for a separate argument to the effect that minds are TMs.

This thesis is often stated without argument, as if it was obvious, but that is not our worry.[29] Our worry is that its supporters say too little about how the view that everything is a TM fits with the view that computation is related to mentation.

Putnam introduced computation, and TMs in particular, in philosophy of mind because he thought that there is an important analogy between humans and TMs. Both minds and TMs can describe themselves and are "open to rational criticism" (Putnam 1960, p. 149). Fodor, the other early proponent of the link between computation and mentation, argued that what minds and computers have in common

---

[27] E.g., cf. the following passage: "… an articulate computational theory of the mind has also gained credence among professional psychologists and philosophers. I have been trying to support it here and elsewhere" (ib., p. 128).

[28] Cf. also: Block and Fodor 1972; "A [physical symbol] system always contains the potential for being any other system if so instructed" (Newell 1980, p. 161); "a standard digital computer … can display any pattern of responses to the environment whatsoever" (Churchland and Churchland 1990); "For any object there is some description of that object such that under that description the object is a digital computer" (Searle 1992, p. 208); "the laws of physics, at least as currently understood, are computable, and … human behavior is a consequence of physical laws. If so, then it follows that a computational system can simulate human behavior" (Chalmers 1996a, p. 329).

[29] Sometimes the thesis that everything is a TM is said to be, or to follow from, the Church-Turing thesis, i.e. the thesis that everything effectively calculable is computable by TMs. But this is based on a misunderstanding of the Church-Turing thesis (Sieg 1994; Copeland 1996, 2000). This topic is addressed in detail in Chapter 7.

is that they execute instructions (Fodor 1968b). It is because computing mechanisms have interesting characteristics that seem to be associated with mentality (are open to rational criticism, execute instructions, etc.) that many believe that computation offers the best hope for a mechanistic explanation of mind—a Computational Theory of Mind (CTM) (e.g., Fodor 1975). For present purposes, we do not need to settle what feature of computing mechanisms allows them to play an explanatory role in a theory of mind. It is enough that many people believe that computation has this explanatory role to play, and that we accept that TMs perform computations.

If everything is a TM, then minds are TMs. But if the fact that minds are TMs follows trivially from the fact that everything is a TM, it is unclear how computation could explain anything significant about minds, specifically anything about how minds exhibit their mental characteristics. In other words, if everything is a TM, it is unclear how being a computing mechanism could have anything interesting to do with being a mind.

The problem becomes more striking when the two claims above are combined with a third claim, which is quite popular among many of the same philosophers. That is, the view that some things are not computing mechanisms: "the solar system is not a computational system, but you and I, for all we now know, may be" (Fodor 1975, p. 74, n. 15).[30] In the same literature on functionalism and CTM, planetary systems, the weather, and stomachs are perhaps the most often cited paradigmatic examples of systems that are not computing mechanisms.

The trouble is that the last claim is inconsistent with the first, as the following simple derivation shows:

Premise 1. TMs compute.

Premise 2. Everything is a TM.

Premise 3. Some things don't compute.

4. Everything computes. [From 1 and 2]

[30] Cf. section 8.4.6 and the references listed therein.

5. Not everything computes. [From 3]

Conclusion. Everything computes and not everything computes. [Contradiction, from 2 and 5]

The obvious way to avoid a *reductio* is to drop one of the premises. Either some TMs don't compute, or not everything is a TM, or the weather and other putative examples of non-computing mechanisms do compute after all. None of these three solutions are appealing; I will discuss them in turn.

1) Few people if any are tempted to reject premise 1. TMs are used to define the notion of computation; they are the very paradigm of things that compute (Turing 1936-7).

2) Hypercomputationalism. Some have rejected premise 2 (e.g., Penrose 1994, Copeland 2000a). They divide the universe into things that are and things that aren't TMs. They face the challenge of drawing the line between these two classes. This is made difficult by the versatility of computational descriptions. The widespread use of computational models in most sciences might be taken as evidence that anything can be given a computational description (and hence a TM description). The hypercomputationalist answer is that things that are not TMs are hypercomputers, i.e. things that compute functions that are not Turing-computable. The most standard reply to hypercomputationalism is that, despite some bold attempts to argue that hypercomputers are physically possible (Penrose 1994, Copeland 2000a), little reason has been given to believe they can be physically built, let alone that they are relevant to explaining the mind. The controversy over hypercomputers is addressed in more detail in Chapter 7. In the present context, there is a more fundamental objection to hypercomputationalism: it does not avoid the above *reductio*. This is because hypercomputers still compute, so the *reductio* can be modified to apply to hypercomputers by replacing premise 1 with the premise that both TMs and hypercomputers compute, and premise 2 with the premise that everything is either a TM or a hypercomputer. As things stand, those who deny that everything is a TM have not yet provided a way out of the *reductio*. In the absence of a way to draw the line between things that are and things that aren't TMs without relying on what functions they compute, rejecting premise 2 does not solve the present problem.

3) Pancomputationalism. Finally, a few people have rejected premise 3 and accepted the consequence that everything—including stomachs and the weather—computes (e.g., Churchland and Sejnowski 1992). A general objection to pancomputationalism is that it's *ad hoc*: in the absence of independent reasons to abandon the intuitively plausible distinction between things that compute and things that don't, pancomputationalism seems just an attempt to save the other two premises by biting the bullet. From the point of view of the philosophy of mind, there is a stronger objection to pancomputationalism. TMs are discussed in philosophy of mind because many believe computation may have an explanatory role to play in a theory of mind. But if everything computes, the claim that the mind computes becomes trivially true, and the connection between computation and mentation is likewise trivialized. If everything computes, the notion of computation cannot play the explanatory role that computationalists were hoping for.[31]

Fudge factor. Given how unappealing the obvious solutions are, one wonders how computationalists managed to open this can of worms, and how they propose to close it. The Putnam of the 1960s and many others try to have it both ways: on the one hand, they appeal to the "mentalistic" properties of TMs (i.e. that they are open to rational criticism, execute instructions, etc.) to argue that computationalism sheds important light on the nature of mind; on the other hand, they appeal, implicitly or explicitly, to the fact that everything is a TM. Perhaps in an attempt to avoid pancomputationalist consequences, they also maintain that some TM descriptions are better than others, and that for any entity, there is a unique *best* TM description of that entity (Putnam 1967b). They then assert that it is only the best TM description of something that determines what something's mental characteristics are (Block and Fodor 1972). But this does not avoid pancomputationalism. If everything has a best TM description and TMs compute, it still follows that everything computes.

[31] Fodor notes: "suggesting … that *every* causal process is a kind of computation [trivializes the] nice idea that *thought* is (Fodor 1998, p. 12).

I offer a more satisfactory solution. The contradiction is eliminated by distinguishing different senses in which things can be given TM descriptions (or computational descriptions generally). Some senses are relevant to the philosophy of mind, others are irrelevant.[32] This is because different computational descriptions have different ontological implications about whether something computes in a sense that could explain its behavior, which is the sense that would be relevant to CTM. The important difference is between using a TM (or other computational model) to *describe* the behavior of a system and using computation to *explain* the behavior of a system.

The first is a case of modeling: a system's behavior under different conditions is described by the input-output relations of a TM (or other computational model). In modeling, generally the explanation for the system's behavior has to do with the properties of the modeled system, not with the computation performed by the model. The computation is performed by the model merely in order to generate descriptions of the modeled system. The situation is fully analogous to other cases of modeling: just like a system can be modeled by a diagram or equation without being a diagram or equation, a system can be modeled by a TM without being a computing mechanism.

In the second case, a system's behavior is explained by pointing at a certain computation and the properties of that computation. For instance, when we press the "√," "5," and "=" buttons of our calculator, we explain the calculator's output by pointing to the calculator's internal activity of computing square roots when certain buttons are pressed (*ceteris paribus*, i.e. if the calculator doesn't break, malfunction, etc.). Whether or not we use a TM to describe the calculator's behavior is independent of whether the explanation for that behavior appeals to a computation performed by the calculator. If we do use a TM to describe the calculator's *ceteris paribus* behavior, there will be two different calculations: the calculator's and the TM's. Given that calculators and TMs are subject to quite different functional

[32] Suggestions along these lines are present at least implicitly in the literature: "we wanted to know how the brain works, specifically how it produces mental phenomena. And it would not answer that question to be told that the brain is a digital computer *in the sense* that stomach, liver, heart, solar system, and the state of Kansas are all digital computers" (Searle 1992, p. 208; emphasis added). Searle, however, does not spell out the different senses in which things can be said to be TMs and their implications about whether things compute. Cf. also Block and Fodor 1972, pp. 240-241.

analyses, the two computations of square roots by the two systems would be very different. Nonetheless, the behavior of the calculator is explained by its performing a square root computation.[33]

According to my proposed solution, the above *reductio* is based on an equivocation. Premise 1 appeals to one sense of the statement that something is a TM, whereas premise 2 appeals to another sense. Computation explains the behavior of a system, and hence is relevant to the philosophy of mind, only in the sense used in premises 1 and 3. What follows is a taxonomy of senses in which something may be said to be a TM.

Sense one. "*x* is a TM" might mean that *x*'s states can be represented by the inputs and outputs of a TM, and representations of *x*'s state transitions can be computed by that TM. The representation of state transitions by the input-output relationships of a computer program is behind the widespread use of computational models in science, where a system's dynamical evolution is computed by a program on the basis of data representing the system's initial conditions. This can be done for any system whose dynamical evolution is known or can be approximated by known methods of calculation.[34] This seems to be the sense appealed to by premise 2.[35]

If one allows computational approximations to be arbitrarily distant from the dynamical evolution of the system being approximated, the thesis that everything is a TM becomes trivially true in sense one. If one is stricter about what approximations are acceptable, the thesis that everything is a TM in sense one depends on whether the system can be computationally approximated to the desired degree of precision.[36]

The statement that something is a TM in sense one applies to anything depending on what is known about its dynamical evolution. It applies equally well to things that are paradigms of computing

[33] In Chapter 10, I offer an account of what it means to explain a behavior by appealing to a computation.

[34] It may be known through a system of equations that can be solved analytically or whose solution can be approximated. It may also be known in other ways, e.g. through a series of measurements. For more on computational models in science, see Humphreys 1990 and Rohrlich 1990.

[35] According to Copeland, philosophers endorse premise 2 because either they misunderstand the Church-Turing thesis or they confuse premise 2 with Turing's demonstration that there are *universal* TMs (i.e., TMs that can simulate any other TM) (Copeland 1996, 2000a). I agree with Copeland's diagnosis but add a third element to it. Although philosophers who discuss premise 2 do not draw the present distinctions, what they say often suggests that their belief in premise 2 is implicitly driven by its plausibility in sense one.

[36] If one is strict about approximation and there are systems whose dynamical evolution involves uncomputable state transitions, then the thesis that everything is a TM in sense one would be strictly false. The question of whether there are such uncomputable state transitions is addressed in Chapter 7.

mechanisms (e.g., digital computers can be simulated or approximated by other computers) and to systems that are paradigms of non-computing mechanisms (e.g., the weather can be approximated by meteorological computer programs). What explains the system's behavior has to do with the properties of the system (which may or may not be computational), not with the computation performed by the model. In sense one, "*x* is a TM" says that *x* can be described by a TM in some respect for some modeling purpose. The TM description plays a modeling role fully analogous to the role played by differential equations, diagrams, and other modeling tools. Just like the same equation can describe systems that are physically very different, in sense one the same TM can describe systems that are physically very different. Just like the same system can be described by many different equations, some of which approximate its behavior better than others, in sense one the same system can be described by many different TMs, some of which approximate its behavior better than others. Just like being described by a system of equations does not entail being a system of equations, being described by a TM in sense 1 does not entail being a computing mechanism. So, sense one says nothing about whether something literally computes. It is not the sense appealed to by premises 1 and 3.

Sense two. Taken more literally, "*x* is a TM" might mean that *x* receives inputs, can be in internal states, and yields outputs and new internal states that depend on the inputs and old internal states, and there is a TM *m* such that *m*'s inputs represent *x*'s inputs, *m*'s internal states represent *x*'s internal states, *m*'s outputs represent *x*'s outputs, and for any input and internal state, *m* goes into the next internal state and generates the output representing *x*'s output and next internal state. Strictly speaking, sense two is not trivial; not everything is a TM in sense two. For most things don't seem to have (discrete) inputs, internal states, and outputs like TMs do, so it's not obvious how to compare them to a TM to determine whether they are the same.

A natural suggestion may be that, for any system, there is a TM whose state transitions map onto the state transitions of that system under its ordinary dynamical description.[37] This won't work. In modern science, systems are usually described mathematically not by means of TMs but by systems of

[37] The present point is a shortened version of a point discussed in section 7.3.3.

differential equations, which determine a continuous state space, which assigns an uncountable number of possible states and state space trajectories.[38] But TMs can only be in a finite number of states. Even if we combine a TM program states with the content of the machine's tape to increase the number of possible states, the total number of states that a TM can be in is only countably infinite. Moreover, TMs can only follow a countable number of state space trajectories. So TM descriptions don't have a cardinality of states and state space trajectories that is sufficient for them to map onto ordinary mathematical descriptions of natural systems. So, from the point of view of strict mathematical description, the thesis that everything is a TM in sense two cannot be supported.

Sense two can be loosened by allowing the TM description *m* of a system *x* to approximate rather than strictly map onto the internal states and behavior of *x*. This kind of approximation is behind the increasingly popular use of cellular automata, a kind of computing mechanism, as a modeling tool in physics (Rohrlich 1990, Hughes 1999). If one allows computational approximations to be arbitrarily distant from the behavior of the system being simulated, the thesis that everything is a TM becomes trivially true in sense two (unless there are state transitions that are strictly not Turing-computable).

Sense two is still irrelevant to the philosophy of mind, because it applies to anything depending merely on how discrete it is at the relevant level of description, i.e. on whether it has discrete inputs, outputs, internal states, and state transitions (and perhaps on one's criteria for acceptable approximations). For example, few people would count hard bodies as such as computing mechanisms. Yet, at a high level of abstraction hard bodies can be in either of two states, whole or broken, depending on how much pressure is applied to their extremities. The transition of a hard body from one state to the other can be approximated by a simple two-input, two-state TM, nevertheless there seems to be no useful sense in which this turns every hard body into a computing mechanism. If you hit an ordinary desktop computer sufficiently hard, you will break it. The resulting state transition of the computer, far from constituting a

[38] There are uncountably many real numbers. Any real number specifies a different initial condition of a dynamical system. For any initial condition, there is a separate state space trajectory. Therefore, there are uncountably many state space trajectories. This is true not only in physics but also biology, including neuroscience (for an introduction to theoretical neuroscience, see Dayan and Abbott 2001).

computation, will likely prevent the computer from performing computations in the future. So, sense two says nothing about whether something computes. It cannot be the sense appealed to by premises 1 and 3.

It may be replied that at the appropriate level of description, even a computer that breaks is performing a computation, albeit perhaps an uninteresting one. According to this line of thought, the same system can perform different computations at different levels of description, and the breaking of the computer is just one computation at one level of description among many. Ascribing computations in this way, though, does not make sense two relevant to the philosophy of mind. There are two reasons for this.

First, computation in sense two plays no explanatory role. What explains the breaking of a hard body is the physical properties of the body, the amount of pressure applied to it, and the relevant physical laws; different bodies of different shapes and hardness break under different pressures; the computational description that is common to them is *post hoc* and gives no information about when something will break. Unlike this example, cellular automata can have a nontrivial modeling role, but the important point still applies. The explanation for the system's behavior is given by the relevant physical laws together with properties and initial conditions of the system, not by the model's computation. One may still call what the system does a computation if one likes (cf. Wolfram 2002), but this kind of computation ascription cannot play an explanatory role in a theory of mind (or of computers, for that matter), so it's not what premises 1 and 3 should appeal to.

The second reason is that computations ascribed in sense two, unlike computations properly so called, cannot produce incorrect results. When a computer or person who is computing function *f* gives the wrong output for a given input, we say that a mistake was made (e.g., because of distraction in the case of a person, or component failure in the case of a machine). This possibility of specifying the function to be computed independently of the performance during execution, so that one can point to mistakes in the computation, is an important reason why computation appeals to philosophers of mind.[39] But there is no sense in which something that breaks under pressure can fail to generate the appropriate

[39] The issue of computation error is parallel to that of misrepresentation, which pertains to the semantics of mental states (Dretske 1986, Fodor 1990, Millikan 1993).

output: it is simply a law of physics that it will break under the appropriate pressure; the system can't do anything different. Even if we weaken or strengthen a system so that it won't break under the previous amount of pressure, there is no useful sense in which the modified system is doing something wrong. To the extent that the philosophy of mind requires a notion of computation such that mistakes can be made during computations, sense two is irrelevant to the philosophy of mind. Again, sense two is not what people who are interested in premises 1 and 3 should care about. This second consideration motivates the next sense of the statement that something is a TM.

Sense three. "*x* is a TM" might mean that *x*'s *proper function* is to generate certain outputs on the basis of certain inputs and internal states in a way specified by a certain TM program.[40] Sense three captures the feature of ordinary computation ascription that, given a certain input and a certain function to be computed, a computing mechanism *ought* to produce the right output. So sense three has nontrivial ontological implications for the system being described; it gives functional significance to the behavior of the system, namely it attributes the system the proper function of generating certain outputs given certain inputs. This can be used in functional explanations of the behavior of the system. In fact, only artifacts and biological systems are usually said to have proper functions, which may be invoked in explaining their behavior.[41]

Since proper functions are explanatory, sense three constitutes progress over the previous ones, but at the price of falsifying premise 2. Most physical systems don't have any proper functions, so the thesis that everything is a TM is false in sense three. The only way to make premise 2 true in sense three is to trivialize the notion of proper function to the point that the weather and planetary systems have proper functions, which few people are prepared to accept. Sense three provides a necessary condition for something to compute in a nontrivial sense: it entails that computing mechanisms can make mistakes in

[40] The proper function of computing should not be confused with the function being computed. The notion of proper function has been invoked in the literature on functionalism as a replacement for the notion of computational description (Millikan 1984, p. 139; Sober 1990). The appeal to proper functions is an advance over previous ways of talking about TMs in philosophy of mind, but its invocation as a *replacement* for computational descriptions is misplaced. Behavior characterized by proper functions can be described by TMs too. So sense three incorporates proper functions without abandoning computational descriptions.

[41] On functional explanation, see references in fn. 10 above.

their computations, just like ordinary calculators and computers. And in sense three, premises 1 and 3 are true while premise 2 is false.

As it stands, sense three is still inadequate to underwrite premises 1 and 3. For sense three applies to any artifact and biological system, including stomachs and other paradigmatic examples of non-computing mechanisms. It is still too broad to be directly relevant to the philosophy of mind in the sense of using computation (as opposed to proper function) to explain behavior. The sense that is relevant to the philosophy of mind must be restricted further.

Sense 4. "*x* is a TM" might mean that *x*'s proper function is to yield certain discrete tokens as outputs given certain discrete tokens received as inputs, and there is a TM *m* such that *m*'s inputs and outputs map onto *x*'s inputs and outputs. The input and output tokens are physical particulars that can be typed, and in nontrivial cases they are concatenated to form strings. If *x* is a TM in sense four, *x* deserves to be called a computing mechanism. Finally, *x*'s behavior can be explained by appealing to the computations *x* performs, and *m* is a computation-theoretic representation of *x*'s computation. Why did *x* generate such and such an output? Because *x* performs such and such a computation (which *m* identifies) on its input, and the appropriate output for the input received by *x* in this occasion is such and such. This is the (generalized version of the) notion of computing mechanism that was used in defining computational functionalism in section 8.3 above.

If *x* operates by moving an active device of the appropriate kind along a tape of the appropriate kind, then *x* can be said to be literally a TM. This is because *x*'s behavior is explained by a functional analysis that contains the relevant component parts and assigns them the relevant proper functions. Most physical computing mechanisms are not literal TMs, because their components and their components' proper functions are different from those of TMs. However, given the Church-Turing thesis, their computations can be exactly simulated by a TM, so they are TMs in sense four.

The formulation of computational functionalism offered at the beginning of this chapter entails that the brain is a TM in sense four, but not necessarily a literal TM. If the brain were literally a TM, then computational functionalism would be the thesis that the mind is the TM program for that TM.

Otherwise, according to (the generalized version of) computational functionalism, the mind is the computational organization of whatever computing mechanism the brain is.

With the above taxonomy in place, we can go back to our derivation and eliminate the equivocation that led to a *reductio*. In sense one, the thesis that everything is a TM is more or less trivially true depending on how strict one's criteria for acceptable computational approximations are. In sense two, the thesis that everything is a TM is *prima facie* false, but perhaps it can be made trivially true by sufficiently loose criteria for acceptable approximations. So these may be senses appealed to by premise 2. But "*x* is a TM" in either sense one or two is irrelevant to the philosophy of mind because it does not appeal to computation in a way that explains *x*'s behavior, so these cannot be the senses appealed to by premises 1 and 3. In sense three, "*x* is a TM" does not yet entail that *x*'s behavior is explained by its computation, but at least it provides a necessary condition for *x* to compute. And it makes the thesis that everything is a TM false, so it cannot be the sense appealed to by premise 2. In sense four, "*x* is a TM" finally entails that *x*'s behavior is explained by the computation is performs, but it applies only to a relatively small set of systems. The *reductio* is avoided by showing it to be based on an equivocation.

This solution has many advantages over the previous solutions. Solution one rejects the connection between being a TM and computing, which is paradoxical. My solution avoids the paradox by maintaining that connection for explanatory computational descriptions, while rejecting it for modeling computational descriptions. With respect to hypercomputationalism, my solution draws a principled line between the sense of being a TM relevant to establishing that something computes and the senses that are irrelevant. It does so without invoking computability or uncomputability, hence it avoids the *reductio*. With respect to pancomputationalism, I do not trivialize the Computational Theory of Mind (CTM). On the contrary, my solution lays the groundwork for seeing clearly why CTM is an empirical theory, as formulated in section 8.3 above.

Given my solution, different sources of evidence need to be used to support different versions of the claim that something is a TM, and the versions that are relevant to the philosophy of mind turn out to

be empirical in nature. This is the most important virtue of this solution, because naturalistically inclined philosophers would prefer to settle CTM empirically rather than *a priori*. When more work is done and we have criteria for telling which kinds of functional organizations are relevant for a system to be computational, it will be an empirical question to collect evidence about whether the mind has the appropriate functional organization. Another advantage is that, unlike any of the other solutions, the present one is consistent with the original formulation of the thesis that the mind-brain is a computing mechanism by McCulloch and other cyberneticians, who discussed it as an hypothesis specific to the functional organization of mind-brains (McCulloch and Pitts 1943, Wiener 1948, von Neumann 1958). It also coheres with the formulation of computationalism by many illustrious computationalists (e.g., Fodor 1975, Newell and Simon 1976, Pylyshyn 1984).

As to the thesis that everything is a TM, the upshot is that to the extent that it is true it is trivial, and to the extent that it is nontrivial it is false. In the interesting senses, the ones that cut some ice in the philosophy of mind, it is far from true that everything is a TM. Either way, the thesis that everything is a TM has no relevance to the philosophy of mind.

8.7 How Did This Happen?

As I argued so far in this chapter, in the literature on computational functionalism no reason for the computationalist component of computational functionalism has ever been given. Multiple realizability arguments either are based on the view that the mind is a program, or don't even mention programs. None of the traditional arguments for functionalism or for the view that psychological theories are functional analyses establish computationalism. I also argued that, although functional analysis and computational description have been systematically conflated in the literature, they are not the same, and thus explanation by functional analysis is not the same as explanation by program execution.

Finally, I argued that for computationalism to be relevant to the philosophy of mind, it cannot be a trivial consequence of the thesis that everything is a TM. A nontrivial computationalism about the mind must be based on a notion of computational description that carries explanatory force.

The lack of arguments for the computational component of computational functionalism raises the question of why, as a matter of historical fact, Putnam formulated functionalism in its computational variety and why his followers received Putnam's formulation without requiring a justification. A relevant fact, seen in section 8.4.2, is that Putnam reached his functionalist doctrine by developing an analogy between minds and TMs. Furthermore, it seems that some of the other theses present in this literature contributed to the impression that it is natural to think of minds in computational terms. One of those theses is Putnam's problematic thesis that everything can be described as a TM; another is Fodor's conflation of functional analysis and descriptions in terms of state transitions, which led to conflating giving a functional analysis and describing something as a program. Perhaps the combination of these views contributed to create the philosophical illusion that computational functionalism could solve the mind-body problem, explain the mind in terms of computation, and be based on some simple conceptual point such as multiple realizability or the thesis that everything can be described as a TM program. Once these different ideas are pulled apart, one can see that there is nothing left to the idea that functionalism as a solution to the mind-body problem entails computationalism as a substantive thesis about mind-brains. If my reconstruction is correct, computational functionalism can't be all those things at once. Either it is an empirical theory of mind based entailing the empirical hypothesis that the brain is a computing mechanism or it is a trivial thesis that has no role to play in the philosophy of mind.

The most important element of the explanation for the origin of computational functionalism is probably rooted in the history of science. At the time of Putnam's writing, the question of whether minds were machines and especially whether machines could think had become a hot topic of debate among scientists. In Chapter 6 we saw that by the late 1940s, members of the cybernetic movement proposed the construction of intelligent machines and popularized the idea that machines could think. Their view was that brains were computers, and that the appropriate kind of computers could think like brains. And in the

late 1950s, the new discipline of artificial intelligence (AI) was created with the explicit purpose of programming digital computers to produce behavior that was ordinarily considered intelligent. Some members of the AI community explicitly construed their research program as the construction of psychological theories of human thinking in the form of computer programs.[42] As a result, the debate around the relationship between mind-brains and machines centered on digital computers, whose theoretical model is the universal TM. MIT, where both Putnam and Fodor were working in the early 1960s, was one of the central institutions for cybernetics and AI. It is likely that this historical background made it natural for Putnam, Fodor, and later for other philosophers to assume that any machine capable of thinking, be it natural or artificial, had to be a computing machine (which, by the Church-Turing thesis, could be modeled by a TM).

Several lines of evidence suggest that Putnam and other philosophers conflated functionalism as metaphysics of the mind with computationalism as an empirical hypothesis about mind-brains. First, while putting forward his functional metaphysics of minds, which he defends and compares *a priori* to competing metaphysical views, Putnam says that functionalism is an "empirical hypothesis" (Putnam 1967b, p. 30). He does not explain in what sense it is empirical, nor does he discuss what kind of empirical evidence is supposed to support it or undermine it.[43] Second, by contrast with those who introduced and discussed functionalism in the context of *scientific* theories of mind, those who introduced and discussed functionalism in the context of *folk* theories of mind did *not* give a computational formulation of functionalism, nor did they claim that their views entailed that the mind is a computer, nor did they claim that functionalism was an empirical hypothesis (Lewis 1966, 1972, 1980; Armstrong 1970). This may be because they, not being concerned with psychological theories, were not influenced by current discussions in psychology about minds and computers. Finally, Fodor recently stated that the two were initially conflated in the literature:

[42] On the history of classical artificial intelligence, see McCorduck 1979, Gardner 1985, Crevier 1993.
[43] Claims of this type are reiterated in the functionalist literature. Cf.: "[Computational functionalism] is a scientific or quasi-scientific or at any rate *a posteriori* theory of mind" (Lycan 1987, p. 24).

> Particularly striking in retrospect was the widespread failure to distinguish the computational program in psychology from the functionalist program in metaphysics; the latter being, approximately, the idea that mental properties have functional essences… (For an instance where the two are run together, see Fodor 1968[b].) (Fodor 2000, p. 104.)

Fodor is explicitly stating that initially, he and others did not distinguish between functionalism as a metaphysical thesis about mental states and computationalism, that is, the empirical hypothesis that mind-brains are computing mechanisms. If this is correct, it explains why philosophers who discussed functionalism in the context of scientific psychological theories never felt the need to justify the computational thesis that functional descriptions are computational.

8.8 Functionalism and Computationalism

In this chapter, I discussed two independent theses related to functionalism that are relevant to the investigation of computationalism.

(A) Functionalism about minds: Having a mind is the same as satisfying a certain functional description.

Whether (A) is true is irrelevant to the assessment of computationalism; what matters is that (A) doesn't entail computationalism.

(B) Computational functionalism about minds: Minds are programs running on brains.

(B) is a form of computationalism, but neither Putnam nor any of his commentators gave reasons to believe (B). It appears to be an idiosyncratic formulation of functionalism for which there is no independent metaphysical motivation.

As a consequence, computationalism about the brain is either a trivial thesis that has no relevance to the philosophy of mind, or a non-trivial empirical hypothesis. If it is the latter, it is the thesis that the mind is the program of the brain, which entails that brains are mechanisms that store and execute programs. If brains are computers of this kind, it may or may not be true that minds are the brains' programs; perhaps there are aspects of the mind that have to do with other properties of brains, e.g. their phenomenal qualities. But if brains turn out not to be computers, then computational functionalism is

false. So whether or not one agrees with computational functionalism, one can still focus on whether the brain is a computer and see whether it's true.

The discussion of functionalism in philosophy of mind has made it difficult to discuss computationalism in a productive way. It convinced many philosophers that computationalism is a philosophical thesis, to be discussed by philosophical arguments and thought experiments, and to be judged by the extent to which it solves philosophical problems. This led philosophers to ignore the fact that, in so far as it has empirical content, computationalism is an empirical scientific hypothesis, which comes in several varieties that deserve to be assessed on grounds that are largely empirical.

Philosophers have also found another way to motivate computationalism *a priori*. That is, the hope that computationalism helps give a naturalistic explanation of the intentionality of mental states. Since intentionality is traditionally believed to be difficult to explain naturalistically, any theory that purports to offer a naturalistic explanation of intentionality deserves philosophical attention. Accordingly, the next chapter is addresses the claim that computationalism helps give a naturalistic explanation of intentionality.

9 COMPUTATION AND CONTENT

9.1 Introduction

The notion of content has played an important role in formulations of computationalism. Since the literature on computation and content is largely focused on the intentionality of *mental* states, in this literature computationalism is named Computational Theory of Mind (CTM). Following this usage, in this chapter I adopt the acronym CTM. This chapter discusses the role played by content in formulations of CTM and analyzes the relation between computation and content.

Some things, such as names and predicates, have content, i.e. they represent or are about things. I will refer to descriptions that ascribe content to something, such as "'*b*' represents *b*" and "'*p*' means that *p*," as *semantic* descriptions. Some contentful things, such as beliefs and desires, have satisfaction conditions, i.e. they can be true or false, fulfilled or unfulfilled, etc. I will refer to descriptions that ascribe satisfaction conditions to something, such as "*x* believes that *p*" and "*x* desires that *p*," as *intentional* descriptions, and to the content they ascribe as intentional content. This is not meant as a definition of content but as a way to anchor the following discussion to paradigmatic cases. Paradigmatic cases of contentful things include names and predicates; paradigmatic cases of intentional things include beliefs and desires.

A popular view about computing mechanisms is that in performing their computations, they are "insensitive" or "indifferent" to the content of computational states (as well as inputs and outputs; I will omit this qualification from now on); rather, they are sensitive only to non-semantic properties of computational states. The properties to which computing mechanisms are sensitive are often characterized as "formal" or "syntactic."[1] The same point is sometimes put by saying that the causally

[1] The locus classicus is Fodor 1980: "computational processes are both *symbolic* and *formal*… What makes syntactic operations a species of formal operations is that being syntactic is a way of *not* being semantic" (Fodor 1980, p. 64). See also Newell 1980.

efficacious properties of computational states supervene on their intrinsic physical properties. I will call this the *non-semantic view of computational causation*.

Another popular view about computing mechanisms is that their states are essentially endowed with content, so that their identity conditions must be specified using semantic or even intentional descriptions. I call this the *semantic view of computational states*. There has been a debate about the proper way to ascribe content to computational states.[2] There has also been a debate about whether the content of computational states is individualistic or narrow (i.e., it supervenes on intrinsic properties of the computing mechanism) or wide.[3] We need not be concerned with the resolution of these debates; here we focus on the semantic view of computational states, which is shared by almost all participants.[4]

Before proceeding, a potential source of confusion needs to be eliminated. Some supporters have pitched the semantic view of computational states against the non-semantic view of computational causation. They argue that, since computational states are identified semantically, the non-semantic view of computational causation should be rejected (Dietrich 1989; Peacocke 1994a, 1999; Shagrir 1999). This presupposes that the semantic view of computational states and the non-semantic view of computational causation are incompatible. Perhaps this is true, but none of these authors have yet provided arguments to this effect. As things stand, their conclusion is unwarranted. It seems possible that computational states must be identified semantically but are causally efficacious by virtue of their non-semantic properties. As a matter of fact, some of the staunchest supporters of the non-semantic view of computational causation are also some of the most forceful believers in the semantic view of computational states.[5]

---

[2] E.g., Harman 1987, Dennett 1987, Fodor 1998. The vast literature on theories of content will be discussed below.

[3] Much of this debate has focused on whether the computational theory of vision associated with David Marr (Marr 1982) ascribes wide or narrow content to computational states, and whether it identifies the computational states by their content (Burge 1986; Butler 1996; Davies 1991; Egan 1992, 1995, 1999; Segal 1989, 1991; Shapiro 1997).

[4] Two prominent exceptions to the consensus are Stich (1983) and Egan (1992, 1995, 1999). Their views are discussed below.

[5] Cf. Fodor: "I've introduced the notion of computation by reference to such semantic notions as content and representation: a computation is some kind of content-respecting causal relation among symbols… Suppose, however, it's your metaphysical view that the semantic properties of a mental representation depend, wholly or in part, upon the computational relations that it enters into; hence that the notion of a computation is *prior* to the notion of symbol. You will then need some *other* way of saying what it is for a causal relation among mental representations to *be* a computation; *some way that does not presuppose such notions as symbol and content*. It may

The aim of this chapter is to begin to sort out the relationship between computation and content, and thus to sort out the relationship between CTM and theories of content. I will argue for the following:

(1) Computational states are identified by their functional properties (as given by a functional analysis of the mechanism), in a way that does not involve semantic properties. This may be called the *functional view of computational states*. Although it runs against the semantic view of computational states, it is compatible with the ascription of content to computational states so long as this content is not seen as an essential property of them.

(2) The semantic view of computational states, according to which computational tokens and states are identified by their semantic properties, should be abandoned—quite independently of the functional view—because of two shortcomings: (a) under any existing theory of content, ascribing content to computational states presupposes a non-semantic identification of computational states, and (b) the semantic view begs the question of the computationalist who is also an eliminativist about content.

(3) Corollary 1: Computation ascription does not entail content ascription, so CTM does not presuppose a theory of content and it is not necessarily a representational theory (*contra*, e.g., Fodor 1998).

(4) Corollary 2[6]: Computation ascription is independent of content ascription, in the sense that it is legitimate to do the one without the other and *vice versa*. CTM and the Representational Theory of Mind (RTM) address independent (orthogonal) problems. CTM should be formulated and discussed without any theory of content, indeed without even presupposing that minds have content, so as to avoid getting entangled with the difficult issue of mental content. (I offer an account that satisfies this constraint in Chapter 10.)

---

be possible to find such a notion of computation, but I don't know where" (Fodor 1998, p. 11). Cf. also Pylyshyn 1984, p. 30.

[6] In drawing this further conclusion, I'm relying on the premise that content ascription does not entail computation ascription. Since I'm not aware of any claim to the contrary, I will not argue for it.

I will proceed as follows. In section 9.2, I will formulate the functional view of computational states and defend it on the basis of computability theory and computer design. In section 9.3, I will respond to arguments in favor of the semantic view of computational states. In section 9.4, I will reconstruct the history of the semantic view and argue that it has been generated by historical accretion rather than by a careful analysis of computing mechanisms. In section 9.5, I will discuss the relationship between CTM and various theories of content, arguing that all of the existing theories of content presuppose a non-semantic view of computational states. In the final sections, I will discuss some consequences of the functional view of computational states.

Although I do believe that minds have content, for the purposes of this chapter I will be neutral on whether they do or not.

9.2 The Functional View of Computational States

In the mathematical theory of computation, abstract computing mechanisms are identified by formal descriptions, some of which are called programs (see Chapter 1). Programs and other formal descriptions of computing mechanisms specify what inputs may enter the mechanism, how the inputs affect the internal states of the mechanism, and what outputs come out of the mechanism. Inputs and outputs are strings of letters from an alphabet, often called symbols. In computability theory, symbols are typically marks on paper identified by their geometrical shape. Symbols and strings of symbols may or may not be assigned an interpretation; if they are interpreted, the same string may be interpreted differently, e.g. as representing a number, or a program, etc., depending on what the theorist is trying to prove at any given time. In these computational descriptions, the identity of the computing mechanism does not hinge on how the strings are interpreted.

For example, the best-known computational formalism is that of Turing Machines (TMs, described in detail in Chapter 1). TMs are defined by listing instructions of the form: if on the tape there is a certain mark and the machine is in a certain state, then the processing device prints a certain mark,

moves one step to the left or right, and goes into a certain state. A specific TM is uniquely identified by its list of instructions. Nothing in these descriptions involves any content ascription; they simply describe how the hypothetical processing component of the TM reacts to the presence of certain marks on the tape while in certain internal states.

Although the inputs, outputs, and internal states of computing mechanisms are often assigned interpretations, sometimes it is useful to describe computing mechanisms without assigning any interpretation to their inputs, outputs, and internal states. A good example is a TM discovered by J. Buntrock and H. Marxen in 1989 (cited by Wells 1998). It uses two symbols and has only five possible internal states. This machine is offered as a demonstration of how difficult it is to predict the behavior of a TM from its abstract description, and how a very simple TM can have very complex behavior. When started on a blank tape, this simple TM halts after executing 23,554,764 steps. As Wells describes it, nothing in Buntrock and Marxen's TM has any content under anybody's notion of content; yet a computability theorist has no difficulty in recognizing it as a computing mechanism, uniquely identified by its instructions. The identity of TMs is determined by their instructions, not by the interpretations that may or may not be assigned to their inputs and outputs. The whole mathematical theory of computation can be formulated without assigning any interpretation to the strings of symbols being computed (e.g., Machtey and Young 1978).

In the practice of computer programming, however, programs are created by combining instructions that are *prima facie* contentful. For example, a high-level programming language may include a control structure of the form UNTIL P TRUE DO ___ ENDUNTIL.[7] In executing this control structure, the computer does ___ until the variable P has value TRUE and then moves on to the next instruction. The programmer is free to insert any legal sequence of instructions in the ___, knowing that the computer will execute those instructions until the value of P is TRUE. This awesome ability of computers to execute instructions is one of the motivations behind the semantic view of computational

[7] I took this example from Loop Programs, a simple but powerful programming language invented by Robert Daley at the University of Pittsburgh.

states. For when people execute instructions, i.e. they do what the instructions *say* to do, they do so because they *understand* what the instructions say. By analogy, it is tempting to conclude that in some sense, computers respond to the semantic properties of the instructions they execute, and hence the instructions and the corresponding computational states of the mechanism are identified by their content. This temptation is innocuous to the extent that one understands how computers execute instructions and specifies the relevant notion of computational content accordingly; otherwise, to speak of computer responding to semantic properties is misleading.[8] So I will briefly explain how computers execute instructions.

In ordinary stored-program computers, instructions are encoded as binary strings (strings of bits). Each bit is physically realized by a voltage level in a memory cell or some other state capable of physically affecting the computer in the relevant way. Before the processor of a computer can execute a binary string written in a high-level programming language, the computer must transform the string into a different binary string. The computer has a mechanism, typically a so called compiler, which takes binary strings encoding high-level programming language instructions as inputs and outputs binary strings encoding assembly language instructions. In assembly language, complex instructions such as UNTIL P TRUE DO ___ ENDUNTIL are replaced by sequences of instructions that refer to operations that can be immediately executed by the processor, and variables like P are replaced by names of memory registers.[9] Like high-level programming language instructions, assembly language instructions are encoded as binary strings. But from the point of view of the programmer, all that assembly language instructions do is tell the computer to transfer certain bits from one register to another and to perform primitive operations on those bits. (Primitive operations include arithmetical operations, such as addition and multiplication, and logical operations, such as conjunction and disjunction.)

---

[8] For example, Dietrich (1989) argues that since computing mechanisms respond to semantic properties of computational states, the non-semantic view of computational causation should be rejected. The following considerations explain why Dietrich's conclusion is unwarranted.

[9] An English rendition of assembly language instructions sounds somewhat like the following: COPY THE CONTENT OF REGISTER 20034 INTO REGISTER 20054, ADD THE CONTENT OF REGISTER 30005 TO THE CONTENT OF REGISTER 20067 AND PUT THE RESULT IN REGISTER 30005, etc.

This still makes it sound like the computer understands its instructions: the assembly language instructions are interpreted by programmers as referring to primitive operations, primitive operations are interpreted as arithmetical or logical, and certain bit sequences are interpreted as naming registers. But before an instruction can be executed, it must still be transformed into a new form. There is another mechanism, called assembler, which transforms assembly language instructions into machine language instructions, which the machine can execute. A machine language instruction is a binary string that, when placed into the appropriate register of a computer processor, *causes* the computer's control unit to generate a series of events in the computer's arithmetic-logic unit or memory or both. The sequence of events may include the transfer of binary strings from one register to another, the generation of new strings from old ones, and the placement of the new strings in certain registers. For example, the control unit may generate three signals and send them as inputs to the arithmetic-logic unit: one signal sets up the arithmetic-logic unit to perform a 32-bit addition and the other two signals act as inputs for the addition. Then, the control unit receives the output from the arithmetic-logic unit and places it in an appropriate register.

The crucial point is that the computer is designed so that the operations performed by the computer's components in response to a machine language instruction correspond to what the instruction means *in the relevant programming language*. The assignment of content to instructions, to the effect that an instruction asserts what its execution accomplishes *within* the computer, may be called *internal* semantics of the computer. An internal semantics should be kept distinct from an *external* semantics, which relates states of something to things other than the thing itself and its components. (The semantic view of computational states and the theories of content discussed in this paper are concerned with external semantics.)[10]

At the level of machine language execution, instruction execution is explained by the functional analysis of the mechanisms in terms of how various sequences of bits (causally) affect certain circuits.

[10] Dennett 1987 uses the expressions "internal semantics" and "external semantics" in roughly the same sense. The distinction between internal and external semantics should not be confused with that between semantic internalism and semantic externalism, which pertain to the identity conditions of contents (specified by an external semantics).

Since machine language instructions implement assembly language instructions (in the sense described above), and assembly language instructions implement high-level programming language instructions, the automatic shifting of bits from register to register, coupled with the generation of new binary strings from old ones, corresponds exactly to the execution of complex instructions like UNTIL P TRUE DO ___ ENDUNTIL. In this sense, the internal semantics of a computer turns out to ultimately assign only content about the shifting of bits between registers and the generation of new binary strings from old ones. Internal semantics is assigned to computer instructions without involving anything external to the computer. In other words, having an internal semantics does not entail having an external semantics.

Although assigning instructions an internal semantics is indispensable to programming (because the *programmer* must understand what the computer is going to do in order to write the program), from the point of view of explaining computer instruction execution, a complex instruction like UNTIL P TRUE DO ___ ENDUNTIL is a string of symbols, which will be encoded in the computer as a binary string, which the compiler will transform into another binary string, which the assembler will transform into another binary string, which will affect the computer's control unit in a certain way. A computer is a powerful, flexible, and fascinating mechanism, and we may feel compelled to say that it responds to the semantic properties of the instructions it executes. But as I briefly argued, this kind of "computer understanding" is exhaustively and mechanistically explained without ascribing any external semantics to the inputs, internal states, or outputs of the computer, and is fully compatible with the non-semantic view of computational causation. The case is analogous to TMs, whose computational behavior is entirely determined and uniquely identified by their instructions.

For reasons of space, the above discussion was brief. In Chapter 10, I explain instruction execution in more detail, I offer an account according to which computation ascription to a physical system entails no content ascription, and I argue that my account satisfies the desiderata of an account of computing mechanisms better than the competition. A computer is a physical system with certain input-output properties described by a certain kind of functional analysis. Although for practical purposes computers are usually ascribed content by an external semantics, this need not be the case and is

unnecessary to identify the functions they compute, to identify their computational states, and to explain their behavior.

This functional view of computational states goes beyond the non-semantic view of computational causation. It holds that the identity conditions of computing mechanisms, their states, and the functions they compute are determined by their functional properties. Even in the special case of stored-program computers, where the functional identification of computational states gives rise to an internal semantics, external semantics is not part of the identification of computational states. In other words, I reject the semantic view of computational states. From the functional view of computational states, it follows that computational descriptions are not *ipso facto* (external) semantic descriptions. So, if the functional view is correct, then the semantic view of computational states is wrong, and (external) content ascription must come from something other than computation ascription.

From now on, unless otherwise noted, by "semantics" I will mean external semantics, and by "content" I will mean content ascribed by an external semantics.

The functional view of computational states bears some similarity to a view proposed by Egan (1992, 1995, 1999). Egan rejects the semantic view of computational states, but there are three differences between our views. First, Egan believes that the identity conditions of computational states are individualistic, i.e. they supervene on physical states of the organism or machine, whereas the functional view of computational states is neutral about whether they are individualistic or not (for a defense of anti-individualism about computational states, see Wilson 1994). Second, although her view is that computational states do not have their content essentially, she nevertheless says that "computational states have content" (Egan 1999, p. 181). As I pointed out above, the functional view of computational states is neutral on whether computational states have content. Perhaps some do and some don't. Finally, and most importantly, Egan's putative non-semantic view of computational states is really a semantic view in disguise. This is because for Egan, the computational states are identified by the "mathematical" function whose domain and range elements are denoted by the inputs and outputs of the computation, i.e.

by a semantic property of the computational states (Egan 1995, p. 187).[11] From the present perspective, this is the semantic view all over again, which should be replaced by the functional view of computational states.

The above line of argument may be unpersuasive to someone firmly committed to the semantic view of computational states. She might prefer to use the semantic view of computational states as a premise and conclude that the functional view of computational states must be incorrect. I'm not sure what she would say about the fact that computability theorists identify abstract computing mechanisms, or that computer designers identify concrete computing mechanisms, independently of their content. But the fact that philosophers have maintained the semantic view of computational states for decades in the face of computability theory and computer design shows that she wouldn't be deterred. To address this possible reply, I will discuss arguments for the semantic view of computational states.

9.3 Against the Semantic View of Computational States

There are two main arguments for the semantic view of computational states on offer. The first pertains directly to computing mechanisms and their states, and goes as follows:

Argument from the identity of computed functions

(1) Computing mechanisms and their states are identified by the functions they compute.

(2) Functions are identified semantically, by the ordered couples <domain element, range element> *denoted* by the inputs and outputs of the computation.

---

[11] She contrasts her view with the one, championed by many philosophers, that computational states posited by psychological theories are identified by the contents of the mental states explained by the theory. See the next section for discussion.

Previously, Egan claimed that the proper taxonomy of computational states is given by a mapping between equivalence classes of physical features of a system and "symbolic" (Egan 1992) features. Egan is not very explicit about how this mapping is to be found. According to the functional view of computational states, the proper taxonomy of computational states is given by a certain kind of functional analysis of the system. For more details on the functional analysis of computing mechanisms, see Chapters 8 and 10.

(3) Therefore, computing mechanisms and their states are identified semantically.

Variants of the argument from the identity of computed functions can be found in the writing of several authors (Dietrich 1989, Smith 1994, Shagrir 1997, 1999, Peacocke 1999).[12]

The mistake in the argument from the identity of functions is to ignore that in talking about computation, functions can be identified in two ways. One appeals to the set of the ordered couples <domain element, range element> denoted by the inputs and outputs of the computation (for example {<1, 10>, <10, 11>, …}, where '1', '10', '11', … denote the numbers 1, 2, 3, ...). The other identifies functions as the set of ordered couples <input type, output type>, where input and output types are realized by the strings of tokens (or "symbols") that enter and exit the computing mechanism (for example {<'1', '10'>, <'10', '11'>, …}, where ''1'', ''10'', ''11'', ... denote inscriptions of types '1', '10', '11', ...). In other words, functions can be defined either over entities such as numbers, which may be the content of computational inputs and outputs, or over entities such as strings of (suitably typed) tokens, which are the inputs and outputs themselves. Both ways of identifying functions are important and useful for many purposes. Both can be used to describe what is computed by a computing mechanism. The relevant question is which of these ways of identifying functions is relevant to identifying computing mechanisms and their states.

In light of the previous section, the function description that is relevant to identifying computing mechanisms and their states is the one based on strings. The other may be useful for many other purposes, including explaining why people build computers the way they do and why they use them, but it is irrelevant to identifying computing mechanisms and their internal states.

Given a functional description of a computing mechanism that identifies the function being computed as defined over strings, one may ask how it is that that mechanism also computes the function <domain element, range element>, defined over numbers or other entities. In order to explain this, what

[12] Cf. Dietrich: "a correct account of computation requires us to attribute content to computational processes in order to explain which functions are being computed" (Dietrich 1989, p. 119).

is needed is a further fact: that the inputs and outputs of the computation *denote* the elements of the domain and range of the function. This is a semantic fact, which relates functionally identified inputs and outputs to their content. Stating this semantic fact requires that we identify the inputs and outputs of the computation independently of their denotations. So identifying inputs and outputs functionally, independently of their content, is a necessary condition for stating this semantic fact. So a functional (non-semantic) identification of computational states is a prerequisite for talking about their content.

Another problem with the argument from the identity of computed functions is that using the semantic values of the inputs and outputs does not identify computing mechanisms and their states as finely as we need when talking about computing mechanisms, and it is hard to see what other semantic properties should be added to the semantic values in order to reach an adequate degree of fine-grainedness. First, for any domain of objects (e.g., numbers), there are indefinitely many ways to represent it (notations). Second, for any function, there are indefinitely many algorithms that compute that function. Finally, for any algorithm, there are indefinitely many programs that implement that algorithm, using different programming languages and executed by indefinitely many computer architectures. Even within the same programming language or computer architecture, typically there are different ways of implementing the same algorithm. So the semantically identified function itself, or even the function in combination with the algorithm,[13] does not identify the computing mechanism and its states as finely as we need. This way of identifying computational states has the paradoxical consequence that mechanisms that have different architectures, use different programming languages, execute different programs that implement different algorithms (perhaps of different computational complexity) and manipulate different notations, are ascribed the same computational states only because they compute the same semantically identified function. To avoid this, we should allow not only the computed function (defined over strings), but also other functional (non-semantic) aspects of the computation, such as the

[13] Using algorithms in combination with semantically identified functions has been proposed in the literature as a way to identify computational states (e.g., Pylyshyn 1984). However, there is no accepted way to identify algorithms themselves, and some authors have expressed serious doubts that any satisfactory account of the identity conditions of algorithms is forthcoming (Dean 2002).

program and the architecture, to be part of the total functional analysis that identifies the computing mechanism and its computational states.

Moreover, using the semantic values of the inputs and outputs to identify computational states introduces spurious distinctions between computing mechanisms. The same physical mechanism performing the same manipulations of the same tokens may be ascribed different computational states—hence be identified as two different computing mechanisms—at different times simply because the interpretation of the tokens is different at those times. Since this is the same concrete mechanism working in the same way, explaining its behavior in terms of its internal states requires a language that is neutral between the different possible interpretations assigned to its inputs and outputs. A language that fulfills this purpose is the language of functional analysis. This is, as noted above, the language of choice of computability theorists and computer designers. The resulting identification of computational states is functional, not semantic.

The second argument for the semantic view of computational states appeals to computational explanations of mental processes:

Argument from the identity of mental states

(1) Computational states and processes are posited in explanations of mental states and processes (e.g., inference).

(2) Mental states and processes are identified by their semantic properties.

---

(3) Therefore, computational states and processes are identified by the semantic properties of the mental states and processes they explain.

Variants of the argument from the identity of mental states can be found in several authors (the most explicit include Pylyshyn 1984, Burge 1986, Peacocke 1994a, 1999).[14]

Premise 1 is uncontroversial; it simply takes notice that some scientists formulate computational explanations of mental states and processes. Premise 2 has been challenged (e.g., Stich 1983), but for the sake of the argument I will ignore the existing concerns about whether content can be legitimately used to identify mental states for scientific purposes.

As appealing as the argument from the identity of mental states may sound, it is a *non sequitur*. As Egan (1995) notes, the only way that the conclusion can be derived from the premises is by assuming that *explanantia* must be identified by the same properties that identify their *explananda*. This assumption is at odds with most, if not all, of our explanatory practices. For instance, suppose that I leave the room and my sandwich disappears from the table. Suppose the explanation is that my dog ate it. The identification of the *explanandum* (disappearance of the sandwich) depends on where the sandwich is, whereas the identification of the *explanans* (my dog's eating the sandwich) depends on what my dog did. For a more scientific example, consider the explanation of the tides in terms of gravitational forces. The tides are identified by the movements of seas along the earth's coasts, whereas their *explanans* (the motion of the moon and earth and the gravitational forces at work) are identified through a combination of astronomical observations and physical theory. Now consider the explanation of the capacities of a mechanism, which may be considered more similar in kind to the explanation of mental states and processes. For example, consider the explanation of digestion in terms of the secretion of certain glands in combination with the stomach's movements. The *explanandum* is identified by the properties of

[14] Cf. Burge and Peacocke:

> There is no other way to treat the visual system as solving the problem that the [computational] theory sees it as solving than by attributing intentional states (Burge 1986, pp. 28-29; cited by Egan 1995).

> One of the tasks of a subpersonal computational psychology is to explain how individuals come to have beliefs, desires, perceptions and other personal-level content-involving properties. If the content of personal-level states is externally individuated, then the contents mentioned in a subpersonal psychology that is explanatory of those personal states must also be externally individuated. One cannot fully explain the presence of an externally individuated state by citing only states that are internally individuated. On an externalist conception of subpersonal psychology, a content-involving computation commonly consists in the explanation of some externally individuated states by other externally individuated states (Peacocke 1994b, p. 224).

substances before and after they enter the stomach, whereas its *explanans* is identified by what the stomach and its glands do. These examples show that the identification of *explanantia* independently of their *explananda* is common in our explanatory practices. There is no reason to believe that this should fail to obtain in the case of explanations of mental states and processes. And without the assumption that *explanantia* must be identified by the same properties that identify their *explananda*, the argument from the identity of mental states doesn't go through.[15]

Computational explanations of mental states and processes offer a mechanism by which certain outputs are produced based on certain inputs and certain theoretically posited internal states. They also explain mechanistically how the theoretically posited internal states affect one another. To achieve this, computational explanations need not involve content at all. The fact that folk psychology identifies mental inputs, outputs, and internal states by their content does not impugn the appropriateness of computational explanations. The further question of the extent to which the internal states posited by computational explanations match those posited by folk psychology, whether the computational states have content or not, and whether they have the content postulated by folk psychology is a question entirely separate from the appropriateness of the computational (mechanistic) explanation. Perhaps some or all the internal computational states have contents that match the folk psychological states, as many computationalists believe (e.g., Fodor 1987, Pylyshyn 1984). Or perhaps they don't, as other computationalists maintain (e.g., Stich 1983, Dennett 1987, esp. chap. 5).

There is no doubt that most of the computational states ascribed by scientists to agents in explaining their mental states and processes are interpreted, i.e. they are ascribed content. There is also no doubt that this is very useful and perhaps indispensable in formulating and understanding computational theories (or at least the most common types of computational theories) of mental processes as well as in designing artificial intelligent agents. In particular, it may be practically impossible to specify how the computational states of an agent relate to the task the agent is performing without resorting to an interpretation of the computational states. In many cases, one understands the function of

[15] For a similar reply to the argument from the identity of mental states, see Egan 1995, p. 57ff.

a certain set of tokens and their manipulation by an agent when one is told what those tokens represent. Interpreting computational states relative to tasks also allows us to compare different mechanisms that are designed to solve the same task by relying on our understanding of the interpretation, without having to find and understand some non-semantic specification of the task. For instance, two different programs may be interpreted as playing chess, and one can be seen as playing chess better than the other, without needing to specify in a non-semantic way what playing chess consists in and what the properties of the mechanisms are.

None of this entails that the posited computational states are essentially semantic or intentional, i.e. that their identity conditions are stated in terms of their content. If anything, the need to ascribe content to computational states reinforces the conclusion that computational states are not essentially semantic. For even though certain states of two machines may have the same interpretations (e.g., chess positions), their different performance when playing the game forces us to distinguish between them. In order to identify computational states, we need to resort to the functional analysis of the mechanism. The same system of tokens and procedures may be ascribed different contents under different circumstances, and different systems of tokens and procedures may be interpreted in the same way even though they are computationally different. For example, two programs for computing the same semantically identified function—say, addition of natural numbers—may be different programs because they are written in different languages, even though they do addition using the same notation. And within the same programming language, two programs for addition using different notations are two different programs giving rise to different computational states, even though under their intended interpretation they compute exactly the same semantically identified function.

The point is not that content has no role to play in formulating and evaluating computational theories. It has many important roles to play, at least under the most common methodologies and assumptions. The point is that computational states and mechanisms have functional identity conditions, and that is all that is needed to individuate computing mechanisms and their states in the world. Once those conditions are fixed, interpretations may (or may not) be assigned to them.

Rejecting unsound arguments for the semantic view of computational states does not necessarily speak to what motivates its supporters. The main motivation behind the semantic view of computational states, which is well expressed by the argument from the identity of mental states, is the hope that computational states, being semantic or intentional, will help give a naturalistic solution to the problem of mental content.

Many philosophers who believe in CTM also believe minds have intentional content, namely they are truly described by intentional descriptions. There are many theories of intentional content. Some theories are reductionist in the sense that they aim at characterizing content in non-intentional and non-semantic terms, whereas other theories are not reductionist in this sense. CTM is a research program aimed at explaining the mind in broadly scientific or naturalistic terms. So, computationalists who believe in intentional content usually endorse reductionism about intentional content. The problem of explaining how minds have content in naturalistic terms, i.e. how content fits into the natural order, may be called the *problem of intentional content*.[16] At least since the 1970s, philosophers who are interested in naturalistic explanations of the mind and believe in intentional content have devoted a lot of attention to the problem of intentional content, developing several competing theories (e.g., Dretske 1981, Millikan 1984, Block 1986, Dennett 1987, Fodor 1990).[17] Since I'm concerned with computationalism, for present purposes I'm only interested in theories of content that are seen as complementary to CTM, and hence in reductionist theories of content. So, I will largely ignore non-reductionist theories of content.

According to the semantic view of computational states, computing mechanisms are analogous to minds in that they have content essentially. To distinguish it from intentional content, I will call the content of computing mechanisms *computational content*. (Later, I will discuss whether intentional content can be accounted for in terms of computational content.) The problem of how computing

[16] Fodor put is thus:

> [T]he main joint business of the philosophy of language and the philosophy of mind …: the metaphysical question of the place of meaning in the world order. How can anything manage to be *about* anything…? (Fodor 1987, p. xi).

[17] Besides solving the problem of intentional content, a reductionist theory of content may be required to satisfy other desiderata, many of which are listed by Block 1986. I will focus on the problem of intentional content and ignore the other desiderata of a theory of content.

mechanisms have content, analogous to the problem of intentional content, may be called the *problem of computational content*. Since CTM ascribes computations to the mind and the semantic view of computational states holds that computational descriptions are intentional, the conjunction of CTM and the semantic view of computational states entails that CTM is a representational theory, i.e. a theory that ascribes content to the mind. Because of this, many computationalists have supposed that solving the problem of computational content may be (perhaps part of) the solution to the problem of intentional content. Finally, the fact that CTM plus a theory of computational content is seen as a theory of intentional content is used as a reason in favor of CTM: we should believe CTM because it offers (a step towards) a naturalistic explanation of content.[18] So a thorough attempt to uproot the semantic view of computational states must include an explicit discussion of how computation relates to content, and thus how CTM relates to theories of content.

9.4 Origins of the Semantic View of Computational States

Part of my goal is to diagnose how the semantic view of computational states has become so entrenched in the philosophical literature, hoping this will add plausibility to my conclusion that it should be rejected. To do so, I will devote some attention to how computation and CTM were introduced in philosophy of mind, and how the issue of content was initially treated in the computationalist literature, so that computation and content got almost inextricably entangled. Doing this will lead naturally to an analysis of the relation between CTM and theories of content.

[18] This attitude is well expressed in the following passage, where the author seems to see no important distinction between semantic, computational, informational, and intentional descriptions:

> It is widely recognized that computation is in one way or another a symbolic or representational or information-based or semantical—i.e., as philosophers would say, intentional—phenomenon. Somehow or other, though in ways we do not yet understand, the states of a computer can model or simulate or represent or stand for or carry information about or signify other states in the world. … The *only* compelling reason to suppose that we (or minds or intelligence) might be computers stems from the fact that we, too, deal with representations, symbols, meanings, and the like (Smith 1996, p. 9-11; emphasis added).

9.4.1 Content in Early Computationalism

The modern history of computationalism, and the connection between computation and content, goes back to Alan Turing and the origin of computability theory. As we saw in Chapters 1 and 3, Turing formulated his theory of computation in the context of investigations on the foundations of mathematics. For instance, he wanted to prove that the decision problem for first order logic had no algorithmic solution. He formulated a theory of computation in terms of what are now called Turing Machines (TMs; see Chapter 1). Turing argued that some TM can carry out any computation process that a human being can carry out. In his argument, Turing described TMs anthropomorphically as "scanning" their tape, "seeing symbols," having "memory" or "mental states," etc., although he introduced all these terms in quotation marks, presumably to underline their metaphorical use (Turing 1936-7, pp. 117-118). Moreover, in using TMs for his mathematical purposes, Turing assigned interpretations to the inputs and outputs of TMs, usually as encoding real numbers, but sometimes as encoding TM programs or formulae in a logical calculus. So far, there is nothing methodologically problematic with what Turing did. TMs are to be understood as mechanisms for deriving strings of tokens, and the theorist is free to assign interpretations to the strings (within the relevant methodological constraints).

In the 1940s, a few years after the publication of Turing's theory, stored-program digital computers were built. In an article that became very influential, Turing argued that digital computers could be programmed to carry out conversations indistinguishable from conversations with humans (Turing 1950). In explaining digital computers to an audience who was likely to know little about them, Turing used intentional language again. He drew an analogy between digital computers and humans who calculate. The rules followed by a human are analogous to the instructions stored by the computer, and the human process of applying the rules is analogous to the computer's process of executing its instructions: "It is the duty of the [computer's] control to see that these instructions are obeyed correctly and in the right order" (ib., p. 437). Turing's analogy helped explain succinctly what digital computers do, and for that purpose, there is nothing objectionable to it. But it had the potential to suggest that computers somehow understand and obey instructions similarly to how people understand and obey

instructions. This analogy, by no means limited to Turing's writings, was a likely source of the semantic view of computational states.[19] When combined with CTM, the semantic view of computational states would be used to generate theories that seemed to explain the intentional properties of the mind.

As we saw in Chapters 2 through 4, the modern form of CTM was formulated by Warren McCulloch in the 1930s and published by him in the 1940s. McCulloch held that the brain is a computer and that thinking is computation. He also argued that CTM explains the possibility of human knowledge and solves the mind-body problem.[20] During the 1940s, CTM was adopted and elaborated by Norbert Wiener, John von Neumann, and other members of the newly forming cybernetics community, one of whose goals was to explain the mind by building computational models of the brain. During the 1950s, students and younger colleagues of McCulloch, Wiener, and von Neumann turned CTM into the foundation of the new discipline of artificial intelligence, whose goal was to explain the mind by programming computers to be intelligent.

Early computationalists described computers and neural mechanisms using semantic language. For instance, they said that computers (neural or artificial) "manipulate numbers," suggesting that something in the computer means or represents numbers:

> Computing machine are essentially machines for recording numbers, operating with numbers, and giving the result in numerical form (Wiener 1948, p. 137).
>
> Existing computing machines fall into two broad classes: "analog" and "digital." This subdivision arises according to the way in which the numbers, on which the machine operates, are *represented* in it (von Neumann 1958, p. 3; emphasis added).
>
> Thus the nervous system appears to be using a radically different system of notation from the ones we are familiar with in ordinary arithmetics and mathematics: instead of the precise systems of markers where the position—and the presence or absence—of every marker counts decisively in determining the *meaning* of the message, we have here a system of notations in which the

[19] For example, later we will examine writings by Fodor, who was an important early proponent of the semantic view of computers. In his early work on this matter, Fodor maintained that computational descriptions are semantic, and the primary reason he gave is that computers execute instructions (Fodor 1968a, 1968b). Only later did he add that computers operate on representations (Fodor 1975).

[20] CTM is often attributed to Turing (e.g., by Fodor 1998). Although Turing occasionally wrote that the brain is a computer (see the essays collected in Ince 1992), his statements to that effect were made after McCulloch's theory, which Turing knew about, had been published (McCulloch and Pitts 1943). I don't know of any place where Turing states that thinking is computation. In fact, Turing denied that "intelligence" and "thinking" are theoretically useful concepts (Turing 1950). McCulloch, however, explicitly held the view that thinking is computation (see e.g. the essays collected in McCulloch 1965). For more on Turing's views on intelligence, see Piccinini 2003a.

> *meaning* is conveyed by the statistical properties of the message (von Neumann 1958, p. 79; emphasis added).[21]

Early computationalists talked about computers having content, but they did not discuss how this was possible. They were concerned with building machines that exhibited intelligent behavior, not with philosophical issues about content. Accordingly, they did not address the problem of computational content explicitly. They did not even say explicitly whether they thought computers and brains have content in the same sense, although their writings give the impression that they thought so. McCulloch, for example, argued that CTM explains human knowledge, which suggests that the content of the computational states postulated by CTM explains the content of knowledge states. But he did not discuss how a computational state acquires its content or how this relates to the content of mental states.

Part of the reason for this lack of interest in the problem of content may be due to a certain operationalist spirit that early computationalists shared:

> The science of today is operational: that is, it considers every statement as essentially concerned with possible experiments or observable processes. According to this, the study of logic must reduce to the study of the logical machine, whether nervous or mechanical, with all its non-removable limitations and imperfections (Wiener 1948, p. 147).

> The problem of giving a precise definition to the concept of 'thinking' and of deciding whether or not a given machine is capable of thinking has aroused a great deal of heated discussion. One interesting definition has been proposed by A. M. Turing: a machine is termed capable of thinking if it can, under certain prescribed conditions, imitate a human being by answering questions sufficiently well to deceive a human questioner for a reasonable period of time. A definition of this type has the advantages of being operational or, in the psychologists' term, behavioristic. No metaphysical notions of consciousness, ego and the like are involved (Shannon and McCarthy 1956, p. v).[22]

This operationalism may have led early computationalists to discount questions about computational or intentional content on the grounds that either content could not be operationalized, or it had to be operationalized in non-semantic terms. Be that as it may, early computationalists formulated their CTM using semantic language, but they had no theory of content, and they gave no indication that they thought

[21] For a more comprehensive discussion of von Neumann's views on the computer and the brain, see Piccinini 2003b.

[22] Notice that Turing 1950 did not offer a definition of intelligence, let alone an operational one (cf. Moor 2001, Piccinini 2000). In reading Turing as offering an operational definition of intelligence, Shannon and McCarthy show how strong their own operationalist leanings were.

they needed a theory of content. This is probably the origin of the view that CTM ascribes content to the mind, and that it has something to contribute towards solving the problem of intentional content.[23]

9.4.2 Conceptual Role Semantics

Before discussing computationalism in philosophy, I should mention the theme of content in the philosophy of language. In the 1950s through 1970s, many philosophers were led to think about intentional content by thinking about linguistic content and attempting to formulate a semantic theory of language. By semantic theory, I mean both an assignment of content to language (a semantics) and an account of how language acquires its content. For present purposes, the second question is the important one.[24]

A natural way to account for linguistic content is to say that it comes from the content of the language users' minds. In other words, linguistic content can be explained by postulating that certain mental states of the language users have appropriate contents that are transferred to the speaker's utterances. Postulating intentional content as an explanation for linguistic content calls for a theory of intentional content.

A possible solution to the problem of intentional content is a theory that goes back to Ludwig Wittgenstein (1953) and Wilfrid Sellars (1954, 1974). According to this line of thought, linguistic content is constituted by the role played by linguistic structures within a community's linguistic behavior. Crudely put, linguistic content can be identified by observing what stimuli, perceptual or linguistic, generate the use of certain linguistic structures and what responses, linguistic or behavioral, are elicited by those linguistic structures. Sellars (1954, 1974) generalized the point to the content of mental states (specifically, thoughts). In Sellars's theory, thoughts are construed by analogy with linguistic sentences, so that intentional content is constituted by the relations of individual thoughts to inputs, outputs, and

[23] I believe that the cybernetic form of CTM, formulated using semantic language, later spread (in the same semantic form) into AI, psychology, and philosophy. I will not document this thesis here.

[24] On the history of philosophical discussions on content in American philosophy from the 1950s on, including many themes that I have no room to mention here, see Harman 1968, 1988, and especially Dennett 1987, chapt. 10.

other thoughts. Each thought is identified by its content, and its content is identified by on the one hand the inputs and other thoughts that elicit that thought, and on the other hand the outputs and other thoughts that are elicited by it. Different authors describe the role played by linguistic structures or mental states as either functional, or inferential, or conceptual role (henceforth, I will adopt conceptual role), and the resulting theory of content is correspondingly called functional, or inferential, or Conceptual Role Semantics (CRS).

With this landmark in place, we can go back to computationalism.

9.4.3 Computationalism and the Philosophy of Mind

Computation became an important notion in contemporary philosophy of mind through work by Hilary Putnam and his student Jerry Fodor in the 1960s (Putnam 1960, 1964, 1965, 1967a, 1967b; Fodor 1965, 1968a, 1968b).[25]

Putnam was familiar with some of the cybernetics literature, including McCulloch's CTM,[26] and like the cyberneticians, he did not seem concerned with formulating a theory of intentional content. In the first paper where he drew an analogy between minds and TMs (Putnam 1960), Putnam introduced computational descriptions to *dissolve* the mind-body problem. He argued that a problem analogous to the mind-body problem arises for TMs, and that this shows the mind-body problem to be a pseudo-problem. In the same paper, Putnam said that internal states of TMs are identified by their causal relations to inputs, outputs, and other internal states. Two years later (Putnam 1967a, delivered to the Wayne State University Symposium in the Philosophy of Mind in 1962), Putnam argued that mental states are identified in the same way that TM states are, but left open the question of whether minds are

[25] Some other strands in Putnam and Fodor's early computationalism, which do not pertain to the issue of content, are explored in Chapter 8.

[26] McCulloch and Pitts's theory is cited both in Oppenheim and Putnam 1958 and in Putnam 1964. So Putnam knew McCulloch and Pitts's theory before moving to MIT in 1961. During the early 1960s, both Putnam and Fodor were at MIT, which was perhaps the main center of cybernetics research as well as the home institution of Noam Chomsky, who was proposing to explain the human ability to manipulate language by postulating innate knowledge of a recursive (i.e. computational) grammar (Chomsky 1957, 1965). At that time, both Putnam and Fodor were close to Chomsky and his views (Putnam 1997).

TMs or something more complicated. In this occasion, he offered this characterization of minds as a solution to (as opposed to a dissolution of) the mind-body problem. Putnam's mind-body doctrine was going to be called functionalism. In its canonical formulation, called *computational* functionalism, it states that minds *are*, in fact, a kind of TMs (Putnam 1967b).

Fodor was influenced by Putnam and by the work of psychologists J. A. Deutsch (1960) and Stuart Sutherland (Fodor 1965, p. 161, and personal correspondence). Fodor's main goal was to give a philosophical account of psychological explanation. In his first paper on this subject, Fodor said that psychological theories are functional analyses, i.e. theories that describe a system in terms of internal states and state transitions that are specified by their functional role (Fodor 1965). Later, Fodor added that psychological functional analyses are lists of instructions, or programs, which explain behaviors by stating what internally stored instructions are executed by people engaged in those behaviors (Fodor 1968).[27]

Although in these writings Putnam and Fodor did not seem concerned directly with the problem of intentional content, there is considerable overlap between the themes of Putnam and Fodor's papers on the metaphysics of the mind in the 1960s and Sellars's theory of intentional content (Sellars 1954). The main common theme—expressed by different authors using different terminologies—is that mental states are to be identified by their functional role within a network of inputs, outputs, and other mental states. Another theme is the use of functionalism to dispense with arguments from the privacy of the mental to some special ontological status of the mental (Sellars 1954, Putnam 1960). Finally, there is the analogy between minds and computers. In this respect, Sellars says:

> [T]he learning of a language or conceptual frame involves the following logically (but not chronologically) distinguishable phases:
> (a) the acquisition of S[timulus]-R[esponse] connections pertaining to the arranging of sounds and visual marks into patterns and sequences of patterns. (The acquisition of these 'habits' can be compared to the setting up of that part of the wiring of a calculating machine which takes over once the 'problem' and the relevant 'information' have been punched in.)

[27] According to Harman (personal correspondence), this view of psychological theories was influenced by the view of some psychologists, who proposed a similar vision of psychological theories to replace the behaviorist stimulus-response view of psychological theories (e.g., Miller, Galanter, and Pribram 1960).

> (b) The acquisition of thing-word connections. (This can be compared to the setting up of that part of the wiring of the machine which enables the punching in of 'information.') (Sellars 1954, p. 333).

Despite the overlap between Sellars's functionalism in the 1950s and Putnam and Fodor's functionalism in the 1960s, there is strong evidence that Putnam and Fodor's views were developed independently of Sellars's. Neither Putnam nor Fodor cite Sellars in their papers, and according to both of them, in the 1960s they were not acquainted with Sellars's views on this matter (personal correspondence).[28] The development of Putnam and Fodor's functionalism independently of Sellars's functionalism helps explain why Putnam and Fodor did not discuss the problem of intentional content, did not distinguish between the classical mind-body problem and the problem of intentional content, and did not seem to think that after giving their (computational) functionalist solution to the mind-body problem, the problem of intentional content remained to be solved.[29]

---

[28] Dennett wrote that "[i]t is clear that Putnam['s functionalism has] … been quite directly influenced by Sellars" (Dennett 1987, p. 341). Dennett told me he got his sense of Putnam's debt to Sellars during a discussion of Sellars's views with Putnam, a discussion that took place in the early 1970s (personal correspondence). Putnam, however, told me he "arrived at functionalism quite naturally, being at the time both a philosopher and a recursion theorist" (personal correspondence). Speaking of his work in the late 1950s, he also wrote as follows:

> I was in the habit of explaining the idea of a "Turing machine" [fn. omitted] in my mathematical logic courses in those days. It struck me that in Turing's work, as in the theory of computation today, the "states" of the imagined computer (the Turing machine) were described in a very different way than is customary in physical science. The state of a Turing machine—one may call such states *computational* states—is identified by its role in certain computational processes, *independently* of how it is physically realized. A human computer working with paper and pencil, a mechanical calculating engine of the kind that was built in the nineteenth century, and a modern electronic computer can be in the *same* computational state, without being in the same physical state. I began to apply images suggested by the theory of computation to the philosophy of mind, and in a lecture delivered in 1960 [fn. omitted; the lecture was published as Putnam 1960] I suggested a hypothesis that was to become influential under the name *functionalism*: that the mental states of a human being are computational states of the brain (Putnam 1997, pp. 180-181).

Curiously, here Putnam follows a common pattern in the literature on functionalism, which attributes computational functionalism to Putnam 1960 even though Putnam didn't formulate it until Putnam 1967b. If Putnam's recollections are otherwise correct, Dennett's impression that Putnam's functionalism was indebted to Sellars's may be due to a misunderstanding between them.

[29] A fortiori, they did not discuss whether mental states have their content essentially, a view that is very popular nowadays. If one believes that mental states have their content essentially, then one will automatically see Putnam and Fodor's early functionalism as providing a theory of content. Otherwise, one will interpret their work as offering a theory of the identity conditions of mental states that is neutral about their content (cf. Jackson and Pettit 1988, p. 388).

9.4.4 The Semantic View of Computational States in the Philosophy of Mind

Another reason for ignoring the problem of intentional content may be that Putnam and Fodor formulated their functionalist theory of mind using computational descriptions, and—like the cyberneticians—Putnam and Fodor identified computers using semantic idioms.

In the papers cited above, Putnam was ambivalent on computational states and content. In his 1960 paper, he drew an analogy between mental states and TM states, and between introspective reports and TM self-descriptions, but he added that TMs cannot be properly said to use a language. Later (Putnam 1967b), he stated that mental states are TM states. Together with the generally shared premise that some mental states are intentional, which Putnam never rejected, this entails that at least some TM states are intentional.

Fodor argued that computational explanations in psychology were intellectualist in Ryle's sense (Ryle 1949). An intellectualist explanation accounts for an intentionally characterized overt behavior by postulating an intentionally characterized internal process. Ryle criticized intellectualist explanations; roughly speaking, he argued that they require the postulation of an internal homunculus to explain the intentionally characterized internal process, thereby generating an infinite regress of homunculi inside homunculi (Ryle 1949). Fodor (1968b) rejected Ryle's criticism of intellectualist explanations on the grounds that computers' activities are characterized in intellectualist terms but involve no infinite regress. Fodor built an explicit view about computation ascription around the idea that computational descriptions are intentional. For something to be a computing mechanism, there must be a mapping between its physical states and certain intellectualistic (hence intentional) descriptions of what it does:

> [A] programming language can be thought of as establishing a mapping of the physical states of a machine onto sentences of English such that the English sentence assigned to a given state *expresses the instruction* the machine is said to be executing when it is in that state (Fodor 1968b, p. 638, emphasis added).

> Every computational device is a complex system which changes physical state in some way determined by physical laws. It is feasible to think of such a system as a computer just insofar as it is possible to devise some mapping which pairs physical states of the device with formulae in a computing language in such a fashion as to preserve desired *semantic relations* among the formulae (Fodor 1975, p. 73; emphasis added).

These passages show that for Fodor, computational descriptions ascribe semantic properties to a mechanism and identify its states by those semantic properties. This is the semantic view of computational states. The semantic view is not a theory of content. A theory of content explains how something acquires its content. The semantic view of computational states is simply the view that computing mechanisms do have content; more precisely, that describing something as a computing mechanism is a way of ascribing content to it. The semantic view of computational states does not specify by virtue of which properties or conditions computational states acquire their putative content.

The blending of computational functionalism, CTM, and the semantic view of computational states culminates in Fodor's Language of Thought (LOT) hypothesis and his famous slogan "no computation without representation" (Fodor 1975). According to Fodor 1975, learning what a predicate in a public language means requires representing the extension of that predicate in some previously understood language—LOT. Now, if understanding LOT required representing the extension of *its* predicates in some previously understood language, this would lead to infinite regress. Fodor avoided this infinite regress by appealing to stored-program computers and the way they respond to their inputs and instructions.[30] Modern computers receive instructions as inputs written in some programming language and then transform those inputs into machine language code that they can execute. But executing code does not require a new transformation of internal code into another language, or to possess a representation of how to carry out the execution. Computers are hardwired to carry out certain elementary operations in response to certain lines of internal code, so the regress stops at those hardwired processes (Fodor 1975, p. 65ff.).

Fodor likened human public languages to high level programming languages, and the human LOT to the machine language of computers. He argued that LOT is our best explanation for human cognition, and specifically for the human ability to manipulate language and make inferences in a way

[30] Fodor also argued that the human LOT is innate. This component of Fodor's version of LOT, which was rejected by many who accepted LOT, is irrelevant to the present discussion.

that respects the semantic properties of thoughts.[31] As we saw above, Fodor described computers and their languages using intentional idioms, perhaps in part because his appeal to stored-program computers was intended to render LOT mechanistically intelligible. But LOT is not a theory of content. To be hardwired to execute certain lines of code is a very interesting property that stored-program computers have, but it cannot be identified with having content, much less having content corresponding to the content of human language and thought, without argument.[32] Fodor 1975 did not address how the process of translation between public language and LOT respected the semantics of the public language, namely how LOT acquired *its* semantics and managed to match it with the semantics of the public language. Fodor 1975 offered no solution to the problem of content, nor did he purport to offer such a solution.[33]

During the 1970s, CTM became very influential in philosophy of mind. Many philosophers accepted some version of CTM, even though some of them rejected one or another tenet of Fodor's LOT. At the same time, we will see that the main authors who discussed CTM sympathetically subscribed to the semantic view of computational states. They thought the mind is computational, and computational states have content, so they thought the problem of intentional content might be reducible to the problem of computational content. And since computing mechanisms are mechanisms built by humans, the problem of computational content may have seemed less philosophically pressing than the problem of

[31] This is one more similarity with Sellars's ideas, which included the postulation of an internal language as explanation for thought (esp. Sellars 1956). Sellars's ideas were turned into a systematic theory by Harman (1973, discussed below). Fodor told me he learned about Sellars and Harman's theory of thought only after writing his 1975 book (personal correspondence). He added that around the same time Zeno Vendler also wrote a book that proposed a similar theory of thought (Vendler 1972). Although Vendler preferred not to talk of an inner "language" (ib., pp. 42, 51), his theory postulated an innate neural "code" that could be scientifically deciphered (ib., p. 142). According to Vendler, he developed his theory by trying to improve on Austin's theory of illocutionary acts under the influence of Chomskian linguistics (ib., pp. viii, 4). Vendler did not refer to Sellars's theory of thought. Although Fodor told me he knows of no mutual influence on their respective versions of LOT between himself, Harman, and Vendler (personal correspondence), Harman recalls that Vendler attended a presentation of Harman's version of LOT in 1968 (personal correspondence).

[32] At most, it can be identified with having an internal semantics. But as I argued earlier, having an internal semantics does not entail having an external semantics. Unfortunately, Fodor 1975 did not distinguish between internal and external semantics.

[33] Nor did he make explicit that his theory presupposed such a solution. I insist on this because LOT is sometimes mistaken for a theory of content. For example, Putnam criticized Fodor's LOT as if it were a theory of content (Putnam 1988, pp. 21, 40-41). Perhaps this says something about how Putnam was thinking about computation and content. I took this reference to Putnam from Loewer and Rey, who also point out that LOT is not a theory of content (Loewer and Rey 1991, p.. xix). The reason why Fodor's LOT is misread as a theory of content may be partially due to how Fodor blended it with the semantic view of computational states, without distinguishing between internal and external semantics.

intentional content. Nevertheless, solving the problem of intentional content requires combining CTM with a theory of content.

9.5 Computationalism and Theories of Content

In this section, I argue that computation ascription alone is insufficient for the ascription of intentional content. If there were consensus about what intentional content is, I could run a general argument of the following form:

> Premise 1. Having intentional content is the same as satisfying condition C.
>
> Premise 2. Being a computational state does not entail satisfying condition C.
>
> Conclusion: Being a computational state does not entail having intentional content.

Since there is no consensus on what intentional content is, I will go through the main theoretical approaches to content and argue that in each case, ascribing content to computational states presupposes a non-semantic identification of the computational states.

A consequence is the rejection of the thesis that CTM contributes to solving the problem of intentional content, which eliminates this as a reason for believing CTM. My conclusion has no consequences about whether minds or computers have content, whether intentional and computational content are the same, and the project of reducing intentional content to computational content. All I argue is that those questions must be answered by a theory of content, not by a theory of computation or a CTM.

9.5.1 CTM meets Conceptual Role Semantics

Among computationalist philosophers, the first who took the problem of intentional content seriously was probably Gilbert Harman. Harman was familiar both with Putnam and Fodor's computational

functionalist writings and with Sellars's theory of content (Harman 1968).[34] His idea was to explicitly combine computational functionalism about mental states with a Conceptual Role Semantics (CRS) about their content, to have a naturalistic theory of intentionally identified mental states. According to CRS, the content of mental states is constituted by their conceptual roles. But CRS, as Sellars left it, did not specify a mechanism that could physically realize these conceptual roles. From a naturalistic perspective, a CRS for mental states calls for a theory of the mechanism realizing the conceptual roles. Computational functionalism had a mechanism that seemed to have the properties needed to realize conceptual roles (after all, at least under the semantic view of computational states, computing mechanisms draw *inferences*), while at the same time lacking a theory of the mechanism's content. By combining the two, Harman (1973) could use the one theory to solve the problem left open by the other, and *vice versa*. On the one hand, he appealed to the roles of computational states within a computing mechanism as appropriate realizers of the conceptual roles that ascribe content to mental states (ib., pp. 43-48). On the other hand, he appealed to those conceptual roles to ascribe content to the states of the mechanism (ib., p. 60). The result is a CTM in which the computational states and processes that constitute the mind are also the physical realizers of the conceptual roles that give the mind its content. In the same theory, Harman also maintained Sellars's construal of thoughts as analogous to linguistic sentences, i.e., as internal representations.

After Harman's revival of it, CRS found many advocates. Some accepted Harman's combination of CRS and a representational CTM (e.g., Field 1978). In a similar vein, Ned Block argued that CRS was the best available theory of intentional content to combine with LOT (Block 1986). Others took from Harman only CRS and a functional identification of mental states, while discarding the representational component (e.g., Loar 1981). Still others took only CRS without CTM (e.g., Churchland 1979).

For present purposes, it is important to understand the division of labor in the combined CTM-CRS theory. The content of a mental state comes from its conceptual role, so whether the component of

[34] In personal correspondence, Harman has added to these the influences of Miller, Galanter, and Pribram 1960 and Geach 1956.

the theory that accounts for intentional content is successful or not depends on whether conceptual roles are adequate to provide the semantics of mental states. The conceptual role of a mental state is (partially) identified by the computational relations that the state bears to other states, inputs, and outputs.[35] So, the computational states and relations must be identified in a way that does not presuppose their content, otherwise the theory of content becomes circular. In other words, a combination of CTM and CRS identifies content by conceptual role, and conceptual role (at least in part) by computational role. If the computational role is identified by appeal to content, as the semantic view of computational states would have it, CTM-CRS is running in a small circle. So, if CTM-CRS wants to avoid circularity, it needs a non-semantic view of computational states, i.e. a way to identify computational states and their relations without appeal to their content.[36]

As natural as it may seem to combine CTM and CRS, it is not mandatory. The two are logically independent. Even if one believes that content is constituted by conceptual roles, the physical realization of the conceptual roles need not be computational.[37] On the other hand, even if one believes that the mind is computational and that computational states have content, one need not believe that content is constituted by the conceptual roles of the computational states. For a first important alternative theory of content for minds and computing mechanisms, we will look at the tradition that follows Daniel Dennett.

---

[35] CTM-CRS theorists do not identify *content* itself with computational role, however. Some, like Harman, postulate that the relations between inputs and outputs on the one hand, and the environment on the other, also contribute to the conceptual role and hence to content (broad CRS); others, like Block, postulate that conceptual roles are only one of the factors determining content, the other being reference (narrow CRS). These subtleties make no difference for our purposes; see Block 1986 for discussion.

[36] I don't know of any place where CTM-CRS theorists have offered such an account. Block 1986 says he doesn't know how to identify conceptual roles. To a direct question, Harman answered thus: "I don't think I have ever tried to provide a noncircular theory of content in that sense" (personal correspondence).

[37] To say more about what a non-computational realization of conceptual roles would be would take us too far afield. Given my account of computing mechanisms in Chapter 10, CRS could be combined with a non-computational functional analysis of the mind-brain, i.e. a functional analysis that explains cognitive processes without ascribing computations to the mind-brain. For versions of CRS that are not committed to computational roles as physical realizers of conceptual roles, see Peacocke 1992 and Brandom 1994.

9.5.2 CTM meets Interpretational Semantics

Like Harman, Dennett was familiar with both computational functionalism and some of Sellars's work, and he was interested in the problem of intentional content. Dennett also continued the tradition of his mentor Ryle, whose analytical behaviorism rejected the postulation of intentionally characterized mental states (Ryle 1949). Following Sellars and Putnam, Dennett did develop a version of functionalism about content, according to which intentional content is constituted by the mutual relations between contentful states, inputs, and outputs (Dennett 1969, chap. 4).[38] But Dennett's conceptual roles, unlike Harman's, were not realized by computational roles (Dennett 1969, 1971).

Following Ryle, Dennett argued that explaining content by postulating contentful mental states is tantamount to postulating a homunculus who understands the content of the mental states. This, however, either begs the question of how the homunculus understands content, or leads to the infinite regress of postulating homunculi inside homunculi. Dennett argued that mental states and their content come from the external observer of a system; they are ascribed to people, not discovered in them, by describing and predicting the behavior of a system from what Dennett called the *intentional stance*. According to Dennett, people ascribe contentful states, e.g. beliefs and desires, to each other in order to predict each other's behavior, but inside people's brains there is nothing that realizes those contentful states in an exact way, like Harman's computational states were supposed to do (Dennett 1987, pp. 53, 71). If a system's behavior is sufficiently complex and adaptive and an external observer does not know its internal mechanisms well enough to derive its behavior mechanistically, then the observer interprets the system as

[38] However, it seems that Dennett's functionalism about content was more indebted to Putnam's computational functionalism than to Sellars's CRS. Dennett put it as follows:

> I had read some Sellars when I finished Content and Consciousness [Dennett 1969], but I hadn't thought I understood it very well. Several of his students had been in Oxford with me, and had enthused over his work, but in spite of their urging, I didn't "become a Sellarsian." I'd read all of Putnam's consciousness papers (to date), and was definitely influenced strongly by Putnam. One of Sellars' students, Peter Woodruff, was a colleague of mine at UC Irvine, and it was he who showed me how my work was consonant with, and no doubt somewhat inspired by, Sellars. But that was after he read C&C. I thereupon sent Sellars one of the first copies of C&C, and he wrote back enthusiastically…
> I would think that my sketchy functionalist theory of meaning was more influenced by Putnam's "Minds and Machines" paper [Putnam 1960] than anything I'd read in Sellars while at Oxford, but I can't be sure. I have sometimes discovered tell-tale underlinings in my copy of a book years later and recognized that I had been influenced by an author and utterly forgotten it (Dennett, personal correspondence).

possessing beliefs and desires about its environment—hence as being intentional—so as to explain the system's adaptive behavior as the satisfaction of appropriate desires given mostly true beliefs. In summary, an observer ascribes content to the system. So intentional content comes from the interpretation of external observers.

A few years after Dennett formulated his theory of content, Fodor's LOT hypothesis convinced Dennett that content could be ascribed to internal states of a system, and specifically to states of stored-program computers, without begging the question of how content is understood (Dennett 1978). Dennett explained why the question is not begged by applying his interpretational semantics to the internal states, in a version of his theory that was later called *homuncular functionalism* (Dennett 1975, 1978a). According to homuncular functionalism, content (and the intelligence needed to understand it) can be ascribed to the internal states of a system without begging the question of how it is understood as long as the ascribed content is ultimately discharged by a completely mechanistic explanation of the behavior of the system. Discharging content ascription mechanistically consists of decomposing the system into parts and explaining the content and intelligent behavior of the system in terms of the content and intelligent behavior of the parts. In doing this, the parts and their behavior can still be ascribed content, but understanding their content must require less intelligence than the intelligence required by the content ascribed to the whole. The same process can be repeated for the content ascribed to the parts, explaining their content and intelligence in terms of the content and intelligence of their parts, but the parts' parts' intentional descriptions must ascribe less and less content and intelligence to them. The process ends with components whose behavior is so simple and obviously mechanical that it warrants *no* content ascription. Dennett offered his homuncular functionalism as a way to cash out the intentional descriptions commonly used for computers and their programs, especially in the context of artificial intelligence research. Thus, homuncular functionalism explicates the semantic view of computational states in a

metaphysically non-mysterious way. It dissolves, rather than solve, the problem of computational content.[39]

Dennett's theory went through elaborations and revisions (Dennett 1978, 1987), but its core remained: content, whether intentional or computational, comes from the interpretations of external observers. Dennett's theory has the great advantage of being equally applicable to organisms and artifacts like computers, giving a common treatment of computational and intentional content. This is because both organisms and artifacts are equally subject to intentional interpretation by external observers. However, interpretational semantics also denies the reality of intentional content: for Dennett, intentional content is an instrument of prediction; as much as the predicted behavioral patterns are objective (Dennett 1987, pp. 15, 25), the states posited by the predictive tool do not correspond to any of the internal states posited by a correct mechanistic explanation of the system whose behavior is being predicted. Put another way, Dennett does not believe in original or intrinsic intentionality (e.g., Dennett 1987, p. 288). A similar point applies to computational content: when computational content is discharged by the decomposition of the system into the activity and internal states of purely mechanical (non-contentful) components, computational content is explained away. Furthermore, interpretation is somewhat indeterminate: in principle the same behavior by the same system can be interpreted in different and equally adequate ways (e.g., Dennett 1987, p. 40, chap. 8). Dennett's theory explains content at the cost of deeming it unreal.

A corollary of Dennett's conjunction of the semantic view of computational states and interpretational semantics is interpretationism about computation: whether something is a computing mechanism is a matter of interpretation not fact. From Dennett's theory, it follows that there is no principled difference between a desktop computer and a refrigerator, except that applying the intentional stance to the computer is more useful than applying it to a refrigerator. This leads to the paradoxical

[39] In formulating homuncular functionalism, Dennett also argued that the Church-Turing thesis entails that a mechanistic theory has to be computational, hence that "AI is the study of all possible modes of intelligence" (Dennett 1975, p. 83). I discuss this further component of Dennett's homuncular functionalism, which makes no difference to the present discussion, in Chapter 7.

effect that Dennett's intentional stance, which seems to account so well for computational content, does not account for our practice of applying the term "computer" only to some machines, which seem to belong to a special class distinct from other machines. Given Dennett's theory, in order to explain the difference between computing mechanisms and other machines, one must abandon the intentional stance and take some other stance (perhaps the design stance) towards the mechanisms. A similar problem arises in comparing different interpretations of the same computation. Given that for Dennett interpretation is partially indeterminate, there may be two equally adequate computational interpretations of a process. What they have in common, then, cannot be expressed from within the intentional stance: one needs to leave the intentional stance and resort to some other stance. It turns out that if Dennett wants to explain the difference between computing mechanisms and other mechanisms, or explain what two adequate interpretations of a computation have in common, he needs to identify computing mechanisms and their states using non-intentional language.

Dennett's theory of content was very successful among philosophers interested in CTM. For example, John Haugeland used a version of Dennett's homuncular functionalism as an explication of the research program, pursued by many in psychology and AI, of developing a CTM (Haugeland 1978, 1985, 1997). For Haugeland, like for Dennett, the content ascribed by a computational theory of a system, including a CTM, comes from the theorist's interpretation of the system.

The author who elaborated Dennett's theory of content in the most sophisticated and systematic way is perhaps Robert Cummins. Cummins built a theory of computational explanation (1983), which includes a theory of computational content (1983, 1989), drawing from both Dennett and Haugeland's writings. Like Dennett and Haugeland's theories, Cummins's theory is squarely based on the semantic view of computational states and explains content in terms of how an external observer interprets a system.

Unlike Dennett, who was frankly anti-realist about intentional content, Cummins takes a more realist position (1983, pp. 74-75). Cummins sharply distinguishes between intentional content and computational content. He argues that in formulating a CTM, psychologists and AI researchers postulate

(contentful) computational states as explanations for people's cognitive capacities, and offers his interpretational theory as an account of the content of computational states postulated by CTM theorists. But Cummins also argues that CTM falls short of explaining genuine intentional content, or as he put it, the intentionality of mental capacities. He discusses five strategies to explain intentional content in terms of computational content and argues that they all fail (1983, pp. 91-100). Cummins still states that intentionality is somehow going to be accounted for in terms of computation, but he adds that he has no idea how this can be done (1983, pp. 89-90). He denies that he has a theory of intentional content (1989). So, Cummins's theory of content does nothing to solve the philosophical problem of intentional content with which we are presently concerned (nor is it intended to do so). Moreover, Cummins's theory entails that the same system can be interpreted in different ways that ascribe different computations.

Another author in the interpretationist tradition is Patricia Churchland. In her book written with neuroscientist Terrence Sejnowski, she offers a detailed account of the brain as a computing mechanism, predicated on the semantic view of computational states. In explicating what it means to be a computer, Churchland and Sejnowski state an informal version of Cummins's theory (with a reference to a paper by Cummins for more details: Churchland and Sejnowski 1992, p. 65):

> We count something as a computer because, and only when, its inputs and outputs can usefully and systematically be interpreted as representing the ordered pairs of some function that interests us (Churchland and Sejnowski 1992, p. 65).

Unlike Cummins, Churchland and Sejnowski did not discuss explicitly what notion of content is at stake in their theory. Perhaps because of this, some authors have accused Churchland and Sejnowski of being ambiguous about whether or not they ascribe genuine intentional content to brains (Grush 2001). In light of the preceding discussion of interpretational semantics, I offer a more charitable reading. Churchland and Sejnowski were working with an interpretational semantics, which has the same advantages and disadvantages of all interpretational semantics. On the one hand, it applies equally well to organisms and artifacts. On the other hand, it does not solve the problem of intentional content. Churchland and Sejnowski may choose whether to side with Dennett or Cummins: either they accept Dennett's anti-realism about intentional content, so that their interpretational semantics explains intentional content

away (with Dennett), or they endorse Cummins's realism about intentional content, but then they must defer to some other theory as far as intentional content is concerned.

In conclusion, interpretational semantics is a very natural and attractive way to cash out the semantic view of computational states, but it comes at the cost of losing the ability to explain how computing mechanisms differ from other mechanisms, and what two distinct but equally adequate computational interpretations of a process have in common. In order to regain these abilities, an interpretational semanticist needs a non-semantic way to identify computational states. In addition, interpretational semantics makes computational descriptions, even construed as semantic or intentional descriptions, insufficient to characterize intentional content as a real property of minds. This may not trouble those who don't believe in intentional content to begin with, but it leads others to look elsewhere for a theory of content.

9.5.3 CTM meets Informational and Teleological Semantics

Until the mid-1970s, Fodor freely appealed to the semantic view of computational states in formulating his version of CTM, without discussing the need for a theory of content. In a series of papers in the late 1970s (collected in Fodor 1981), he became more explicit about the relationship between CTM and intentional content. He argued that folk psychology formulates its generalizations by quantifying over the *semantic* properties of propositional attitudes, whereas cognitive (computational) psychology formulates its generalizations by quantifying over the *syntactic* properties of mental representations. He added that the strength of a computational psychology, postulating LOT, was that computational psychology had the resources to reduce the generalizations of folk psychology to scientific generalizations, so as to underwrite our intuitive discourse about contentful propositional attitudes. This is because computations, albeit being causally driven by the syntactic properties of representations, are processes that (can) respect the semantic properties of representations. So, given a computational psychology, the two stories about the mind, cognitive and folk, would eventually match, in the sense that under the appropriate ascription of content to the states posited by cognitive psychology, the relations between semantically identified folk

psychological states would match the causal relationships between cognitive psychological states. What was needed to complete this picture was a theory ascribing the right content to the computational states: a theory of content (Fodor 1981; see also Fodor 1987, chap. 1).

A theory purporting to do this was CRS, but Fodor rejected it. He wrote a critique of procedural semantics, a theory of content that was popular in AI and psychology. According to procedural semantics, the content of a computational instruction is given by the computational procedures that execute that instruction. Since the relations between an instruction and the computational procedures that operate on it are functional relations, procedural semantics is a version of CRS. Fodor argued that, since the procedures that execute computer instructions are entirely internal to the machine and, when transformed into machine language, are naturally interpreted as referring to the shifting of bits from one register of the machine to another, procedural semantics reduced the content of computational descriptions to content about shifting bits from one register to another, without ever involving anything external to the machine (Fodor 1978).[40] Besides procedural semantics, Fodor also rejected CRS in general, in part because he saw that, given a semantic view of computational states, CRS was circular (e.g., see Fodor 1990, chapt. 1). Fodor maintained the semantic view of computational states and treated content ascription as prior to computation ascription. For him, computations are defined over representations, which are identified by their content, so computations are identified by the semantic properties of the representations over which they are defined. Because of this—which is the semantic view of computational states—a theory of content cannot identify contents by appealing to the notion of computation, on pain of circularity.[41]

By the end of the 1970s, any computationalist philosopher of mind who—like Fodor—took intentional content as a real property of minds but rejected CRS, needed an alternative (naturalistic)

[40] In my terminology, Fodor's point is that the internal semantics of a computer is insufficient to generate an external semantics.

[41] The most explicit discussion of this point that I know of is in Fodor 1998. He writes that since his notion of computation presupposes the notion of content, he can't account for content in terms of computation (ib., esp. p. 13). He also says explicitly that because of this, his theory of content (unlike his theory of thought) is not computational (ib., p. 11).

theory of content. This demand was soon met by two new approaches, Informational Semantics and Teleological Semantics (ITS). According to ITS, the content of a mental state comes from natural relations between that state and the mind's environment. Informational semantics says that the crucial relations for identifying content are informational, namely relations determining what information is carried by a state (Dretske 1981, 1986). Teleological semantics says that the crucial relations involve the evolutionary history of the state, namely what the mechanism generating that state was selected for (Millikan 1984, 1993). The main difference between CRS and ITS is that while the former is holistic and (partially) internalist, specifying (part of) the content of a state by its relation to other contentful states, ITS is externalist and atomistic, specifying the content of a state independently of its relation to other contentful states. So, ITS specifies the content of a state independently of what computations it enters. Following Dretske and Millikan, Fodor developed his own version of ITS (1987, 1990, 1998) with the explicit goal of finding a theory of content for the representations postulated by LOT.

According to Fodor, the combination of ITS and CTM offers our best hope for a scientific theory of mind that will respect folk intuitions about intentional content, so that the stories told by cognitive and folk psychology will match. This is because CTM accounts for how mental processes can be causally efficacious while respecting the semantic properties of mental representations, and ITS accounts for how representations get their content (Fodor 1987, 1990, 1998).

Given a theory of content of the ITS form, it should be obvious that being a computational state is insufficient for having intentional content. I can run an instantiation of the argument schema described at the beginning of this section:

> Premise 1. Having intentional content is the same as entering certain informational or teleological relations with the environment.
>
> Premise 2. Being a computational state does not entail entering the relations mentioned in premise 1.
>
> Conclusion: Being a computational state does not entail having intentional content.

Premise 1 is just ITS. Premise 2 expresses the familiar fact that what we ordinarily call computers, and use as computers, are rarely if ever hooked up to the environment in the complicated ways postulated by ITS. Some computing mechanisms, which computer scientists call *embedded* systems (e.g., cars' computers or digital thermostats), *are* connected to the environment in ways that resemble those postulated by some versions of ITS, but they are not the typical case. Ordinarily, whether something is a computing mechanism is independent of the ITS relations it bears to the environment. This conclusion does not come as a surprise to ITS theorists: ITS theorists generally don't ascribe intentional content to ordinary computers.[42] This has the important consequence that, to the extent that they accept ordinary computation ascription, ITS theorists who believe in CTM are committed to there being something in common between computing mechanisms that satisfy the demands of ITS for intentional content and ordinary (non-embedded) computing mechanisms: although they are all computing mechanisms, some have intentional content while others don't. That is, a consequence of conjoining CTM and ITS is that minds and ordinary computing mechanisms have something in common that cannot be specified by ITS. The way to specify it, I submit, is by a functional account of computational states.

To summarize, given ITS the search for a theory of intentional content is not by itself a motivation to endorse CTM, because the solution to the problem of intentional content is independent of CTM. If anything, it is the search for a theory of contentful mental states that still motivates computationalist philosophers who want to save the semantic view of computational states to match CTM with ITS. If they succeed, they find themselves in a position from which they cannot tell what minds have in common with ordinary computing mechanisms. In order to tell, they need a non-semantic way to identify computing mechanisms and their states.

[42] For a critical discussion of this feature of Dretske and Fodor's view, see Dennett 1987, chapt. 8.

9.5.4 CTM meets Intentional Eliminativism

All the computationalist philosophers discussed until now shared the semantic view of computational states. The first who questioned their assumption was Stephen Stich (1983). Like other computationalist philosophers, Stich's primary goal was not to give a philosophical account of computation but a theory of mind; his rejection of the semantic view of computational states was an implicit consequence of his theory of mind. Stich was motivated by the belief that folk psychology, including the intentional contents it postulates, will be eliminated in favor of a cognitive psychological theory of mind. *Contra* Fodor, he argued that the generalizations of cognitive psychology would not match those of folk psychology (Stich 1983, chaps. 4, 7, and 9).

Stich formulated a version of CTM that does not require mental states to have content, a theory that he called syntactic theory of mind. In order to have a CTM without intentional content, Stich implicitly rejected the semantic view of computational states. According to him, the mind is computational, but computational descriptions are not semantic and *a fortiori* not intentional. For something to be computational, its physical states must be mappable onto syntactically defined states, without presupposing any semantics. A system is a computing mechanism if and only if there is a mapping between its behaviorally relevant physical states and a class of syntactic types, specified by a grammar that says how complex types can be formed out of primitive types. According to Stich, the mind is a computing mechanism in this sense (Stich 1983).

By formulating his version of CTM in terms of his syntactic view of computational states and doing away with intentional content, Stich renounced what Fodor considered the crucial consideration in favor of CTM: the hope that cognitive psychological generalizations ranging over syntactically identified states would correspond to folk psychological generalizations ranging over contentful propositional attitudes. According to Fodor, the point of CTM was to explain how a mere mechanism could mirror semantic relations by invoking the match between mechanical processes that respond only to syntactic properties (computations) and processes identified by semantic properties (inferences). From Fodor's point of view, if one follows Stich in denying that mental states have content, it is unclear why and how

mental states should be construed syntactically. Perhaps because of this, Stich's syntactic theory of mind has won few converts. But Stich argued that a syntactic theory of mind offered the best explanation of mental phenomena: for instance, Stich said that beliefs in the folk sense would likely be identified with some non-contentful, syntactically identified states that have a functional role similar to the one played by beliefs in folk psychology (Stich 1983, chapt. 11). Moreover, Stich argued that his syntactic theory of mind was the best construal of the computational theories of mental phenomena offered by cognitive psychologists.

If Stich's proposal of a syntactic theory of mind is coherent, it shows how CTM can be formulated without the semantic view of computational states. This point is independent of Stich's intentional eliminativism. There is an important sense in which Stich's syntactic theory of mind is compatible with the existence of intentional content and the Fodorian argument for CTM. Stich's syntactic criterion for mental states can be taken as a way to identify mental states and processes as computational in a non-semantic way, while leaving open the question of whether they also have content and whether there are generalizations ranging over contentful states that match those formulated over syntactically identified states.[43]

This is an open possibility so long as we abandon the semantic view of computational states, which maintains that computational states are identified by their content. For if computational states are identified by their content, it would be impossible to identify them non-semantically, as Stich's theory requires, and then ask whether they have content and what content they have. From the point of view of the semantic view of computational states, Stich's coupling of CTM and intentional eliminativism, according to which mental states are computational but contentless, is incoherent. And from the point of view of Stich's combination of CTM and intentional eliminativism, the semantic view of computational states, and any version of CTM that is formulated using the semantic view of computational states, begs the question of whether the computational mind has content.

[43] In fact, Egan has advocated the conjunction of a non-semantically formulated CTM, *a la* Stich, with an externalist theory of mental content (Egan 1992, 1995, 1999).

Indeed, Stich's notion of syntax has been challenged on the grounds that it makes no sense to speak of syntax without semantics. According to this line of thought, something can be a token of a syntactic type only relative to a language in which that token has content (e.g., Crane 1990, Jacquette 1991).[44] If this is right, then Stich's proposal is incoherent, but I don't think it is.

The coherence of Stich's proposal is easy to see when we reflect on the functional properties of stored-program computers. Some special mechanisms, namely stored-program computers, have the ability to respond to (non-semantically identified) strings of tokens stored in their memory by executing sequences of primitive operations, which in turn generate new strings of tokens that get stored in memory. Different bits and pieces of these strings of tokens have different effects on the machine. Because of this, the strings of tokens can be analyzed into sub-strings. An accurate description of how tokens can be compounded into sub-strings, and sub-strings can be compounded into strings, which does not presuppose that the strings of tokens have any content, may be called the syntax of the programming language of the computer, and indeed that is what computer scientists call it. Furthermore, because of how computers are designed, the global effect of a string of tokens on the machine can be reduced to the effects of its sub-strings on the machine. Then, the effect of sub-strings on the computer can be assigned to them as their content, and the way in which the content of the whole string depends on the content of its sub-strings can be specified by recursive clauses, with the result that the global effect of a string on the computer is assigned to it as its content. This is what the internal semantics of a computer, which I discussed in section 9.2 above, amounts to. Given that the strings manipulated by the computer have a syntax and an internal semantics, they may be called a language, and indeed that is what computer scientists call them. As I argued in section 9.2, none of this entails that the language of the computer has any external semantics, i.e. any content in the sense used by Stich's critics. Stich may be construed as arguing that the functional organization of the mind is similar to that of a stored-program computer, so that the mind contains a system of strings of tokens with a syntax analogous to that of the strings manipulated by

[44] Dietrich 1989 has a slightly different line of attack, according to which one cannot ascribe computations to a system without ascribing content to it. His is the argument from the identity of computed functions, which I already rejected in section 9.3 above.

stored-program computers. Stich would probably have no difficulty in accepting that mental strings, so construed, also have an internal semantics.

The above account of syntax is functional, specified in terms of the components of a stored-program computer, their states, and their interactions. From the vantage point of this functional view of computational states, we not only see the coherence of Stich's proposal, but we can give a functional account of his notion of syntax without presupposing any external semantics.

Stich's proposal shows that one can be a computationalist without having a theory of content and while rejecting the semantic view of computational states, because one can be a computationalist without believing in intentional content at all. A computationalist who wishes not to beg the question against the intentional eliminativist should formulate CTM without the semantic view of computational states, and independently of any theory of content.

9.6 CTM With or Without Semantics

The first moral of this chapter is that CTM was originally conceived in tandem with the semantic view of computational states, and this convinced a lot of philosophers that CTM necessarily ascribed content to the mind. Between the 1940s and the late 1960s, both in science and philosophy, computationalists promoted CTM not only as offering a mechanistic explanation of the mind but also, by construing computational description semantically or even intentionally, as offering the beginning of a naturalistic account of content.

In the 1970s, it became clear that CTM *per se* offered no solution to the problem of intentional content. The ensuing investigations of a theory of content revealed four main ways to combine CTM with a theory of content. The first combines CTM with Conceptual Role Semantics (CRS), which sees intentional content as (partially) reducible to the computational relations among mental states. This option cannot construe computational relations semantically on pain of circularity, and hence it presupposes a non-semantic way of identifying computational states. The second combines CTM with

interpretational semantics, which sees intentional content as a tool for predicting behavior. This option maintains the semantic view of computational states at the cost of denying the reality of intentional content. The third combines CTM with Informational or Teleological Semantics (ITS), which sees content as reducible to a combination of causal and counterfactual relations between intentional states and the environment. This option maintains the semantic view of computational states at the cost of being inapplicable to ordinary computing mechanisms, because most ordinary computing mechanisms don't enter the causal and counterfactual relations postulated by ITS. The fourth combines CTM with intentional eliminativism, which abandons the semantic view of computational states in favor of a non-semantic construal of computation. These options seem to exhaust the possibilities that are open to the computationalist: either intentional content comes from the computations themselves (CRS), or it comes from some non-computational natural properties of the content-bearing states (ITS), or it is in the eye of the beholder (interpretational semantics), or there is no intentional content (eliminativism). Under any of these options, either the problem of intentional content is solved by something other than computation ascription, or computation ascription must be construed non-semantically, or intentional content is unreal. Usually more than one of the above is true of each option.

For each of the above theories of content, neither the theory of content entails CTM nor CTM entails the theory of content. None of the existing theories of intentional content offers a reason to endorse CTM. Whether the mind is computational and whether the mind has content (and how it manages to have content) are different problems that need to be solved independently of each other. The semantic view of computational states also begs the question of the CRS and intentional eliminativist theorists, who need a non-semantic identification of computational states in order to formulate their views. In order to keep the two problems separate, we should avoid formulating CTM as a theory that ascribes content to the mind, as it is often done (e.g., by Fodor 1998, Rey 1997). Even those who are skeptical about full-blown intentional content but believe in some form of computational content (e.g., Churchland and Sejnowski 1992) should avoid formulating CTM as a theory that ascribes content to the mind. CTM should be formulated in a way that is neutral about content, leaving it to considerations about

content to determine which theory of content is correct. CTM is a theory of the internal mechanisms of the mind or brain, which may or may not explain some mental phenomena. So, the semantic view of computational states should be abandoned in favor of the functional view of computational states, and CTM should be formulated without using intentional or semantic language. Fortunately, as I argued on independent grounds in section 9.2 and as I argue more extensively in Chapter 10, this is also the best way to understand computing mechanisms in their own right. Stich is right in one important respect: there is no need to invoke content to understand computing mechanisms and how they work; it's actually misleading to do so.

Construing CTM without the semantic view of computational states leaves the field entirely open for different positions about content. Perhaps some computational states have content and some don't; perhaps all do or none do. Perhaps some have content in one sense and not in others. CTM should not be seen as answering any of these questions. If the mind is computational but has no content, then CTM will explain the mind without requiring a theory of content. If the mind does have content, this is going to be explained by a theory of content. If mental states are both contentful and computational, the true version of CTM and the true theory of content will be compatible with each other. One example of compatibility is offered by combining a non-semantically formulated version of LOT with ITS; this is analogous to Fodor's view minus his semantic view of computational states. Another example is the combination of a non-semantically formulated version of CTM and interpretational semantics; this is analogous to Dennett's view minus his semantic view of computational states. A third example is the conjunction of a non-semantically formulated version of CTM with CRS.

9.7 Two Consequences

If the question of content is independent of the question of computation, there are some consequences that deserve to be explored. I will briefly mention a two:

1. During the last two decades, it has become common to hear criticisms to CTM based on the rejection of representationalism (Brooks 1997, Thelen and Smith 1994, van Gelder 1995, and certain passages in Clark 1997). According to these criticisms, some or all mental phenomena can be explained without postulating contentful mental states, and therefore CTM should be rejected. As I've tried to show, many computationalists endorse the semantic view of computational states, and therefore their position is vulnerable to this criticism. But I also argued that the semantic view of computational states should be rejected. If this is done, the anti-representationalist critique of CTM turns out to be confused in the same way that the semantic view of computational states is. Even if we don't need representations to explain cognition (which I doubt), this would do nothing to undermine CTM *per se*, but only the combination of CTM with representationalism. CTM can and should be formulated independently of any theory of content, which makes it invulnerable to anti-representationalist critiques.

2. Above, I mentioned Fodor's argument according to which CTM is our best theory of mind because, by postulating causal processes that can mirror semantic relations between representations, it offers the hope to generate a scientific theory of mind close to our folk theory of mind. (Fodor has forcefully made this argument in conjunction with ITS, but a version of it could be run in conjunction with CRS.) However, if we accept that the question of computation is independent of the question of content, it becomes clear that this argument is missing a crucial premise. Before we accept that CTM has the potential to match contentful relations by syntactically defined processes, we should ask by what mechanism this match is achieved. In other words, we need to conjoin CTM not only with a theory of content, but also with a theory of how the computational relations get to mirror the semantic properties of the internal states. Notice that the mechanism that accomplishes the mirroring cannot be computational on pain of circularity. For if it were a computational mechanism, we should ask whether *its* computational processes match *its* semantic properties. If they don't, then it is unclear how such a mechanism could achieve the syntax-semantic match in the first mechanism. If they do, we need to answer the question of how they do, and we are back where we started. So the matching must either be done by a non-computational mechanism, or be innate. In the first case, one needs to explain how such a

non-computational mechanism works; in the second case, one needs to explain how the innate matching is achieved.[45] Fodor has come close to admit that he doesn't know how to solve this problem, and this is one of the reasons for his skepticism that CTM is going to offer a complete explanation for the mind (see Fodor 2000, esp. pp. 71-78).

[45] I think here there is tension between Fodor's nativism and his rejection of teleological semantics, because innate mechanisms do not enter the causal and counterfactual relations that assign content to mental states according to Fodor's ITS (1990, 1998). But then, how do innate mechanisms match their semantics with their syntax?

10 COMPUTING MECHANISMS

10.1 Introduction

This chapter offers an account of what it is for a physical system to be a computing mechanism, namely a mechanism that performs computations. This account, which I call the *functional account of computing mechanisms*, can be used to identify computing mechanisms and the functions they compute. It can also be used to taxonomize computing mechanisms based on their different computing power. The present account stems from two main moves.

First, I use non-semantic and *a fortiori* non-intentional language to identify computing mechanisms and the functions they compute; i.e., I keep the question of whether something is a computing mechanism and what it computes separate from the question of whether something has semantic content. I defended this move at length in Chapter 9, where I showed that this move rules out most of the existing accounts of computing mechanisms in the philosophical literature.

Second, I explicate the notion of computing mechanism using the tool of functional analysis. I construe functional analysis as in engineering and biology, where a functional analysis of a mechanism is a partition of a mechanism into component parts and an assignment of proper functions to those parts. Given this construal, analyzing a mechanism functionally is different from (though compatible with) giving a computational description of that mechanism. I defended this move in Chapter 8. Again, this move rules out almost all existing accounts of computing mechanisms in the relevant philosophical literature, where giving a functional analysis is construed as being the same as giving a computational description.[1]

[1] For more on this notion of functional analysis, see Bechtel and Richardson 1993. For a sophisticated version of the same notion, with emphasis on neural mechanisms, see Craver 2001. For an early application to psychological and neural mechanisms, see Deutsch 1960. The classic sources of the notion of functional analysis in philosophy of psychology, Fodor 1968 and Cummins 1975, 1983, have a lot in common with the notion used here but are potentially misleading for present purposes because, as I argued in Chapter 8, those authors conflate giving a functional analysis of a mechanism with giving a computational description of that mechanism.

The functional account of computing mechanisms flows naturally from these two moves. Calculators and computers are analyzed in terms of their component parts (processor, memory, input and output devices) and their functions. Those components are also analyzed in terms of their component parts (e.g., registers and circuits) and their functions. Those, in turn, are analyzed in terms of primitive computing components (logic gates) and their functions. Primitive computing components can be further analyzed but their analysis does not illuminate the notion of computing mechanism. In contrast to extant philosophical accounts of computing mechanisms, the present account explicates very naturally both our ordinary language about computing mechanisms and the language and practices of computer scientists and engineers.

Although I believe I have motivated the above moves sufficiently well in the relevant chapters, in this chapter I'm going defend the functional account of computing mechanisms on grounds that are as independent of those moves as possible. I will do so by first stating six desiderata that an account of computing mechanisms should satisfy, and then comparing and contrasting the main existing accounts of computing mechanisms *vis a vis* those desiderata. I will point out the difficulties of existing accounts in satisfying those desiderata, and the advantages of the functional account of computing mechanisms, as independently as possible of the above moves.

10.1.1 Desiderata for an Account of Computing Mechanisms

An optimal account of computing mechanisms should satisfy the following six desiderata:

1. *Paradigmatic computing mechanisms compute*. A good account of computing mechanisms should entail that all paradigmatic examples of computing mechanisms, such as digital computers, calculators, both universal and non-universal Turing Machines, finite state automata, and humans who perform mathematical calculations, compute.

2. *Paradigmatic non-computing mechanisms don't compute*. A good account of computing mechanisms should entail that all paradigmatic examples of non-computing mechanisms and systems, such as planetary systems, hurricanes, and stomachs, don't compute. As I showed in Chapter 8, in the

literature computers and Turing Machines are usually taken to be paradigmatic examples of computing mechanisms, whereas planetary systems, the weather, and stomachs are taken to be paradigmatic examples of mechanisms that don't perform computations.

When we have pre-theoretical intuitions about clear cases of things that have a property and things that don't, as in the case of computing mechanisms, a general requirement of a good philosophical theory is that it fit those clear cases. By satisfying desiderata 1 and 2, a good account of computing mechanisms would draw a principled distinction between the class of computing mechanisms and the class of other mechanisms, and it would draw this line in a place that fits our pre-theoretical intuitions as much as possible.

3. *Computation is observer-independent*. A good account of computing mechanisms should entail that the computations performed by a mechanism can be identified independently of which observer is studying the mechanism. This is a desideratum for two reasons. First, it underwrites ordinary computer science and computer design, where different observers agree on which computations are performed by which mechanisms. Second, it underwrites a genuinely scientific computational theory of the brain, according to which the computations performed by the brain can be empirically discovered.[2]

4. *Computations can go wrong*. A mechanism m miscomputes just in case m is computing function f on input i, $f(i) = o_1$, m outputs o_2, and $o_2 \neq o_1$.[3] A good account of computing mechanisms should explain what it means for a computing mechanism to miscompute. This is a desideratum because making "mistakes" is an important aspect of the intuitive notion of computing as well as the practice of using machines to perform computations. Those who design and use computing mechanisms devote a

[2] This desideratum is one instance of the general desideratum that scientific concepts and methods be inter-subjective, or public. For an extended clarification and defense of this desideratum for scientific methods, see Piccinini forthcoming.

[3] Here o_1 and o_2 represent any possible outcome of a computation, including the possibility that the function is undefined for a given input, which corresponds to a non-halting computation. Miscomputation is analogous to misrepresentation (Dretske 1986), but it's not the same. Miscomputation is possible whether or not the inputs and outputs of a mechanism represent anything. Misrepresentation is possible whether or not the representations involved are computational inputs, outputs, or internal states.

large portion of their efforts to avoid miscomputations. To the extent that an account of computing mechanisms makes no sense of that effort, it is unsatisfactory.

5. *Some computing mechanisms are not computers*. A good account of computing mechanisms should explain how the machines that are ordinarily called computers, such as our desktops and laptops, are different from other computing mechanisms, such as calculators and finite state automata. This is a desideratum because the term "computer" in ordinary language is not used for all computing mechanisms but is reserved for some special ones. A good account of computing mechanisms should explain why the term is used in this restrictive way.

6. *Program execution is explanatory*. Ordinary digital computers are said to execute programs, and their behavior is normally explained by appealing to the programs they execute. In Chapter 8, I showed that the computationalist literature contains appeals to explanation by program execution (and more generally explanation by appeal to the computations performed by a mechanism) as the appropriate form of explanation for psychological capacities. So a good account of computing mechanisms should say how appeals to program execution (and more generally to computation) explain the behavior of a computing mechanism. It should also say how program execution relates to the general notion of computation: whether they are the same or whether program execution is a species of computation, and if so, what is distinctive about it.

With these desiderata as landmarks, we can proceed to formulate the functional account of computing mechanisms.

10.2 The Functional Account of Computing Mechanisms

The central idea in my proposal is to analyze computing mechanisms using functional analysis. Computing mechanisms are understood as mechanisms whose proper function (in the sense introduced in section 8.3) is computing. It then becomes evident that this use of functional analysis applies to what are

called computers in ordinary language in a way that matches the language and practices of computer designers.[4]

Systems that are characterized by functional analysis in the present sense can malfunction, break, or even be malformed or defective.[5] Mechanisms, including computing mechanisms and their components, perform the activities we ascribe them *ceteris paribus*, as a matter of their proper function. In the rest of our discussion, we will mostly focus on their normal operation, but it is important to keep in mind that they can malfunction, break, or be malformed or defective. This will help satisfy desideratum 4 (Computations can go wrong).

Functional analysis of a mechanism can proceed either top-down—starting from the whole mechanism and its activities and discovering its components and their activities—or bottom-up—starting from the components and their activities and discovering how they are put together and how they contribute to what the whole mechanism does. In the research strategies of scientists who study biological mechanisms, functional analysis proceeds in both directions at once; that is, scientists work on both the whole mechanism and its components at the same time (Craver and Darden 2001).

In the present case, we are not analyzing any particular mechanism but offering an account of what is peculiar to a class of mechanisms, namely computing mechanisms. Since there are different kinds of computing mechanisms, which have properties in common but also different degrees of functional complexity and computation power, it would be long and repetitive to analyze each kind of computing mechanism separately from the others. The best strategy is not to analyze each kind of computing mechanism separately, but to start with the component types that are common to many computing mechanisms, and then to show how those components are put together in various ways to form different types of computing mechanisms. In this sense, my exposition will proceed from the bottom up, but this should not suggest that in studying a putative computing mechanism, such as the brain, this needs to be the preferred strategy in the context of discovering how it works and what it computes.

[4] For a standard introduction to computer organization and design, from which I took some of the technical terminology and details, see Patterson and Hennessy 1998.
[5] These aspects have been especially emphasized by Millikan (1984, 1993).

Computing means generating certain output strings of tokens (or symbols) from certain input strings of tokens (and perhaps internal states), according to a general rule that applies to all inputs and outputs. A token is a particular that belongs to a (usually finite) number of types. Upon entering a mechanism in the relevant way (explained below), a token affects the input-output behavior of the mechanism. Every token of the same type affects a mechanism in a way that has the same effects relative to generating output, and each type of token has a different effect on the mechanism relative to generating output, in the sense that ceteris paribus, substituting a token of type T_2 for a token of type T_1 in a string (where $T_2 \neq T_1$) results in a different computational process, which may generate a different output string.

A string is a list of permutable tokens identified by the tokens' types, their number, and their order within the string. Every finite string has a first and a last token member and each token member (except for the last member) has a unique successor. A token within a string can be substituted by another token without affecting the other tokens' types, number, or positions within the string. In particular, when an input string is processed by a mechanism, the tokens' types, their number, and their order within the string make a difference to what output string must be generated.

I'll start my analysis of computing mechanisms with the simplest components that can be attributed a computing function within a mechanism, i.e. components that can be attributed a computing function but whose components cannot be attributed a computing function. I call them primitive computing components. Then I'll discuss how primitive computing components can be put together to form more complex computing components, such as Boolean circuits, arithmetic logic units, etc. Finally, I'll discuss how complex components can be put together to form more complex computing mechanisms, such as calculators and various types of computers properly so called.

10.2.1 Primitive Computing Components

Primitive computing components are mechanisms that perform computations contributing to what their containing mechanism computes, but whose components do not perform any computation relative to what they compute. In other words, the computations performed by primitive computing components cannot

be analyzed in terms of simpler computations. So their computations are primitive relative to the mechanism under analysis.

Primitive computing components belong to the larger class of input-output devices, i.e. devices whose proper function is to receive physical tokens as inputs and yield physical tokens as outputs in response to their inputs. But unlike most input-output devices, the inputs and outputs of primitive computing components are not identified by their physical properties (e.g. chemical composition, temperature, pressure, etc.) but only by their functional relevance to the behavior of the mechanism, to the effect that the mechanism generates certain types of outputs given certain types of inputs. Because of this, inputs and outputs of primitive computing components can be labeled by letters from an alphabet, or symbols, without losing information about the functional properties of the components. At any given time, primitive computing components take a finite number of tokens as inputs and yield a finite number of tokens as output. Typically the output of primitive computing components is a deterministic function of the inputs and internal states of the component, but it can also be a stochastic function of them.

The primitive computing components of most modern computing mechanisms are logic gates.[6] Logic gates are devices with two or three extremities. During any relevant time interval, a logic gates receive one or two input tokens through one or two of their extremities and produce one output token through another extremity. Logic gates' input and output tokens belong to one of two types, usually called "1" and "0." Tokens of these types are normally called *bits*. In analyzing and designing logic gates, 1 and 0 are respectively interpreted as truth and falsehood, so that functional operations on them can be interpreted as logic functions applied to truth values. Under the standard interpretation of logic gates' tokens, the type of a logic gate's output token corresponds to what would be generated by applying a logical connective, such as conjunction, disjunction, or negation, to the types of the input token(s). This

[6] Logic gates should not be confused with logical connectives. Logical connectives are abstract, whereas logic gates are concrete objects.

is why these devices are called *logic* gates. Accordingly, the input-output behavior of logic gates can be represented by truth tables, or equivalently by logic equations written in Boolean algebra.[7]

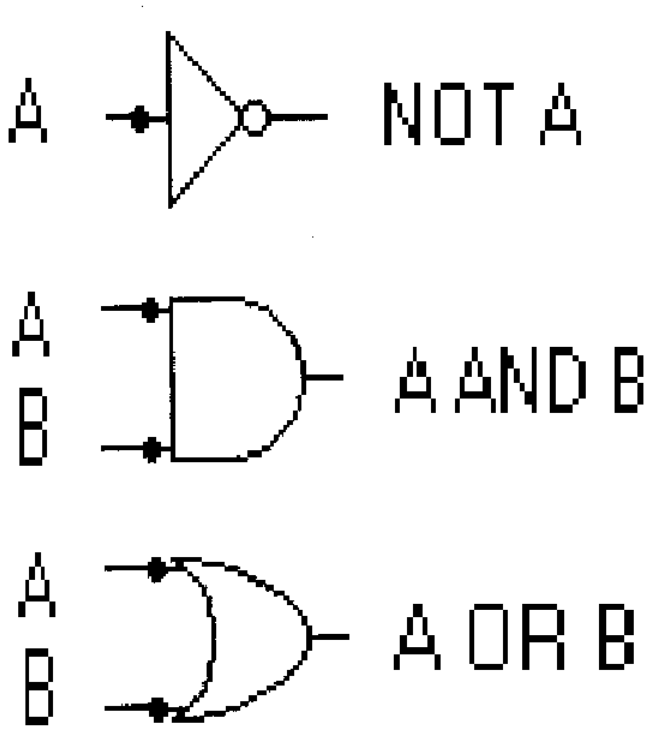

Figure 10-1. A NOT gate, an AND gate, and an OR gate.

Figure 10-1 contains three examples of logic gates. A NOT gate takes a token (i.e., either a 0 or 1) as input and yields a token that belongs to the other type (i.e., a 1 or a 0 respectively) as output. An AND gate takes two tokens as input and yields a 1 if and only if two 1's were received as input, a 0 otherwise. An OR gate also takes two tokens as input and yields a 1 as output if and only if it received at least one 1 as input, a 0 otherwise.

Not all computing mechanisms are constructed out of logic gates. For example, so-called analog computers (see below) contain physical devices that manipulate their inputs in ways that are naturally

[7] From now on, this is what is meant by the expression "logic equation."

interpreted as arithmetic operations (e.g., addition or multiplication) or integration. When they receive certain input tokens, they generate output tokens whose relevant physical properties stand in relation to the same property of the inputs in a way that corresponds to a certain arithmetical function of the input. These devices are primitive computing components too. Like in the case of logic gates, the input-output behaviors of primitive computing components of analog computers can be composed together to execute the steps of an algorithm that takes their operations as primitive, so that their combined work produces the computation of their containing mechanism. Because of this, they can be said to compute the input-output functions they instantiate.

Any physical realizion of a primitive computing component is a particular mechanism made out of physical components, which have particular physical properties and interact in particular ways to generate the desired input-output behavior. For instance, primitive computing components of most modern computing mechanisms are made out of electrical circuits, which are made out of wires, switches, resistors, and other components studied by electrical circuit theory. The same component, e.g. an AND gate, can be made using different electrical circuits so long as they exhibit the relevant input-output properties. And the same electrical circuit can be made by using different physical materials.

The functional analysis of a physical realizion of a primitive computing component explains how that component exhibits its specific input-output behavior. In our example, the configuration of a particular electrical circuit (as detailed by its functional analysis in terms of electrical components and their interconnections), in combination with the principles of electrical circuit theory, explains how it realizes an AND gate. This is still a functional explanation, because it explains the behavior of the whole mechanism in terms of the activities of its components. But it is not a computational explanation, because its components are identified in terms of their physical properties (e.g., voltage levels) and not in terms of which tokens they manipulate and how. Each physical realizion of a primitive computing component that uses different physical components or a different design requires a specific functional explanation that appeals to that specific design or those specific physical properties.

What primitive computing components are made of and how they work makes a big difference to those who have to physically implement a given computing mechanism. Logic gates (as well as other components) that are used in building a particular computing mechanism must be technologically compatible with each other. In implementing a design for a complex computing mechanism made out of logic gates, it makes no sense to use a certain device as a logic gate unless there are other devices (other putative logic gates) that it can be physically connected with. If one takes two devices that act as logic gates within two different machines and hooks them up together, one doesn't reconstitute a computing mechanism unless the two devices physically interact in the right way. This points again to the fact that in building a computing mechanism one has to take into account a lot of details about the physics of the device. But all the physical stuff happens "below" the level of primitive computing components. And everything below the level of primitive computing components, such as the functional explanation of specific physical realizions of primitive computing components, is irrelevant to the functional analysis of computing mechanisms as such.

Once one reaches the level of primitive computing components, one can ignore all physical issues as "implementational details" and worry only about the input-output properties of primitive computing components. This is what computer designers do. The behavior of primitive computing components is relevant to the functional analysis of computing mechanisms only in so far as they generate certain outputs in response to certain inputs. These input-output behaviors, together with the way primitive computing components are interconnected, can then be used to explain the computations performed by larger components. Aside from their input-output behaviors, in the functional analysis of computing mechanisms (as well as in computer design), primitive computing components can be treated as black boxes. Since the functional explanation of primitive computing components does not appeal to computations by their components, while at the same time primitive computing components' input-output behavior can be used as primitive computational operations to explain the computations performed by larger components, primitive computing components are the simplest components to be ascribed computations within a mechanism, and for that reason they are called *primitive*.

So the sense in which logic gates compute derives from the fact that they can be composed to form more complex mechanisms that compute more complex functions, and the complex functions computed by the complex mechanisms can be deduced from those computed by their components together with the way the components are connected and the use of Boolean algebra or truth tables. This makes logic gates extremely useful and versatile in computer design, and accounts for the fact that they are usually ascribed computations even though what they do is computationally trivial. They compute because their input-output behavior is a component of the input-output behavior of their containing mechanism, and the input-output behavior of their containing mechanism can be analyzed in terms of their input-output behavior by means of Boolean algebra.

Since computers are built out of primitive computing components, the fact that primitive computing components can be built out of an indefinite number of different materials and designs is why computers can be built out of components with indefinitely many different physical properties.

This multiple realizability of primitive computing components may have helped generate the temptation to see everything as a computer.[8] But that temptation should be resisted: from the fact that any material or almost any material can be used to build a primitive computing component, it doesn't follow that everything is a computer. This is because in order to constitute a primitive computing component, a physical system must still exhibit certain very specific and well-defined input-output properties, which can be exploited in designing and building computing mechanisms. And in order for a mechanism made out of primitive computing components to be a full-blown computer, those components must be functionally organized in very specific ways. One of the burdens of this section is to show that for something to be a computing mechanism with interesting computing power, its components must exhibit very specific properties and be organized in very specific ways. What is peculiar to computing mechanisms is that in order to understand their computations, they can be analyzed or designed (up to

[8] This temptation was amply documented in Chapters 7 and 8.

their primitive computing components) with complete disregard for their physical composition and dynamical properties, using only principles of computer design and computability theory.[9]

10.2.1.1 Comparison with Cummins's Account

The only philosopher who has devoted significant attention to the primitive components of computing mechanisms is Cummins (Cummins 1983). The present account of primitive computing components can be usefully contrasted with Cummins's account. Under both accounts, the input-output behaviors of primitive components are taken as primitive relative to the computing mechanism they are part of, i.e. the input-output behaviors of primitive computing components are not analyzed in terms of simpler computations performed by them or their components. But there are some important differences between the two accounts.

First, Cummins says that primitive computing components "instantiate" the functions from their inputs to their outputs as a matter of physical law, without specifying whether the law is strict or *ceteris paribus*. I add to Cummins's account the point that primitive computing components instantiate their input-output functions *ceteris paribus*, i.e. not because they fall under a strict law but because it is the proper function of the component to instantiate it. Primitive computing components, just like any other mechanisms, can malfunction, break, or be malformed or defective.[10] This point generalizes to non-computing components (see below) and to anything built our of primitive computing components, for two reasons: (1) if a primitive computing component malfunctions, or breaks, or is malformed or defective, this may carry over into a malfunction, or break-down, or malformation or defect in the containing system—though this may not be the case if there is redundancy in the system; (2) the containing system is a mechanism too, and it has its own ways of malfunctioning, breaking, being malformed or defective.

[9] Real hardware designers, of course, design their machines while keeping in mind constraints coming from the technology that is going to be used to implement their design. This does not affect the point that in principle, different technologies can be used to implement the same design or similar designs.

[10] Millikan 1993, chapt 1 criticizes Cummins's account of functional analysis because, she argues, it is incapable to accommodate the notion of malfunction. I think Cummins's account might be amended to respond to Millikan's criticism, but Millikan is right at least in pointing out that Cummins pays no attention to the issue of malfunction.

From now on, I will take this point for granted and won't repeat it for other kinds of components or mechanisms.

Second, for Cummins the input-output function instantiated by a primitive computing component is a step in the computation performed by the whole computing mechanism, whereas for me it is only part of the activity performed by the component that contains the primitive computing components, an activity that may or may not be a computation. In Cummins's account, computation is always the execution of a program, which is analyzed into primitive operations. These operations are the primitive steps, and may all be performed by the same components. In my account, there are more levels of analysis. Not all computations are executions of programs, and before we get to program execution, we have to understand the complex components made out of primitive computing components. For now, all that a primitive component contributes to is its containing mechanism, which is not necessarily a program-executing mechanism. When we get to program execution, we'll see that there is no need to assume that what a primitive computing component computes is an elementary step in the execution of a program.

Third, for Cummins it is inappropriate to say that a primitive computing component computes a function. We should say only that it instantiates a function. I prefer to say that it is not insightful to say that a primitive computing component, isolated by itself, computes a function (e.g., that an AND gate computes whether both of its inputs are 1's). But in the context of a computing mechanism and of designing computing mechanisms, it may be useful to talk of primitive computing components as performing computations. It is useful because it allows one to analyze the computations performed by the whole mechanism in terms of the computations performed by the primitive components using certain kinds of analytic tools, such as Boolean algebra or truth tables. This is important in light of the fact that computer designers like to say that logic gates compute. Rather than accusing them of misspeaking, it is better to make sense of what they say.

10.2.2 Primitive Non-computing Components

Not every component of a computing mechanism performs computations. We've already seen this in the case of the components of primitive computing components, which do not deserve to be ascribed computations.

More generally, computing mechanisms contain many components whose function is not computing. Some of these non-computing components act as physical support for the computing components, others provide energy to the mechanism, others generate and spread the signals that keep the mechanism synchronized, yet others transmit inputs and outputs from one computing component to another, etc. None of these components perform any computation, but they are necessary for the functioning of the whole mechanism. So, in judging whether a mechanism is a certain kind of computing mechanism, which requires certain non-computing components for its proper functioning (such as components that keep the system synchronized), it may be useful to see whether it has those non-computing components.

In most computing mechanisms that can change their internal states, synchronization among components is obtained through the use of a *clock*. In modern computing mechanisms, a clock is a mechanism that generates a signal with a fixed *cycle time*. A cycle time is a temporal interval divided into two subintervals, during which the clock signal takes, respectively, a high and a low value. A fixed cycle time is a cycle time of fixed length, which repeats itself. Clocks are used in computing mechanisms with internal states; clocks determine how often, and at what time, the mechanisms update their states. The main constraint of clocked computing mechanisms is that the clock cycle time must be long enough to allow the signals determining state changes to stabilize on the values that are relevant to the state change.

When a component receives an input signal that is not already synchronized with its internal clock, the signal must be synchronized with the component so that it can be properly processed. *Synchronizers* are components that take asynchronous signals and a clock as input and yield a signal synchronous with the input clock as output.

Clocks and synchronizers are usually ignored in computability theory, where the synchronization of abstract computing mechanisms can be assumed to be perfect by fiat. But clocks and synchronizers are among the most important non-computing components of concrete computing mechanisms. A clock malfunction or a mistake in the design of the clock and related synchronization mechanisms, such as a too short cycle time, can render a computing mechanism useless, because its components will not update their states at the time when their input signals have stabilized on the relevant values, or will update their states in the wrong order.

Another important function of certain non-computing components is to help isolate computing mechanisms from various elements of their environment, which might interfere with the mechanisms' proper functioning. Like most other mechanisms, computing mechanisms can be damaged, or their proper functioning can be disrupted, by an indefinite number of environmental factors, such as pressure on their rigid components, too high or too low temperature, electrical or magnetic fields, etc. What factors interfere depends on the specific physical and functional characteristics of a given mechanism. In order to protect computing mechanisms from the interference of relevant external factors, precautions (e.g., surrounding the mechanism by a protective shell) must be taken in designing and building them.

10.2.3 Complex Computing Components

Primitive computing components can be combined together and with non-computing components to form complex computing components. The important point here is that the computation performed by the complex component can be exhaustively analyzed in terms of the computational steps performed by the primitive computing components and the token manipulations performed by the non-computing components. So one can design a complex component by appealing only to the known functional properties of its components with respect to token manipulation, without worrying about how the primitive computing components will physically work together. The details of the physical interactions between primitive components are left out as "implementational details," by relying on the assumption that those who will build the mechanism will rely on components that are physically compatible with one

another and physically connect them in an appropriate way. Since all complex computing components are identified by their functional analysis, which includes the computations they perform, the same point applies to complex computing components built out of simpler but non-primitive computing components.

The way the components of a computing mechanism are combined together to form the whole mechanism depends on the specific characteristics of the components. Typically, each component has appropriate extremities, which carry token signals into and out of the component. The extremities of two components can be physically joined together, so that the signals coming out of one component can enter the other component.

An important property of complex computing components is that their functional analysis can be carried out simply by deducing the computational properties of the complex component from the computational properties of its components together with their functional organization, without needing to take into account their physical composition or dynamical properties in real time. The computational properties of complex computing components depend on their specific functional properties, and do not apply to mechanisms that do not share those functional properties.

10.2.3.1 Combinational Computing Components

Combinational computing components take a fixed finite number of input tokens and yield a fixed finite number of output tokens that depend only on their input (and not on internal states). Combinational computing components cannot assume different internal states, and hence they always yield the same output in response to the same input.

In modern computers, complex computing components are built by joining a number of logic gates at their extremities. Logic gates can be hooked up to one another to form combinational computing components called Boolean circuits (because they are combinations of logic gates, each of which computes a Boolean operation). Some sets of logic gates are called *universal*, because combinations of them are sufficient to design all Boolean circuits, i.e. circuits for all logic functions, i.e. functions expressible as truth tables or logic equations in Boolean algebra. For example, the set {AND, NOT} is a

universal set, and so is the set {OR, NOT}. This sense of universality should not be confused with the universality of Turing Machines, which was discussed in Chapter 1 and will be revisited below in relation with computing mechanisms.

Figure 10-2 presents a simple Boolean circuit called half adder, which is composed of two AND gates, one OR gate, and one NOT gate, connected together at their extremities. A half adder takes two tokens, labeled A and B, as input and yields two tokens, labeled Sum and Carry, as output. The function of this circuit is to generate an output that is naturally interpreted as the sum of the two input signals, each of which is now interpreted as representing the numbers 1 and 0. The Sum output is 1 if and only if either A or B is 1, but not both. The Carry output is 1 if and only if both A and B are 1. Under this interpretation of the tokens as representing numbers, this simple combination of logic gates can be seen as performing the arithmetical operation of two-bit addition. Due to the versatility of arithmetic, this is very convenient in designing complex computing components.

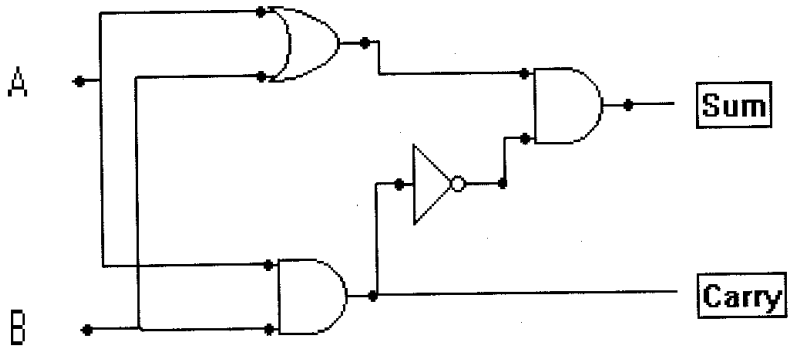

Figure 10-2. Half (two-bit) adder.

The circuit in Figure 10-2 takes only two tokens (A and B) as input, which may be seen as two strings of one token each. In designing computing mechanisms, it is useful to have mechanisms that can manipulate strings of more than one token. For example, one way to sum two strings of more than one bit each is to couple the individual bits of the two strings in the order in which they occur within the strings (the first two bits of the two strings, the second two bits, etc.), then sum the individual couples of bits while also adding any eventual carry out tokens from each of the couples of bits to the next couple of bit. In order to do this, the circuit in Figure 10-2 is insufficient, because it does not take a carry out signal as input. If one wants to generate the sum of two strings that are longer than one token each, one needs the slightly more complex circuit depicted in Figure 10-3.

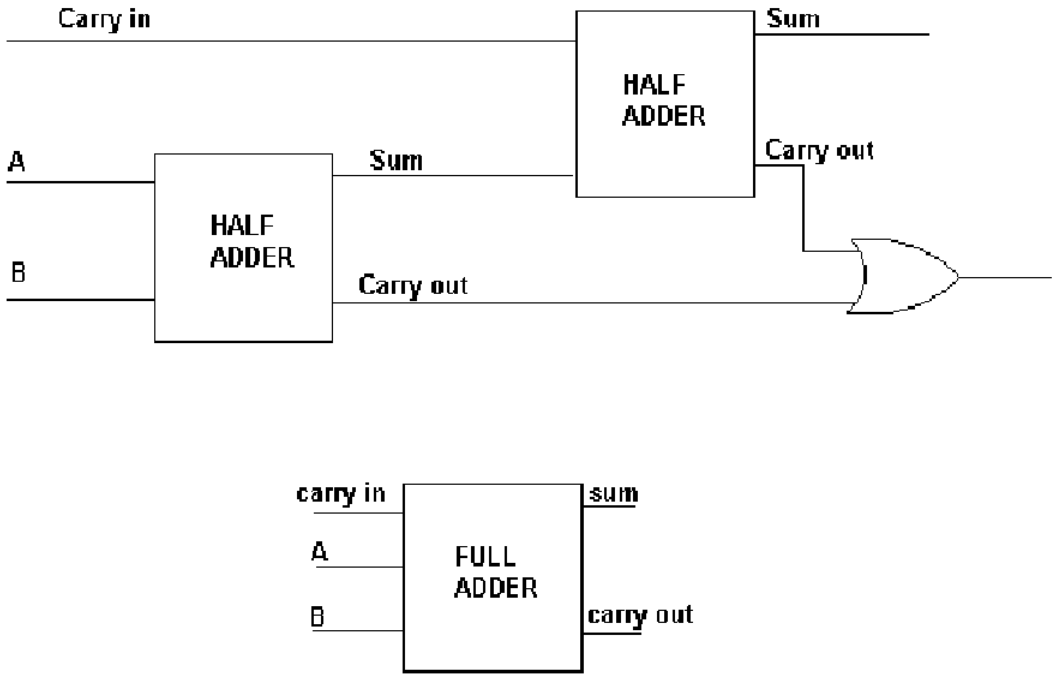

Figure 10-3. Full (two-bit) adder.

By combining two half adders like those of Figure 10-2 in the way depicted in Figure 10-3, one generates a full adder, which takes three tokens—A, B and Carry in—as input and yields two tokens—

Sum and Carry out—as output. Full adders can then be combined together in any number, and the result is a circuit that generates a string of tokens that is naturally interpreted as the sum of two strings of tokens.

When this is done, an ordering is introduced in the input and output tokens. Until now, we dealt with operations on two single tokens at a time. When we move to operations on strings of tokens, which token is first in the string and how the tokens are concatenated together makes a difference to what output must be generated. For example, in the case of addition, the carry out from the first two bits in the string must be given as input to the full adder that adds the second two bits in the string, and so on until the end of the string. Therefore, circuits that manipulate strings of tokens impose an implicit ordering on their input and output tokens, so that there is a unique first token of each string and a unique successor of each token in the string. This ordering of tokens to form strings is a crucial property of complex computing mechanisms, which is implicitly exploited in designing computing mechanisms that perform more complex computations.

Analogously to adders, one can design Boolean circuits that perform other logical or arithmetical operations, such as subtraction, identity, conjunction, negation, etc., on two strings of tokens of arbitrary (finite) length.

Boolean circuits are very convenient because given a desired input-output function constructed out of logical connectives (or equivalently, Boolean operations), there is an effective method for constructing a Boolean circuit that computes it. It is enough to replace each connective by the corresponding logic gate and connect the logic gates together in a way that corresponds to the nesting of the parentheses in the logic equation. Also, given a Boolean circuit, there is an effective method for constructing a truth table or a logic equation representing the logic function computed by the circuit. It is enough to represent each logic gate by the corresponding logical connective (or Boolean operation) and nest the logical propositions in a way that corresponds to the connections between the logic gates.

10.2.3.2 Arithmetic Logic Units

Boolean circuits can be combined to form various more complex computing components. In modern computers, a particularly important complex computing component is a combination of Boolean circuits called Arithmetic Logic Unit (ALU). An ALU operates on two input strings of fixed length, such as 16 or 32 bits, in combination with a few other input tokens that act as control signals. Control signals determine which operation the ALU performs on the two input strings. ALUs then yield one output string of fixed length, which is the output of the operation performed on the input strings. Like complex Boolean circuits, ALUs impose an ordering on their inputs and outputs, so that there is a unique first token of each string of inputs and outputs, and a unique successor to each token. The name "arithmetic logic unit" comes from the fact that in modern computers, ALUs are designed so that their operations are naturally interpreted as either arithmetic operations (such as addiction and subtraction, but not multiplication or division) or logic operations (such as conjunction and disjunction) on their inputs.

ALUs are part of the core of modern computers, but computers do not *have* to be built around ALUs. Arithmetic and logical operations are used as primitives in modern computers because computer designers find them convenient to use in defining more complex operations. Other primitive operations could be used. Moreover, in explaining the behavior of an ALU we don't need to appeal to the fact that the functions it computes are arithmetic or logic; all we need to understand is how its components interact in the presence of certain inputs. The characterization of the unit and its operations as arithmetic or logic is done because it is useful in designing and programming the whole computing mechanism, just as the characterization of logic gates as logic is useful in designing complex computing components.

10.2.3.3 Sequential Computing Components

Sequential computing components are composed of combinational computing components and memory units. Memory units are described in more detail in section 10.2.4.1. For now, suffice it to say that the

presence of memory units allows sequential computing components to assume a number of internal states. The output tokens generated by sequential computing components depend both on the input tokens the components receive and on the content of their memory, that is, their internal state.

Since they use memory components, sequential computing components are more powerful than combinational computing components. Their computational behavior cannot be described by truth tables or logic equations, but it can be described by finite state automata (FSA). A FSA can be represented formally as a finite set of states plus two functions, called next-state function and output function. The set of states corresponds to all the possible contents of the memory of the component. The next-state function says, given the current state of the component and its input, what the next state should be. The output function says, given the current state of the component and its input, what the output of the component should be. The FSA used in designing modern computers are synchronous, i.e. they update their internal state and generate their output once every clock cycle.

Just like there are effective methods for generating a circuit design given a logic equation or truth table and *vice versa*, there are effective methods for generating a circuit design given a FSA and vice versa.[11]

10.2.3.4 Multiplication and Division Components

In modern computers, multiplication and division are heavily used, so there are circuits that are hardwired to perform these operations on words of fixed length (e.g., 32 bits). Multiplying finite strings of tokens requires several additions or subtractions of strings while keeping track of how many steps have been performed and of intermediate results. Thus multiplication and division of finite strings of tokens are more complex operations than addition and subtraction. They cannot be performed by ALUs, which have no means to store intermediate results and keep track of intermediate steps. Multiplication and division components are sequential computing components.

[11] For details, see Patterson and Hennessy 1998, pp. B38-B39.

Multiplication and division components are simple FSAs made out of an ALU, a few memory registers to keep track of intermediate results, and a control mechanism that keeps track of the number of steps to be performed at any given time. But like ALUs, multiplication and division components take a fixed number of tokens as inputs and yield a fixed number of tokens as output. They go through a cycle of internal operations that stops when the operation is completed, at which time they output their result and reset themselves to their initial state.[12]

10.2.3.5 The Computation Power of Complex Computing Components

The computation power of complex computing components is nontrivial and much more interesting than that of primitive computing components, but there are still interesting differences between different kinds of complex computing components. One way to understand the difference in computation power between different computing components is to distinguish between algorithms and *pseudo*-algorithms. A genuine algorithm is an effective procedure defined over infinitely many inputs of arbitrary length, whereas a pseudo-algorithm is an effective procedure defined over finitely many inputs of bounded length. The distinction between algorithms and pseudo-algorithm is often underestimated in its importance.[13]

Pseudo-algorithms can be further divided into *simple* ones, which require at most one operation from each component of a mechanism, and *complex* ones, which require more than one operation from at least one component of the mechanism. Complex pseudo-algorithms, such as pseudo-algorithms for 32 bit multiplications, require a mechanism to go through several cycles of activity before terminating and generating an output. This requires recurrent connections within the mechanism and a control structure that keeps track of the cycles of activity, inputs, and outputs.

---

[12] For more details on the design of multiplication and division components, see Patterson and Hennessy 1998, pp. 250-274.

[13] An example of somebody who underestimates the importance of this distinction is Cummins. He argues that four-bit adders execute algorithms for addition (Cummins 1983, Appendix). This makes it sound like every computing mechanism, including a simple four-bit adder, has the same computation power; namely, to execute an algorithm. But four-bit adders do not literally compute the whole addition function; they compute at most four-bit addition. So they cannot possibly execute algorithms for addition; at most they execute pseudo-algorithms for (four-bit) addition.

In simple Boolean circuits, the interest in their computations derives on the one hand from the fact that the function computed by the circuit can be exhaustively analyzed in terms of the computations performed by its component logic gates using Boolean algebra or truth tables, and on the other hand from the fact that Boolean circuits can be put together to form ALUs and other complex computing components. There is no sense in which Boolean circuits can be said to execute a program or an algorithm, because they operate on inputs of bounded length. But Boolean circuits can be said to execute simple pseudo-algorithms on their inputs.

ALUs are computing mechanisms capable of executing a finite number of simple pseudo-algorithms, but they still have no power to execute genuine algorithms or even complex pseudo-algorithms. ALUs are computationally more powerful than ordinary Boolean circuits in that they can execute more than one operation on their inputs, depending on their control signal. Just like in the case of Boolean circuits, the simple pseudo-algorithms executed by an ALU are not represented by a program in the ALU but are hardwired in the functional organization of the ALU.

Multiplication and division components, and other similar fixed-length sequential computing components, execute complex pseudo-algorithms, which require them to go through several cycles of activity before generating an output. In order to do this, they must have a control structure that keeps track of their cycles of activity and terminates the computation at the appropriate time. Because they require several cycles of activity, multiplication and division are more complex operations than those computed by ALUs, and multiplication and division components are correspondingly more powerful components than ALUs. But multiplication and division components, like ALUs and Boolean circuits, are still limited to computing functions of inputs of bounded length.

None of these components have the power to execute genuine algorithms, in the way that ordinary FSAs or TMs do. A fortiori, they don't execute programs and are not computationally universal. To obtain computing mechanisms with these interesting properties, we need to reach a higher level of functional organization.

10.2.4 Complex Non-computing Components

Computing and non-computing components can be combined together to form complex non-computing components. For example, *decoders* are components that receive *n* tokens as inputs and yield 2^n output tokens, in such a way that only one of the output tokens takes the value 1 for each of the possible input combinations. Decoders are built out of logic gates and they are useful to build larger computing and non-computing mechanisms (e.g., memory units), but their function is simply to turn a specific string of tokens coming from a bundle of lines of communication into a single token coming from a specific line.[14]

10.2.4.1 Memory Units

A memory unit is an input-output component that assumes and maintains one among a number of stable states, which may be fixed or variable depending on some of the memory's input tokens, and can generate (when triggered by an appropriate signal) one or more output tokens that reflect its internal state. After the state of a memory unit is updated, it remains the same until it is updated again. Because of this, a memory unit effectively stores one or more tokens. Memory units employed in the design of modern computers are clocked, i.e. they update their state at the appropriate time during every clock cycle. This ensures that their state change is synchronized with the behavior of components that generate their inputs.

Logic gates can be combined together to form memory *cells*, i.e. memory units that store one token. Memory cells can be combined together in arrays to form memory *registers*, which can store a string of tokens. Registers can be combined with other non-computing mechanisms (such as decoders) to form *register files* and RAMs (random access memories). These are large memory units arranged so that supplying a specific string of tokens (called *address*) as input to the whole unit can be used to retrieve or update the content of a specific register.[15]

[14] For more details on the design of decoders and other relatively simple non-computing components out of logic gates, see Patterson and Hennessy 1998, pp. B8-B18.
[15] For more details on the design of memory units, see Patterson and Hennessy 1998, pp. B22-B33.

Memory cells and larger memory units don't compute by themselves, but they are necessary components of computing mechanisms that need to store data, programs, and intermediate results of computations.

The fact that memory units can be made out of logic gates shows that something that can be usefully seen as a computing mechanism, such as a logic gate, can be used to build components whose proper function is different from computing, such as memory components. (More generally, logic gates have applications outside computer design.)

10.2.4.2 Datapaths

Memory components and other non-computing components can be added to ALUs and multiplication and division components to form *datapaths*, i.e. components that perform the operations that are primitive within a computer. Datapaths act on inputs of fixed (finite) length and generate the corresponding outputs of fixed (finite) length. Datapaths execute one out of a finite number of operations on their inputs depending on their (control) state.

We've seen that ALUs and other computing components, such as multiplication and division components, perform a finite number of operations on inputs of fixed length. For simplicity we'll only speak of ALU operations, but the same applies to other complex computing components such as multiplication and division components. Which operation the ALU performs depends on other inputs it receives. So for any operation that the ALU can perform, there are strings of tokens that—when given as input to the ALU—determine which operation the ALU performs and specify the inputs on which it operates. Strings of tokens that trigger operations on certain inputs by the ALU can be stored in memory registers, and are a type of instruction called *arithmetic-logic instruction*. One function of the datapath is to retrieve arithmetic-logic instructions from the registers in which they are stored, convey them to the ALU, and put the result of the operation into an appropriate memory register. This process is a form of instruction execution.

After one instruction is executed, the datapath must be prepared to execute another instruction. Instructions are stored in memory registers, and the content of each memory register can be either written or retrieved by sending a certain string of tokens as input to the memory (this is called *addressing* the memory). One function of the datapath is to keep track of which instruction is being executed at any given time by holding the string corresponding to the register that holds it in an appropriate register, called *program counter*. A related function of the datapath is to determine which instruction comes next, which can be done by updating the content of the program counter in an appropriate way. This can be done by a combination of the following: storing instructions in consecutive memory registers, having a convention for addressing memory registers that uses consecutive strings for consecutive registers, and replacing the string in the program counter by its successor after every instruction execution.

Instead of having arithmetic-logic instructions specify the data for the arithmetic-logic operations, it is convenient to have those data stored in an appropriate register file within the datapath. The content of the register file can be altered by appropriate strings of tokens, which specify which register must be altered and what must be put into it. An analogous string of tokens can be used to take the content of a register within the datapath and copy it into a register in the main memory of the computing mechanism. These strings of tokens, which load and store words between the datapath and the main memory of the computing mechanism, are called *memory-reference instructions*. By using memory-reference instructions, data for computations can be moved back and forth between the datapath and the main memory independently of the instructions that specify which operations must be performed on them. One function of the datapath is to retrieve memory-reference instructions, retrieve the string of tokens from the input register indicated by the instruction, and put the same string of tokens into the output register indicated by the instruction.

As defined so far, the datapath is confined to executing instructions in the order in which they are stored in memory, without ever going back to previously executed instructions or skipping some of the

instructions. This limits the computation power of a computing mechanism.[16] To overcome this limitation, the datapath can be set up so that it can update the program counter in a way that depends on whether the content of a certain register satisfies some condition that can be checked by the ALU. Then, there will be strings of tokens that can specify under what condition the program counter should be updated in the normal way, and under what condition it should be updated abnormally, and what this abnormal update should be. These strings are called *branch instructions*. One function of the datapath is to retrieve branch instructions from the memory registers where they are stored, use the ALU to determine whether the condition they indicate is satisfied, and update the program counter accordingly.

It should be clear by now that the datapath has many important proper functions, whose accomplishment (in the appropriate order) constitutes the computations performed by a computing mechanism capable of executing instructions. What the datapath needs in order to accomplish its various jobs is a mechanism that determines, for each type of instruction, which job the datapath must accomplish. One way to determine which type each instruction belongs to is to include within each instruction a sub-string, in a conventional fixed place (e.g., the beginning of the instruction), and to use different sub-strings for different instructions types. To react appropriately to that instruction sub-string is the function of the control unit.

10.2.4.3 Control Units

A control unit receives as input the part of an instruction that determines its instruction type, and outputs a signal that sets up the datapath to do the job corresponding to that instruction type. When seen in isolation, a control unit can be described as a computing component, such as a Boolean circuit or a simple FSA with inputs and outputs of fixed size. But the control unit's contribution to its containing system is a control function. A control unit has the function of setting up the datapath to accomplish the kind of job that corresponds to each type of instruction when an instruction is executed.

[16] Specifically, such a computing mechanism is limited to computing primitive recursive functions. It cannot compute partial recursive functions that are not primitive recursive.

Depending on the design and timing methodology, a control unit may be a combinational or a sequential computing component. For instance, if all the events within the datapath take place during one clock cycle, then the control unit can be a combinational component. But if different events within the datapath take place during distinct clock cycles, then the control must keep track of which event is taking place at any given time and which event comes next, so that it can send the appropriate signals to the datapath. In order to do this, the control unit must have a register that keeps track of the different stages of the execution, and so it must be a combinational component.

A crucial function of the control unit is to deal with at least some kinds of events that may result in a mistake in the computation. An event of this type is the presence of an instruction whose execution requires more memory than is available.[17]

Together, the control unit and the datapath constitute the *processor*, which is the core computing mechanism of modern computers. In modern computers, the processor is the mechanism that actually carries out the computations.

Just as in the case of ALUs and other complex computing components, the inputs and outputs of memory units and processors are concatenated strings of tokens, in which each token occupies a well defined position. Different parts of the string may have a different functional significance depending on where they are located along the string. For example, the first part of a string may determine the instruction type, whereas other parts may determine which registers must be accessed for retrieving data. This ordering of strings is accomplished within the mechanism by the structure of components together with the appropriate ordering of the communication lines between the components. Because of this, an observer can tell which is the first token in a string, which is its successor, and so on until the end of the string, so that each string is unambiguously identified. Without this ordering of the tokens to form strings, complex computing and non-computing mechanisms would not be able to function.

[17] For more details about how control units avoid certain kinds of miscomputations, see Patterson and Hennessy 1998, pp. 410-416.

10.2.4.4 Input and Output Devices

There is no room here to review the many important properties of input and output devices. We will focus on what is essential for present purposes.[18]

An input device is a mechanism through which a computing mechanism receives inputs, i.e. strings of tokens coming from the environment external to it. Input tokens must be concatenated in appropriate ways, so that the mechanism can respond differentially to the first token in a string and to the successor of each token in a string. This ordering of the input tokens is guaranteed by the input devices and is then preserved throughout the computing mechanism. In ordinary computers, examples of input devices include keyboards, mice, and scanners.

An output device is a mechanism that conveys the tokens generated by a computing mechanism, or outputs, to the mechanism's environment. Output tokens are also concatenated, so that the receiver of an output from a computing mechanism can (in principle) tell which token starts a string and which token follows each other token, until the end of the output string is reached. This ordinal arrangement of outputs is guaranteed by the functional properties of the output device. In ordinary computers, examples of output devices include monitors and printers.

10.2.4.5 Internal Semantics

We've seen that in designing and analyzing computing mechanisms, the operations computed by logic gates and Boolean circuits are naturally interpreted as logical operations, and the operations computed by arithmetic-logic units are naturally interpreted as arithmetical and logical operations. This natural interpretation is an external semantics, relating the inputs and outputs of these computing mechanisms to objects external to the mechanism. External semantics is not necessary to identify these components and the computations they perform, because as I argued in sections 9.2 and 9.3, they could be identified equally well as computations defined over strings. But external semantics is useful in designing complex

[18] For more details on input and output devices, see Patterson and Hennessy 1998, chap. 8.

components out of simpler components or in analyzing the computations of complex components into those of their simpler constituents.

In discussing the operations of ordinary processors, the input data, the intermediate results, and the outputs of the computation are naturally interpreted as referring to numbers. The same considerations apply: this external semantics is not necessary to identify the strings of tokens and the operations performed on them. Everything that processors do could be characterized in terms of operations on strings, without saying anything about what the strings represent.

However, not all strings manipulated by a processor are data, intermediate results, or outputs. Many are instructions or parts of instructions. It is natural and useful, in designing and programming a computer, to interpret instructions as representing what a processor is going to do in executing them. This interpretation does not assign an external semantics, because it does not relate strings to objects external to the machine. I call it *internal semantics*.

A processor shifts instructions and data between the main memory of the computing mechanism and its internal registers, shifts data between its internal registers and its datapath, and controls its datapath so that it performs certain operations on the data. All of this is possible because of how the instructions are written and how the processor is functionally organized to respond to various parts of the instructions.

Part of each instruction is fed to the control unit, and the way it affects the control unit determines which operation the processor performs. This part of the instruction is naturally interpreted as representing a type of instruction, to which a type of operation corresponds (e.g., addition, subtraction, loading, writing, branching, etc.). Depending on the instruction, parts of the instruction may be fed to the datapath's register file, where they activate different registers depending on their value. These parts of the instruction are naturally interpreted as containing *addresses* of the registers within the datapath, i.e. as naming the registers. Depending on the instruction, parts of an instruction may be fed to registers of the main memory, where they activate different registers depending on their value. These parts of the instruction are naturally interpreted as naming the registers within the main memory. Finally, part of an

instruction may be fed to the datapath as input to an operation. This part is naturally interpreted as containing data.

Not all instructions have all of the above-mentioned parts, although in modern computers, all of them have the first part. Moreover, each (machine-language) instruction part must have a specific and fixed length and be placed in a fixed position within the instruction, so that the processor can react appropriately to each part of the instruction. Only when placed in an appropriate position within the rest of the instruction does a string of tokens acquire its internal semantics. The same string of tokens placed in different parts of an instruction, or in the same position within different instructions, will represent completely different things (e.g., a memory address, a datum, or an operation). In this sense, internal semantics is not intrinsic to the "syntactic" type of the strings (that is, which tokens constitute the string), but rather it is context sensitive.[19]

Because of how their parts are segmented and manipulated by the processor, arithmetic-logic instructions can be interpreted as telling which registers' contents must be manipulated in which way, and which register the result must be put in. Memory-reference instructions are naturally interpreted as telling which registers' contents must be shifted into which other registers. Branch instructions are naturally interpreted as telling how the program counter must be updated under which circumstances. Designers and programmers use this natural, internal semantics of instructions to interpret and write instructions and programs. Under this internal semantics, instructions and programs constitute a *code*, which is commonly referred to as the computer's *machine language*.

10.2.5 Calculators

Calculators are relevant to computationalism because they are paradigmatic examples of computing mechanisms, yet they are never used as models of the brain. Computationalists assert that the brain is a computer not a calculator. The functional account of computing mechanisms sheds light on why this is.

[19] This is not only true for instruction parts. The same string of tokens may constitute an instruction or a datum depending on which part of the memory it is placed in.

A calculator is a computing mechanism composed of four types of components, appropriately connected together: input devices, output devices, memory units, and processors. In this section I only analyze non-programmable calculators. So called "programmable calculators," from a functional perspective, are special purpose computers with small memory, and are subsumed within the next section.

Calculators take strings of discrete tokens as inputs and return strings of discrete tokens as output, and their outputs are computable functions of the inputs. However, the computing power of calculators is limited by at least three important factors.

First, calculators cannot perform branches in their computations, namely, they cannot perform different operations depending on whether some condition obtains. Branching between two different operations is necessary to compute all computable functions. Since they cannot branch, calculators are not universal computing mechanisms, i.e. they cannot compute all computable functions.

Second, the range of values that calculators can compute is limited by the size of their memory and by their input and output devices. Typically calculators' input and output devices only take inputs and deliver outputs of fixed size, hence calculators can only operate within the size of those inputs and outputs.

Finally, a calculator's output may be one of a (finite) number of functions of the input. Which function is computed by a calculator is determined by setting the calculator on one of a finite number of initial states that correspond to the functions the calculator can compute. This setting is typically done by pressing an appropriate button. Given each initial condition, the calculator's control mechanism sets up the calculator so that its internal computing mechanism performs the appropriate operation on the input. The functions computed by a calculator can be combined in sequences but not changed (e.g., by adding new instructions or programs); that is to say, calculators are not programmable.

Calculators are computationally more interesting and powerful than the computing components discussed in section 10.2.3. Nevertheless, their computing power is limited in a number of ways. Contrasting these limitations with the power of computers explains why computers, rather than calculators, appeal to computationalists as models for the brain.

10.2.6 Computers

We now have all the ingredients to understand today's most popular computing mechanisms: computers. This discharges the burden acquired in Chapter 8, when I promised I would explicate the notion of program-executing mechanism, i.e. a mechanism whose behavior is explained by appeal to program execution.

Like a calculator, a computer is made out of four types of components: input devices, output devices, memory units, and processors. In modern computers, processors can be analyzed as a combination of datapaths and control units. A schematic representation of the functional organization of a computer (without input and output devices) is shown in Figure 10-4.

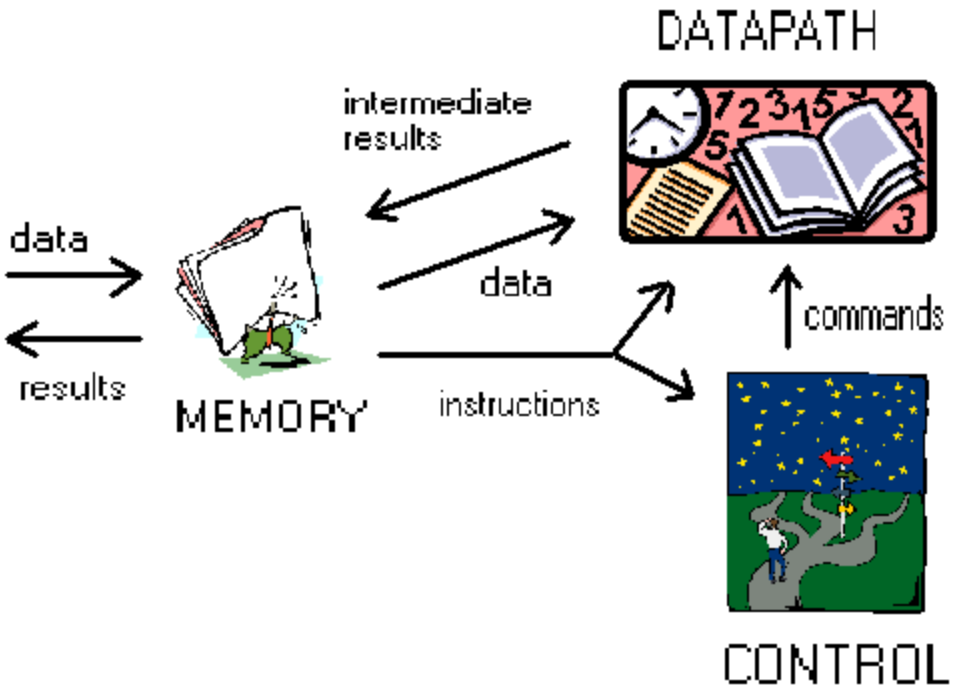

Figure 10-4. The main components of a computer and their functional relations.

Computers and their components can be classified according to the technology they use, which can be mechanical, electro-mechanical, electronic, etc., as well as according to other characteristics (size, speed, cost, memory size, etc.). These differences don't matter for our purposes, because they don't affect the computation power of computers.

The crucial difference between calculators and computers is in the functional properties and functional organization of their components. Computers' processors are capable of branching behavior, and can be set up to perform any number of primitive operations in any order. Computers' memory units are orders of magnitude larger than those of calculators, and often they can be increased in a modular fashion if more storage space is required. This allows computers to take inputs and programs, and yielding outputs, of unbounded size. So computers, unlike calculators, are programmable, capable of branching, and capable of taking inputs and yielding outputs of unbounded size.

Programmability is explained in the next subsection. In the rest of this section on computers, I will draw other important functional distinctions between classes of computers, which are relevant to discussions of computationalism.

10.2.6.1 Programmability

At least until the 1940s, the term "computer" was used to designate people whose job was to perform calculations, usually with the help of a calculator or abacus. Unlike the calculators they used, these human computers could compute the values of any function for whose calculation they received appropriate instructions. The term "computer" started to be applied to machines when machines were developed that could be easily modified to compute large classes of computable functions up to their memory limitations. This feature, called programmability, is the essential feature of (what are ordinarily called) computers.

Being programmable means being modifiable so that a different function is computed depending on the modification. What counts as a legitimate modification depends on the context of use and the means that are available to do the modification. In principle, one could modify a calculator by taking it

apart and rewiring it or adding new parts, and then it would compute new functions. But this kind of modification would ordinarily be described as building a new machine rather than programming the old one. So we should relativize the notion of programmability to the way a machine is ordinarily used. This will not make a difference in the end, especially because the kind of programmability that matters to computationalism is soft-programmability (see below), where what counts as a relevant modification is determined unambiguously by the functional organization of the computer.

Programmability comes in two main forms, hard and soft, depending on the type of modification that needs to be made to the machine for it to compute a different function.

10.2.6.1.1 Hard Programmability

Some early computers had plug-ins and wires with plugs, which sent signals to and from their computing components. Plugging the wires in different configurations had the effect of connecting the computing components of the machine in different ways, to the effect that the machine would perform different calculations. In other machines, other mechanisms (e.g., manual switches) were used to physically connect different computing components so that they performed operations in the appropriate order. I call any computer whose modification involves the rewiring of the machine *hard programmable*. Hard programming requires that users change the way the computing components of a computer are spatially joined together, which changes the number of operations that are performed or the order in which they are performed, so that different functions are computed as a result.

Hard programming requires technical skills and is relatively slow. To make programming easier and more efficient, soft programmability of computers was introduced and quickly became the standard.

10.2.6.1.2 Soft Programmability

Modern computers contain computing components that are designed to respond differentially to different sequences of tokens, so that different computations are performed. These sequences of tokens, then, act as instructions to the computer, and lists of instructions are called programs. In order to program these machines, it is enough to supply the appropriate arrangement of tokens, without manually rewiring any of

the components. I call any computer whose modification involves the supply of appropriately arranged tokens (instructions) to the relevant components of the machine *soft programmable*.

There are two kinds of soft programmability: external and internal. A machine is soft programmable *externally* if and only if it requires the programs to be inserted into the machine through an input device, and does not have components that copy and store the program inside the machine. For example, universal Turing Machines and some early punched cards computers are soft programmable externally.

A machine is soft programmable *internally* if and only if it contains components whose function is to copy and store programs inside the machine and to supply instructions to the computing components of the machine. Internal soft programmability is by far the most flexible and efficient form of programmability, and it has become so standard that other forms of programmability are all but forgotten or ignored. One common name for machines that are soft programmable internally, which include all of today's desktops and laptops, is stored-program computers. For present purposes, however, it is best to use the term "stored-program" to refer to the presence of programs inside a mechanism, whether or not those programs can be modified.

10.2.6.2 Stored-Program Computers

A stored-program computer has an internal memory that can store strings of tokens. Programs are stored in memory as lists of instructions placed in appropriate memory units. A stored-program computer also contains at least one processor. The processor contains a tracking mechanism (program counter) that allows the processor to retrieve the instructions in the appropriate order. The processor can extract strings of tokens from memory and copy them into its internal registers (if they are data) or execute them (if they are instructions). After an instruction is executed, the tracking mechanism allows the control to retrieve the next instruction until all the relevant instructions are executed and the computation is complete.

A stored-program machine need not be programmable. Some kinds of memory units are read-only, namely they can communicate their content to other components but their content cannot be

changed. Stored-program calculators can be built using this kind of memory. Other kinds of memory units are such that tokens can be inserted into them. This opens up the possibility that a program be inserted from the outside of the computer (or from the inside, i.e. from other parts of the memory or from the processor) into a memory unit, allowing a computer to be (soft) programmed and the existing programs to be modified. Once the program is in the appropriate part of the memory, the computer will execute that program on any input it gets. A programmable stored-program computer can even be set up to modify its own program, changing what it computes over time. Stored-program computers are usually designed to be soft programmable and to execute any list of instructions, including branch instructions, up to their memory limitations, which makes them computationally universal (see below). For this reason, the term "stored-program computer" is often used as a synonym for "universal computer."

The idea to encode instructions as sequences of tokens in the same form as the data for the computation and the idea of storing instructions inside the computer are the core of the notion of programmable, stored-program computer, which is perhaps the most fundamental aspect of modern mechanical computing.[20] At this point, it should be clear that not every computing mechanism is a computer, and not every computer is a programmable, stored-program computer. In order to have a programmable, stored-program computer, one needs to have a system of tokens, a scheme for encoding instructions, a writable memory to store the instructions, and a processor that is appropriately designed to execute those instructions. Several technical problems need to be solved, such as how to retrieve the instructions from the memory in the right order and how to handle malformed instructions. These properties of computers, the problems that need to be solved to design them, and the solutions to those problems are all formulated in terms of the components of the mechanism and their functional organization. This shows once again how the functional account of computing mechanisms sheds light on the properties of computers.

[20] Patterson and Hennessy list these two ideas as the essence of modern computers (Patterson and Hennessy 1998, p. 121).

10.2.6.3 Special-Purpose, General-Purpose, or Universal

Many old computers, which were not stored-program, were designed with specific applications in mind, such as solving certain classes of equations. Although they are programmable, i.e. they can be modified to solve different systems of equations at different times, special-purpose computers cannot be modified to compute every computable function. So they are called *special purpose* to distinguish them from their general-purpose successors. At any time, what function a special-purpose computer computes depends on the way it is set up or *programmed* at that time. A special-purpose computer can be either digital or analog (see below). Special-purpose computers are still used for a number of specialized applications, e.g. computers in automobiles are special-purpose.

In the 1940s, Von Neumann and others designed computers whose function was to solve as many problems as possible while being programmable in the most flexible way (see Chapter 5). They were called general-purpose or, as von Neumann said, all-purpose computers. The extent to which computers are general- or all-purpose is a somewhat relative matter, which can be evaluated by looking at how much memory they have, how easy they are to program and use for certain applications, etc. General-purpose computers are programmable but need not be soft programmable, let alone stored-program. They can be digital or analog, but are usually digital.

Assuming the Church-Turing thesis, Turing (1936-7) showed how to design universal computing mechanisms, i.e. computing mechanisms that would take any appropriately encoded program as input and thereby compute *any* computable function.[21] Notice that Turing's universal computing mechanisms, universal Turing Machines, are not stored-program (see below for more discussion of Turing Machines).

We have seen that soft programmable computers can respond to programs of unbounded length and manipulate their inputs (of finite but unbounded length) according to the instructions encoded in the program. This does not immediately turn them into universal computers, for example because they may not have the ability to handle branch-instructions, but no one builds computers like that. Ordinary soft programmable computers, which are designed to execute all the relevant kinds of instructions, are

[21] Turing's universal machines is covered in detail in Chapter 1. The Church-Turing Thesis is discussed at length in Chapter 7.

universal computing mechanisms. For example, ordinary computers like our desktops and laptops are universal in this sense. Even Charles Babbage's famous analytical engine was universal. These computers can compute all Turing-computable functions up to their memory limitations. Because of this, the expressions "general-purpose computer" and "all-purpose computer" are sometimes used as synonyms of "universal computer."

10.2.6.4 Functional Hierarchies

In some of the above sections, we have described the internal mechanism that allows soft-programmable computers to perform a finite number of primitive operations in response to a finite number of the corresponding kinds of instructions. Computers with this capability are computationally universal up to their memory limitations. In order to get them to execute any given program operating on any given notation, all that is needed is to encode the given notation and program using the instructions of the computer's machine language.

In early stored-program computers, this process of encoding used to be done manually by human programmers. It was slow, cumbersome, and prone to errors. To speed up the process of encoding and diminish the number of errors, early computer designers introduced ways to mechanize at least part of the encoding process, giving rise to a hierarchy of functional organizations within the stored-program computer. This hierarchy is made possible by automatic encoding mechanisms, such as operating systems, compilers, and assemblers. These mechanisms are nothing but programs executed by the computer's processor, whose instructions are often written in special, read-only memories. These mechanisms generate virtual memory, complex notations, and complex operations, which make the job of computer programmers and users easier and quicker. I will now briefly explain these three notions and their functions.

When a processor executes instructions, memory registers for instructions and data are functionally identified by the addresses of the registers that contain the data or instructions (cf. section 10.2.4). Moreover, each memory register has a fixed storage capacity, which depends on the number of

memory cells it contains. *Virtual memory* is a way to functionally identify data and instructions by virtual addresses, which are independent of the physical location of the data and instructions, so that a programmer or user need not keep track of the physical location of the data and instructions she needs to use by specifying the register addresses of every relevant memory location. During the execution phase, the physical addresses of the relevant memory registers are automatically generated by the compiler on the basis of the virtual addresses. Since virtual memory is identified independently of its physical location, in principle it can have unlimited storage capacity, although in practice the total number of physical tokens that a computer can store is limited by the size of its physical memory.[22]

In analyzing how a processor executes instructions, all data and instructions must be described as the binary strings (strings of bits) that correspond to the physical signals traveling through the processor, and the functional significance of these strings is determined by their location within an instruction or data string (cf. section 10.2.4). Moreover, all data and instruction strings have a fixed length, which corresponds to the length of the memory registers that contain them. *Complex notations*, instead, can contain any characters from any finite alphabet, such as the English alphabet. Programmers and users can use complex notations to form data structures that are natural and easy to ascribe a convenient external semantics (e.g., English words and phrases). Thanks to virtual memory, these strings can be concatenated into strings of any length (up to the computer's memory limitations). During the execution phase, the encoding of data structures written in complex notations into data written in machine language is automatically taken care of by the functional hierarchy within the computer (operating system, compiler, and assembler).

The processor can only receive a finite number of instruction types corresponding to the primitive operations that the processor can execute (cf. section 10.2.4). *Complex operations* are operations effectively defined in terms of primitive operations or in terms of already effectively defined complex operations. As long as the computer is universal, the Church-Turing thesis guarantees that any

[22] For more details on virtual memory, including its many advantages, see Patterson and Hennessy 1998, pp. 579-602.

computable operation can be effectively defined in terms of the primitive operations of the computer. Complex operations can be expressed using a complex notation (e.g., an English word or phrase; for instance, a high-level programming language may include a control structure of the form UNTIL P TRUE DO ___ ENDUNTIL) that is more transparent to the programmers and users than a binary string would be, and placed in a virtual memory location. A *high-level programming language* is nothing but a complex notation that allows one to use a set of complex operations in writing programs. For their own convenience, programmers and users will typically assign an external semantics to strings written in complex notations. This ascription of an external semantics is often implicit, relying on the natural tendency of language users to interpret linguistic expressions of languages they understand. Thanks to virtual memory, complex instructions and programs containing them can be of any length (up to the memory limitations of the computers). During the execution of one of these programs, the encoding of the instructions into machine language instructions is automatically taken care of by the functional hierarchy within the computer.

The convenience of complex notations and the complex operations they represent, together with the natural tendency of programmers and users to assign them an external semantics, makes it tempting to conclude that computers' inputs, outputs, and internal states are identified by their content, and that computers respond to the semantic properties of their inputs and internal states. For example, both computer scientists and philosophers sometimes say that computers *understand* their instructions. The literal reading of these statements was discussed at length, and rejected, in Chapter 9. Now we are ready to clarify in what sense computers can be said to understand their instructions.

Every piece of data and every instruction in a computer has functional significance, which depends on its role within the functional hierarchy of the computer. Before a program written in a complex notation is executed, its data and instructions are automatically encoded into machine language data and instructions. Then they can be executed in the way described in section 10.2.4. All the relevant elements of the process, including the programs written in complex notation, the programs written in machine language, and the programs constituting the functional hierarchy (operating system, compiler,

and assembler), can be functionally characterized as binary strings. All the operations performed by the processor in response to these instructions can be functionally characterized as operations on strings. The resulting kind of "computer understanding" is mechanistically explained without ascribing any external semantics to the inputs, internal states, or outputs of the computer.

At the same time, data and instructions are inserted into computers and retrieved from them by programmers and users who have their own purposes. As long as the functional hierarchy is working properly, programmers and users are free to assign an external semantics to their data and instructions in any way that fits their purposes. This external semantics applies naturally to the data and instructions written in the complex notation used by the programmers and users, although it will need to be replaced by other external semantics when the complex code is compiled and assembled into machine language data and instructions. This shows once again how the external semantics of a computer's inputs, outputs, and external states is unnecessary to identify computing mechanisms and what functions they compute.

Notice that the content of this section applies only to programmable, stored-program, universal computers. In order to obtain the wonderful flexibility of use that comes with the functional hierarchy of programming languages, one needs a very special kind of mechanism: a programmable, stored-program, universal computer.

10.2.6.5 Digital vs. Analog

The distinction between analog and digital computation is often invoked in discussions of computationalism. It is also a distinction surrounding which there is considerable confusion. For example, it is easy to find claims to the effect that analog computers (or even analog systems in general) can be approximated to any desired degree of precision by digital computers (Rubel 1989), countered by arguments to the effect that some analog systems are not Turing-computable (Siegelmann 1999). There is no room here for a detailed treatment of the digital-analog distinction. For present purposes, it will suffice to briefly draw some of the distinctions that are lumped together under the analog-digital banner.

A first issue concerns modeling vs. analog computation. Sometimes scale models and other kinds of "analog" models or modeling technologies (e.g. wind tunnels, certain electrical circuits) are called analog computers (Hughes 1999, p. 138). This use is presumably due to the fact that analog models, just like analog computers properly so called, are used to draw inferences about other systems. The problem with this use of the term "analog computer" is that everything can be said to be analogous to something else in some respect and used to draw inferences about it. This turns everything into an analog computer in this sense, which trivializes the notion of analog computer. But this use of the term "analog computer" has little to do with computation in the strict sense employed here, hence it should be avoided in discussing computing mechanisms.

A second issue concerns whether a system is continuous or discrete. Analog systems are often said to be continuous, whereas digital systems are said to be discrete. When some computationalists claim that connectionist or neural systems are analog, their motivation seems to be that some of the variables representing connectionist systems can take a continuous range of values.[23] One problem with grounding the analog-digital distinction in the continuous-discrete distinction is that a system can only be said to be continuous or discrete under a given mathematical description, which applies to the system at a certain level of analysis. Thus, the continuous-discrete dichotomy seems insufficient to distinguish between analog and digital computers other than in a relative manner. The only way to establish whether physical systems are ultimately continuous or discrete depends on fundamental physics. Some speculate that at the most fundamental level, everything will turn out to be discrete (e.g., Wolfram 2002). If this were true, under this usage there would be no analog computers at the fundamental physical level. But the physics and engineering of middle-sized objects are still overwhelmingly done using differential equations, which presuppose that physical systems as well as spacetime are continuous. This means that at the level of middle-sized objects there should be no digital computers. But the notions of digital and analog computers have a well-established usage in computer science and engineering, which seems

[23] Cf.: "The input to a neuron is analog (continuous values between 0 and 1)" (Churchland and Sejnowski 1992, p. 51).

independent of the ultimate shape of physical theory. This suggests that the continuous-discrete distinction is not a good way to draw the distinction between analog and digital computers. A second problem with the claim that analog reduces to continuous and digital to discrete is that there are many examples of systems that are intuitively discrete but analog (at the same level of description). For instance analog clocks, whose arms move only in "discrete" steps.

Analog and digital computers can be better distinguished by the systems of tokens—or codes—they manipulate as inputs, outputs, and intermediate results. For present purposes, codes are identified functionally, by reference to the functional difference that tokens of different types make to the behavior of the mechanism.

Digital codes have tokens of only finitely many types, where each type of token has an unambiguously distinguishable effect on the mechanism that manipulates it and different tokens can be concatenated into strings. Moreover, the functional significance of each token depends both on its effect on the mechanism and on its place within the string of which it is part. If the physical world is continuous, any token of a digital code is part of a continuum of possible values of some physical variable, but computing mechanisms that employ digital codes are set up so that only certain values along that continuum are likely to occur at the functionally relevant times, and the mechanism responds to everything lying within relevant value intervals in the same functionally relevant way. Typically each token of a digital code is one among finitely many (usually two) stable states of certain components of the mechanism. All modern computers employ binary digits (bits) as tokens, represented as 0 and 1.

Analog codes, by contrast, have tokens of uncountably many types, whose effect on the mechanism can only be distinguished up to some level of approximation.[24] Moreover, their functional significance does not depend on their concatenation into strings but only on their effect on the mechanism. Analog codes can be used as inputs and outputs of computations, but not to encode

[24] There can be analog codes with only finitely many or countably many types of tokens, but they are not very useful for analog computation.

instructions. This is because instructions, in order to be effective, must be unambiguously distinguished from other instructions, which could not be done if they were represented by an analog token.

A *digital primitive computing component*, such as a logic gate, is a primitive computing component that takes tokens of a digital code as inputs and yield other tokens of a digital code as output. A *digital computer* is a computer whose primitive computing mechanisms, input devices, output devices, and other components are designed to hold and manipulate tokens of a digital code.

An *analog primitive computing component* is a primitive computing component that takes tokens of an analog code as inputs and yield other tokens of an analog code as output. An *analog computer* is a computer whose processing units, input devices, and output devices receive, process, and output analog tokens. Analog computers may well contain digital computing components, especially in their control units.

Digital and analog computers are more similar than they are often portrayed. First, as was already pointed out, in so far as analog and digital computers receive and execute instructions, they must do so by the same (digital) mechanisms. More generally, in order for the computation to be performed correctly, the operations performed by the primitive computing components must occur in the right order. This requires a clock for synchronizing the activities of the various components as well as a control mechanism that keeps track of what operations have been performed and what operations still need to be performed, and in what order. In both digital and analog computers, the functional properties of control units are similar. In both cases, they are built out of digital computing components, such as logic gates. This kind of precise control of the order of the operations performed by primitive computing components is one of the essential characteristics of computers and other complex computing mechanisms, which distinguishes them from other mechanism. In this respect, analog computers are not different from digital ones.

Typical analog computers were hard programmable; soft programmability was introduced by people who built digital computers. This is because soft programmability requires the use of a digital code. In order to build a stored-program (or even just soft programmable) analog computer, one needs to

use digital codes and technology, so the result is a computer that in the many important respects is digital and owes most of its interesting computational properties (such as being stored-program and soft-programmable) to its digital properties.

Analog computers are less flexible then digital ones. For instance, their inputs and outputs cannot literally take all the theoretically possible values, because of the physical limitations of their components (e.g., their components will break down if their voltage level goes above a certain threshold). The values of the inputs and outputs of analog computers must fall within certain limited ranges of values. Analog computers are also more difficult to use for precise computations than digital computers, for two reasons. First, their analog inputs and outputs can be distinguished from one another only up to a degree of approximation that depends on the precision of the measuring process, whereas digital inputs and outputs can always be unambiguously distinguished. Second, analog computational operations are affected by the interference of an indefinite number of physical conditions within the mechanism, which are usually called "noise," to the effect that their output is usually a worse approximation to the desired output than the input is to the desired input. These effects of noise accumulate during the computation, making it difficult to maintain a high level of computational precision. Digital operations, by contrast, are unaffected by this kind of "noise."

In the strict sense that is relevant here, computations performed by analog computers can indeed be approximated by digital ones to any desired degree of precision, provided that the digital computer has enough computational resources, such as memory and time. This is because analog codes can be approximated to any (finite) degree of precision by digital codes, and analog operations on analog codes can be replaced by digital operations on digital encodings of analog codes. From this it doesn't follow that the behavior of every physical system can be approximated to any desired degree of precision by digital computers.

10.2.6.6 Serial vs. Parallel

The issue of parallel vs. serial computation is often invoked in discussions of computationalism. A common claim is that the brain cannot be a "serial digital computer" because it is "parallel" (Churchland *et al*. 1990, p. 47; Churchland and Sejnowski 1992, p. 7). In evaluating this claim, we should keep in mind that digital computers can be parallel too, in several senses of the term. There are many distinctions to be made, and the discussion of different versions of computationalism can only gain from making at least the main ones explicit.

There is the question of whether in a computing mechanism only one computationally relevant event—for instance, the transmission of a signal between components—can occur during any clock cycle. In this sense, almost all complex computing mechanisms are parallel. For example, in most computing mechanisms, the existence of many communication lines between different components allows data to be transmitted in parallel. Even TMs do at least two things at a time, namely they perform an act on their tape and an update of their internal state.

A separate question is whether a computing mechanism can perform only one or more than one computational operation during any relevant time interval. This is the sense that connectionist computationalists appeal to when they say that the brain is "massively parallel." But again, most complex computing mechanisms, such as Boolean circuits, are parallel in this sense. Since most computers are made out of Boolean circuits and other computing components that are parallel in the same sense, they are parallel in this sense too. In this respect, then, there is no principled difference between ordinary computers and the computational formalisms employed by connectionist computationalists.

A third question is whether a computing mechanism executes one or more than one instruction at a time. There have been attempts to design parallel processors, which perform many operations at once by employing many executive units (such as ALUs) in parallel. Another way to achieve the same goal is to connect many processors together within one supercomputer. There are now in operation some supercomputers that include thousands of processors working in parallel. The difficulty in using these parallel computers consists of organizing computational tasks so that they can be modularized, i.e.

divided into sub-problems that can be solved independently by different processors. Instructions must be organized so that executing one instruction is not a prerequisite for executing other instructions in parallel to it, and does not interfere with their execution. This is sometimes possible and sometimes not, depending on what part of what computational problem is being solved. Some parts of some problems can be solved in parallel, but others can't, and some problems must be solved serially. Another difficulty is that in order to obtain the benefits of parallelism, typically the size of the hardware that must be employed in a computation grows (linearly) with the size of the input. This makes for prohibitively big (and expensive) hardware as soon as the problem instances to be solved by parallel computation become nontrivial in size. This is the notion of parallelism that is most relevant to computability theory. Strictly speaking, it applies only to computing mechanisms that execute instructions. Hence, it is irrelevant to discussing ordinary connectionist systems, which do not execute instructions.

In the connectionist literature, there is a persistent tendency to call neurons "processors" (e.g., Siegelmann 2003). Presumably the analogy is with a parallel computer that contains many processors, so that every neuron corresponds to one processor. This is misleading, because there is no useful sense in which a neuron can be said to execute instructions in the way that a computer processors can. In terms of their functional role within a computing mechanism, neurons compare more to logic gates than to the processors of digital computers; in fact, modeling neurons as logic gates was the basis for the first formulation of a computational theory of the brain (McCulloch and Pitts 1943). Nevertheless, it may be worth noticing that if the comparison between neurons and processors is taken seriously, then in this sense—the most important for computability theory—neurons are serial processors, because they do only one functionally relevant thing (either they fire or not) during any given time interval. Even if one looks at an entire connectionist network as a processor, one gets the same result; namely, that the network is a serial processor (it turns one input string into one output string). However, neither neurons nor ordinary connectionist systems are really comparable to computer processors in terms of their organization and function—they certainly don't execute instructions. So, to compare connectionist systems and computer processors in this respect is improper.

Finally, there is the question of whether a processor starts executing an instruction only after the end of the execution of the preceding instruction, or whether different instructions are executed in an overlapping way. This is possible when the processor is organized so that the different activities that are necessary to execute instructions (e.g., fetching an instruction from memory, executing the corresponding operation, and writing the result in memory) can all be performed at the same time on different instructions by the same processor. This kind of parallelism in the execution of instructions diminishes the global computation time for a given program, and it applies only to processors that execute instructions. It is a common feature of contemporary computers, where it is called *pipelining*.[25]

The above are parallel-serial distinctions that apply clearly to computing mechanisms, but the parallel-serial distinction is obviously broader than a distinction between modes of computing. Many things other than computations can be done in parallel. For example, instead of digging ten holes by yourself, you can get ten people to dig ten holes at the same time. Whether a process is serial or parallel is a different question from whether it is digital or analog (in various senses of the term), computational or non-computational.

10.3 Comparison With Previous Accounts of Computing Mechanisms

In this section, I will consider a number of proposals about computing mechanisms and discuss which desiderata they satisfy among those listed in section 10.1. The desiderata are: (1) Paradigmatic computing mechanisms compute, (2) Paradigmatic non-computing mechanisms don't compute, (3) Computation is observer-independent, (4) Computations can go wrong, (5) Some computing mechanisms are not computers, and (6) Program execution is explanatory. Each proposal discussed below rejects at least one of the two moves on which my account is based, and in that respect I've already argued in previous chapters that it should be rejected. Nevertheless, comparing different proposals *vis a vis* the six desiderata further highlights the advantages of the functional account.

[25] For more details on pipelining, see Patterson and Hennessy 1998, chap. 6.

10.3.1 Putnam

Putnam's proposal, and its historical importance, was analyzed in detail in Chapter 8. According to Putnam (1960, 1967b, 1988), a system is a computing mechanism if and only if there is a mapping between a computational description and a physical description of the system. By computational description, Putnam means a formal description of the kind used in computability theory, such as a Turing Machine or a finite state automaton. Putnam puts no constraints on how to find the mapping between the computational and the physical description, allowing any computationally identified state to map onto any physically identified state.

Putnam's account easily satisfies desideratum 1, because computational descriptions can be mapped onto paradigmatic computing mechanisms. But Putnam's account fails to satisfy all the remaining desiderata. As to desideratum 2, it fails because computational descriptions can also be (approximately) mapped onto paradigmatic non-computing mechanisms (as the popularity of computational models in many sciences testifies), which according to Putnam turns them into computing mechanisms. As to desideratum 3, it fails because given Putnam's liberalism about mapping, different observers can devise mappings between the same behavior of the same physical system and different computational descriptions. It is well known that Putnam's account entails that most physical systems implement most computations. This consequence of Putnam's proposal has been explicitly derived by Putnam (1988, pp. 95-96, 121-125) and Searle (1992, chap. 9).[26] As to desideratum 4, it fails because most behaviors of most physical systems, although computational under Putnam's proposal, are not subject to normative evaluation the way ordinary computations are. As to desideratum 5, it fails because the computational power of a physical system depends only on what computational description maps onto it, and given Putnam's proposal there is no fact of the matter as to which of the many computational

[26] Both Putnam and Searle take this to apply to computational descriptions in general, independently of Putnam's account of computing mechanisms. Both use this to argue that computationalism is a trivial thesis of no explanatory value. Here, I simply take this to be a reason against Putnam's account of computing mechanisms. The issue of the triviality of computationalism is addressed at some length in Chapter 8.

descriptions that map onto it is the right one. As to desideratum 6, it fails because in order for program execution to carry nontrivial explanatory force, program execution must be a specific kind of process that is exhibited by specific mechanisms. Since Putnam's account turns every behavior of every mechanism into the execution of a large number of programs, program execution has no explanatory force.

Some authors have attempted to improve on Putnam's proposal so that it satisfies desideratum 3, by introducing restrictions on the mapping between computational descriptions and physical descriptions of a system. According to Chalmers (1996) and Copeland (1996), for a physical description to count as an implementation of a computational description, the state transitions identified by that physical description must support counterfactuals of the following form: if the system were in state s_1, then it would change into state s_2; if the system were not in state s_1, then it would not change into state s_2 (where s_1 and s_2 are states identified by the computational description). Chalmers and Copeland argue that the state transitions employed by Putnam and Searle in their arguments do not obey this constraint, and because of this Chalmers and Copeland reject Putnam and Searle's conclusion that computation is observer-relative.[27] Along similar lines, Scheutz (1999) proposes to start with physical descriptions of a system in terms of electrical circuit theory, and abstract some of the physical properties away until a unique computational description of the system is left.

These authors also introduce some considerations pertaining to the inputs, outputs, and components of computing mechanisms. For example, Chalmers argues that the only appropriate models of cognitive mechanisms are automata with inputs and outputs, whose inputs and outputs must map onto the inputs and outputs of cognitive systems. Chalmers also adds that physical implementations of computational descriptions must decompose into parts and there must be a mapping between the states of those parts and suitably defined sub-states identified by the computational description. That is, the system must have a "fine-grained causal structure" that maps onto the computational structure identified

[27] Further criticisms to Putnam and Searle's arguments, similar to those of Copeland and Chalmers, can be found in Chrisley 1995.

by the computational description. These considerations could be seen as implicit steps towards the functional account of computing mechanisms that is here defended.

To the extent that these refinements of Putnam's account satisfy desideratum 3, they constitute an improvement over Putnam's account of computing mechanisms. There is no room here to analyze these proposals in detail to see if they solve the problem of observer-relativity of computation in a satisfactory manner, and fortunately there is no need to do so. This is because even if it is granted that these refined proposals satisfy desiderata 1 and 3, as they stand they still fail to satisfy the other desiderata. As to desideratum 2 (paradigmatic non-computing mechanisms don't compute), Chalmers and Scheutz explicitly point out that under their proposal, everything implements some computation or other (Chalmers 1996b, p. 331; Scheutz 1999, p. 191). This makes it hard to see how program execution, or computation in general, can play any explanatory role, so that these proposals fail to satisfy desideratum 6 (program execution is explanatory). Copeland's proposal does not appear to be different in this respect. With respect to desideratum 5 (some computing mechanisms are not computers), it seems that these proposals do not have the resources to differentiate between the computing powers of different mechanisms, because they use the same computational formalism to represent the behavior of every physical system.

10.3.2 Cummins[28]

According to Cummins (1977, 1983, 1989, esp. pp. 91-92), a system is a computing mechanism if and only if it has inputs and outputs that can be given a systematic semantic interpretation and the system's input-output behavior is the "execution" of a program that yields the interpretation of the system's outputs given an interpretation of the system's inputs. For Cummins, a process is the "execution" of a program if and only if it is made up of sub-processes each of which instantiates one of the steps described by the

[28] Accounts of computing mechanisms similar to Cummins's account have been offered by Dennett (1978a) and Haugeland (1978). I discuss Cummins's account because it is the most systematic and detailed in this family of accounts. Cummins's account is also conceptually similar to that of Churchland and Sejnowski (1992).

program. (This is a very different explication of program execution from the one advocated in section 10.2 above.)

It is hard to determine the degree to which Cummins's account satisfies our first two desiderata. Many paradigmatic examples of computing mechanisms have inputs and outputs that can be given systematic interpretations, and their processes can be divided into sub-processes each of which instantiates the steps of a relevant program. Cummins's account counts all of those as computing mechanisms, which goes in the direction of satisfying desideratum 1. But there are two kinds of exceptions. One kind includes TMs like the one described by J. Buntrock and H. Marxen in 1989 (cited by Wells 1998), which performs 23,554,764 steps before halting when it's started on a blank tape. It's unclear what if any interpretation can be assigned to this machine. The other kind of exception includes many connectionist systems, whose processes are not analyzable into sub-processes corresponding to steps defined by a program in any obvious way. In fact, connectionist computing mechanisms have been used as examples that not all computation is algorithmic (Shagrir 1997). In order to include these exceptions within the class of computing mechanisms, thereby fully satisfying desideratum 1, Cummins might liberalize the constraints on how to assign interpretations and algorithmic descriptions to systems.[29] But then it becomes hard to rule any system out of the class of computing mechanisms, so desideratum 2 fails to be satisfied. Cummins says he doesn't know how to satisfy desideratum 3 (computation is observer-independent),[30] but his account can probably be combined with the kind of patch offered to Putnam's account by Chalmers and Copeland. Cummins satisfies desideratum 6 because for him, executing a program is the same as satisfying a certain computational description, and satisfying a certain computational description is the same as being functionally analyzed. According to Cummins, functional analysis provides a distinct mode of explanation, i.e. functional explanation (Cummins 1975, 1983).

[29] This seems to be implicitly done by Cummins when he attempts to apply his account of computing mechanisms to connectionist systems (Cummins and Schwartz 1991).

[30] Cf.:

> Reflections such as these make direct interpretation seem like a rather subjective and relativistic affair. Nevertheless, it appears to me to be absolutely central to the notion of function instantiation (and hence computation), so I'm simply going to assume it, leaving to someone else the task of filling this hole or widening it enough to dink the ship (Cummins 1989, p. 105).

Hence, program execution is explanatory in the sense that it provides a functional explanation. Cummins has not addressed desideratum 4 (computations can go wrong), and his account fails to satisfy desideratum 5 (some computing mechanisms are not computers), because it turns every computing mechanism into a program-executing mechanism, thereby trivializing the difference between different computing powers of different computing mechanisms.

10.3.3 Fodor[31]

According to Fodor (1975, 1986b, 1998, pp. 10-11), a system is a computing mechanism if and only if it operates on symbols by responding to internal representations of rules, and the system's operations "respect" the semantic content of the symbols. For Fodor, a symbol is a semantically identified token that can be manipulated by virtue of its formal (non-semantic) properties. At least in the case of natural systems like brains, for Fodor the content of symbols is a natural property that can be discovered in a way that is observer-independent (Fodor 1990, 1998).

Fodor's account improves over both Putnam's and Cummins's accounts. It satisfies desiderata 2, 3, and 6 well. As to 2 (paradigmatic non-computing mechanisms don't compute), non-computing mechanisms are ruled out by their lack of internal representations of rules. As to 3 (computation is observer-independent), computation is made observer-independent by Fodor's naturalistic theory of content, which underwrites his notion of computation. As to 6 (program execution is explanatory), Fodor's account explicates the notion of program execution in terms of the system's internal representation of rules together with the fact that the system's processes are caused by those internal representations. However, Fodor's account still has problems with desideratum 1, and it satisfies neither

[31] An account similar to Fodor's is that of Pylyshyn (1984), who, like Fodor, identifies computational states, inputs, and outputs, by their semantic properties. The main difference between the two is Pylyshyn's notion of functional architecture. A suitably de-semanticized version of Pylyshyn's notion of functional architecture can be seen as the application of the functional account of computing mechanisms to stored-program computers. (However, there remains an important difference between Pylyshyn's functional architecture and the functional account of computing mechanisms: Pylyshyn speaks interchangeably of the hardware and of the "virtual architecture" of a computing mechanism as being their functional architecture, whereas the functional account draws a sharp distinction between the two and accounts for the latter in terms of the former.) Because of Pylyshyn's restrictive use of his notion of functional architecture and his reliance on semantic properties to identify computational states, though, his account does not seem to improve on Fodor's account in satisfying our desiderata.

4 nor 5. As to 1 (paradigmatic computing mechanisms compute), computing mechanisms that are not obviously interpretable as operating on contentful symbols cannot be accommodated within Fodor's account. Perhaps more seriously, there are plenty of paradigmatic computing mechanisms, starting with ordinary (non-universal) TMs, which do not possess internal representations of rules. Fodor's account legislates that these mechanisms don't compute. As to 4 (computations can go wrong), perhaps Fodor's account has the resources to satisfy it, but Fodor has been silent on this. Desideratum 5 (some computing mechanisms are not computers) remains unsatisfied, because for Fodor all genuinely computing mechanisms are essentially stored-program computers, which leaves little room for differentiating between their computing power and the computing power of other mechanisms that are ordinarily seen as computing.

None of the existing accounts satisfies all the six desiderata. None of them satisfies desiderata 4 and 5. I will now show how the functional account satisfies the six desiderata.

10.3.4 The Functional Account and the Six Desiderata

The functional account of computing mechanisms satisfies the six desiderata in a natural way.

1. *Paradigmatic computing mechanisms compute*. All paradigmatic examples of computing mechanisms, such as digital computers, calculators, Turing Machines, finite state automata, and humans who calculate, take strings as inputs and outputs and have the proper function (goal in the case of humans) of generating certain output strings from certain input strings. According to the functional account, this is necessary and sufficient for them to be ascribed computations. Accordingly, the functional account properly counts all paradigmatic examples of computing mechanisms as such.

2. *Paradigmatic non-computing mechanisms don't compute*. The functional account of computing mechanisms explains why paradigmatic examples of non-computing mechanisms don't compute by invoking their functional analysis, which is different from that of computing mechanisms. Planetary systems, hurricanes, stomachs, etc., even when they do receive inputs and yield outputs in a

nontrivial sense, don't receive the right kind of discrete inputs concatenated into strings and don't have the right kinds of components (e.g., input devices, output devices, memory, and processor) functionally organized in the appropriate way. They instantiate the input-output function that represents their behavior, but they don't compute it.[32]

3. *Computation is observer-independent*. The functional account of computing mechanisms does not rely on "interpreting" systems in ways that are potentially observer-relative. Either something has a certain functional organization or it doesn't, depending on how its components functionally interact with one another. For example, either something is a memory register or not, an arithmetic-logic unit or not, etc., depending on what function it fulfills within its containing mechanism. The functional analysis of a computing mechanism is no less observer-independent than any other functional analysis in biology or engineering. It can be discovered in biological organisms or in artifacts independently of the observer.

4. *Computations can go wrong*. The functional account of computing mechanisms explains what it means for a computing mechanism to miscompute, or make a mistake in a computation. Miscomputations are a kind of malfunction, i.e. events in which a functional system fails to fulfill its function. In the case of computing mechanisms, whose function is to compute certain functions, functional failure results in a mistake in the computation. There are many kinds of miscomputations. The most obvious kind is hardware failure, i.e. failure of a hardware component to perform its proper function (specified by the functional analysis of the mechanism). Hardware failure may be due to the failure of a computing component, such as a logic gate, or of a non-computing component, such as a clock. Another kind of miscomputation may be due to a mistake in computer design, so that the designed mechanism does not in fact compute the function it was intended to compute. Again, the design mistake may be due to a computing component that does not compute what it is intended to compute or to a non-computing component that does not fulfill its function (e.g., a clock with a too short cycle time). Another kind of miscomputation may be due to a programming error, whereby instructions are either mistakenly written

[32] Non-computing mechanisms may be said to be "computational" in some looser senses than genuine computing mechanisms. I offered a taxonomy of those looser senses in Chapter 8.

(and hence cannot be executed) or do not play their intended role within the program. Yet another kind may be due to the accumulation of round-off errors in the finite precision arithmetic that computer processors execute. Finally, miscomputations may be due to faulty hardware-software interaction. A familiar example of this last type occurs when the execution of a program requires more memory than the computer physically has available. When there is no more memory available, the computer "freezes" without being able to complete the computation.[33]

5. *Some computing mechanisms are not computers*. The functional account of computing mechanisms explains why only some computing mechanisms are computers properly so called: only genuine computers are programmable. Computing mechanisms that are not programmable deserve other names, such as calculators, arithmetic-logic units, etc., and can still be differentiated from one another based on their computing power, which is determined by their functional organization.

6. *Program execution is explanatory*. The functional account of computing mechanisms explicates the notion of explanation by program execution. It is a special kind of functional explanation that relies on the special kind of functional analysis that applies to soft programmable computers. Program execution is a process by which a certain part of the mechanism, the program, affects a certain other part of the mechanism, the processor, so that the processor performs appropriate operations on a certain other part of the mechanism, the data. A mechanism must be functionally analyzable in this way to be subject to explanation by program execution. Explanation by program execution is the most interesting genus of the species of explanation by appeal to the computations performed by a mechanism. Appealing to the computations performed by a mechanism is explanatory in so far as the mechanism is a computing mechanism, i.e. a mechanism subject to the kind of functional analysis described in this chapter.

[33] For an early discussion of several kinds of computing mistakes by computing mechanisms, see Goldstine and von Neumann 1946. For a modern treatment, see Patterson and Hennessy 1998.

10.4 An Application: Are Turing Machines Computers?

Turing Machines (TMs) can be seen and studied mathematically as lists of instructions (TM programs), without any mechanical counterpart. But they can also be seen as a type of computing mechanism, which is made out of a tape divided into squares and a unit that moves along the tape, acts on it, and is in one out of a finite number of internal states (see Chapter 1 for more details). The tape and the active unit are the components of TMs, whose functions are, respectively, storing tokens and performing operations on the tokens. The active unit of TMs is typically not further analyzed, but in principle it could be functionally analyzed as a Boolean Circuit plus a memory register. When they are seen as mechanisms, TMs fall naturally under the functional account of computing mechanisms. They are uniquely identified by their functional analysis, which includes their list of instructions.

There are two importantly different classes or TMs: ordinary TMs and universal TMs. Ordinary TMs are not programmable and *a fortiori* not universal. They are "hardwired" to compute one and only one function, as specified by the list of instructions that uniquely identifies each TM.

Universal TMs, on the other hand, are programmable (and of course universal), because they respond to a portion of the tokens written on their tape by computing the function that would be computed by the TM encoded by those tokens. Nevertheless, universal TMs are *not* stored-program computers, because their architecture has no genuine memory component. The programs for universal TMs are stored on the same tape that contains the input and the output. In this respect, they are analogous to early punched cards computers. The tape can be considered the input and output device, and with some semantic stretch, a memory. But since there is no distinction between input device, output device, and memory, a universal TM should not be considered a stored-program computer properly so called, for the same reason that punched cards machines aren't.

Even after the invention of TMs, it was still an important conceptual advance to design computers that could store their instructions in their internal memory component. So, it is anachronistic to attribute the idea of the stored-program computer to Turing (as done, e.g., by Aspray 1990, Copeland 2000a). This

exemplifies how a proper understanding of computing mechanisms, based on functional analysis, can shed light on the history of computing.

10.5 A Taxonomy of Computationalist Theses

The functional account of computing mechanisms can be used to formulate a taxonomy of computationalist theses about the brain, in order of increasing strength. The theses range from the commonsensical and uncontroversial thesis that the brain processes information to the strong thesis that it is a programmable, stored-program, universal computer. Each thesis presupposes the truth of the preceding one and adds a further assumption to it:

1) The brain is a collection of interconnected neurons that deal with information and control (in an intuitive, formally undefined sense).
2) Networks of neurons are computing mechanisms.
3) Networks of neurons are Boolean circuits or finite state automata (McCulloch and Pitts 1943).
4) The brain is a programmable computer (Devitt and Sterelny 1990).
5) The brain is a programmable, stored-program computer (Fodor 1975).
6) The brain is a universal computer (Newell and Simon 1976).[34]

Thesis (1) is sufficiently weak and generic that it entails nothing controversial about the properties of neural mechanisms. Even a non-cognitivist (e.g., a behaviorist) should agree on this. I mention it here to distinguish it from nontrivial computationalist theses and leave it out of the discussion.

Historically, thesis (3) is the first formulation of computationalism. Its weakening leads to (2), which includes modern connectionist computationalism. Its strengthening leads to what is often called classical (i.e., non-connectionist) computationalism. Classical computationalism is usually identified

[34] Strictly speaking, (6) does not presuppose (5). For instance, universal TMs are not stored-program. However, in practice all supporters of (6) also endorse (5), for the good reason that there is no evidence of a storage system in the environment—analogous to the tape of TMs—that would store the putative programs executed by brains.

with (5) or (6), but sometimes (4) has been discussed as an option. The above taxonomy allows us to see that different formulations of classical computationalism are not equivalent to one another, and that they vary in the strength of their assumptions about neural mechanisms.

The above list includes only the theses that have figured prominently in the computationalist literature. This list is by no means exhaustive of possible computationalist theses. First, notice that (3) can be divided into a relatively weak thesis—according to which the brain is a collection of Boolean circuits—and a much stronger one—according to which the brain is a collection of finite state automata. The functional account of computing mechanisms shows how to construct theses that are intermediate in strength between these two. For instance, one could hypothesize that the brain is a collection of components that can execute complex pseudo-algorithms, like the multiplication and division components of modern computers.

Another possible version of computationalism is the hypothesis that the brain is a calculator. This possibility is intriguing because no one has ever proposed it even though calculators are *bona fide* computing mechanisms. The functional account of computing mechanisms sheds light on this fact. Among other limitations calculators have, the computational repertoire of calculators is fixed. There is no interesting sense in which they can learn to compute new things or acquire new computational behavior. One important factor that attracted people to computationalism is the flexibility and power of computers, flexibility and power that calculators lack. Because of this, it is not surprising that no one has proposed that brains are calculators.

The kinds of computers that are most strongly associated with computationalism are programmable, stored-program, universal computers, because they have the exciting property that given the appropriate program stored inside the machine, they can automatically compute any computable function, and they can modify their programs automatically (which gives them an important kind of learning ability).

10.6 Questions of Hardware

It is often said that even if the brain is a computing mechanism, it need not have a von Neumann architecture (Pylyshyn 1984, Churchland and Sejnowski 1992). In these discussions, "von Neumann architecture" is used as a generic term for the functional organization of ordinary digital computers. This claim is used to discount apparent dissimilarities between the functional organization of brains and that of ordinary digital computers as irrelevant to computationalism. The idea is that brains may compute by means other than those exploited by modern digital computers.

It is true that not all computing mechanisms need have a von Neumann architecture. For example, Turing Machines don't. But this does not eliminate the constraints that different versions of computationalism put on the functional organization of the brain, if the brain is to perform the relevant kinds of computations. In the current discussion, I am intentionally avoiding the term "von Neumann architecture" because it is so generic that it obscures the many issues of functional organization that are relevant to the design and computing power of computing mechanisms. The present account allows us to increase the precision of our claims about computer and brain architecture, avoiding the generic term "von Neumann architecture" and focusing on various functional properties of computing mechanisms (and hence on what their computing power is).

If the brain is expected to be a programmable, stored-program, universal computer, as it is by some versions of computationalism, it must contain programs as well as components that store and execute the programs. More generally, any kind of computation, even the most trivial transformation of one token into another (as performed by a NOT gate) requires appropriate hardware. So every nontrivial computationalist thesis, depending on the computation power it ascribes to the brain, constrains the functional properties that brains must exhibit if they are to perform the relevant computations. The following are general questions about neural hardware that apply to some or all computationalist theses about the brain:

1. What are the tokens manipulated in the neural computation, and what are their types?

2. What are the elementary computational operations on neural tokens, and what are the components that perform them?
3. How are the tokens concatenated to one another, so that strings of them can be identified as inputs, internal states, and outputs of neural mechanisms and nontrivial computations from input strings to output strings can be ascribed to neural mechanisms?
4. What are the compositional rules between elementary operations, and the corresponding ways to connect the components, such that complex operations can be formed out of elementary ones and performed by the mechanism?
5. If the system stores programs or even just data for the computations, what are the memory cells and registers and how do they work?
6. What are the control units that determine which operations are executed at any given time and how do they work? This question is particularly pressing if there has to be execution of programs, because the required kind of control unit is particularly sophisticated and needs to correctly coordinate its behavior with the components that store the programs.

When McCulloch and Pitts (1943) initially formulated computationalism, they had answers to the relevant versions of the above questions. In answer to (1), they thought that the presence and the absence of a neural spike are the two types of tokens on which neural computations are defined. In answer to (2), they appealed to Boolean operations and claimed that they were performed by neurons. In answer to (3) and (4), they relied on a formalism they largely drew from Carnap, which is equivalent to a mixture of Boolean algebra and FSA, in combination with the beginning of what became the technique of logic design. In answer to (5), McCulloch hypothesized that there were closed loops of neural activity, which acted as memory cells. In answer to (6), they largely appealed to the innate wiring of the brain.

When von Neumann tried to formulate his own version of computationalism (von Neumann 1958), he also tried to answer at least the first two of the above questions. In answer to (1), he thought that the firing rates of neurons were the tokens' types. In answer to (2), he thought the elementary

operations were arithmetical and logical operations on these firing rates. Although von Neumann's answers take into account the functional significance of neuronal spikes as it is understood by modern neurophysiologists, Von Neumann did not have answers to questions 3 to 6, and he explicitly said that he did not know how the brain could possibly achieve the degree of computational precision that he thought it needed under his assumptions about its computational organization.

Today's computationalists no longer believe McCulloch's or von Neumann's versions of computationalism. But if computationalism is to remain a substantive, empirical hypothesis about the brain, these questions need to find convincing answers. If they don't, it may be time to abandon computationalism in favor of other functional explanations of neural mechanisms.

10.7 Conclusion

The functional account of computing mechanisms offered in this chapter is a viable account of what it means for a mechanism to compute. It satisfies the desiderata of an account of computing mechanisms. It allows us to formulate the question of whether a mechanism is a computer as an empirical hypothesis, to be decided by looking at the functional organization of the mechanism. It allows us to formulate in a clear way a useful taxonomy of kinds of computing mechanisms and compare their computing power. I submit that this can be used profitably in discussing computationalism, and that it constitutes an improvement over existing accounts of computing mechanisms that is valuable in its own right.

BIBLIOGRAPHY

Abbott, L. and T. J. Sejnowski (1998). *Neural Codes and Distributed Representations: Foundations of Neural Computation*. Cambridge, MA, MIT Press.

Abraham, T. (2001a). Styles of Visualization: Representing Neurons in the 1930s, presented at the Canadian Society for the History and Philosophy of Science.

Abraham, T. (2001b). Taming Organized Complexity: Logic, Neural Networks, and Theoretical Biology at the University of Chicago, 1934-1943, presented at the Dibner Institute Colloquium, 27 March 2001.

Abraham, T. (2002). "(Physio)logical Circuits: The Intellectual Origins of the McCulloch-Pitts Neural Networks." *Journal of the History of the Behavioral Sciences* **38**(1): 3-25.

Adrian, E. D. (1926). "The Impulses Produced by Sensory Nerve Endings." *Journal of Physiology* **61**: 49-72.

Adrian, E. D. and Y. Zotterman (1926). "The Impulses Produced by Sensory Nerve Endings, Part 2: The Response of a Single End-Organ." *Journal of Physiology* **61**: 151-171.

Anderson, J. A. and E. Rosenfeld, Eds. (1998). *Talking Nets: An Oral History of Neural Networks*. Cambridge, MA, MIT Press.

Arbib, M. A. (1989). Comments on "A Logical Calculus of the Ideas Immanent in Nervous Activity". *Collected Works of Warren S. McCulloch*. R. McCulloch, Ed. Salinas, CA, Intersystems**:** 341-342.

Arbib, M. A. (2000). "Warren McCulloch's Search for the Logic of the Nervous System." *Perspectives in Biology and Medicine* **43**(2): 193-216.

Aspray, W. (1985). "The Scientific Conceptualization of Information: A Survey." *Annals of the History of Computing* **7**(2): 117-140.

Aspray, W. (1990a). *John von Neumann and the Origins of Modern Computing*. Cambridge, MA, MIT Press.

Aspray, W., Ed. (1990b). *Computing Before Computers*. Ames, Iowa, Iowa University Press.

Aspray, W. and A. W. Burks, Eds. (1987). *Papers of John von Neumann on Computing and Computer Theory*. Cambridge, MA, MIT Press.

Attneave, F. (1961). In Defense of Homunculi. *Sensory Communication*. W. Rosenblith, Ed. Cambridge, MA, MIT Press**:** 777-782.

Barabási, A.-L. (2002). *Linked: The New Science of Networks*. Cambridge, MA, Perseus.

Bechtel, W. (2001). Cognitive Neuroscience: Relating Neural Mechanisms and Cognition. *Theory and Method in the Neurosciences*. P. Machamer, R. Grush and P. McLaughlin, Eds. Pittsburgh, PA, University of Pittsburgh Press**:** 81-111.

Bechtel, W., A. Abrahamsen, et al. (1998). The Life of Cognitive Science. *A Companion to Cognitive Science*. W. Bechtel and G. Graham, Eds. Malden, MA, Blackwell**:** 1-104.

Bechtel, W. and J. Mundale (1999). "Multiple Realizability Revisited: Linking Cognitive and Neural States." *Philosophy of Science* **66**: 175-207.

Bechtel, W. and R. C. Richardson (1993). *Discovering Complexity: Decomposition and Localization as Scientific Research Strategies*. Princeton, Princeton University Press.

Benacerraf, P. and H. Putnam (1964). *Philosophy of Mathematics*. Englewood Cliffs, NJ, Prentice-Hall.

Bennett, J. F. (1976). *Linguistic Behaviour*. Cambridge, Cambridge University Press.

Bickle, J. (1998a). *Psychoneural Reduction: The New Wave*. Cambridge, MA, MIT Press.

Bickle, J. (1998b). Multiple Realizability. *Stanford Encyclopedia of Philosophy*. E. N. Zalta, Ed. ISSN 1095-5054, http://plato.stanford.edu.

Block, N. (1978). Troubles with Functionalism. *Perception and Cognition: Issues in the Foundations of Psychology*. C. W. Savage, Ed. Minneapolis, University of Minnesota Press. **6:** 261-325.
Block, N. (1986). Advertisement for a Semantics for Psychology. *Midwest Studies in Philosophy X: Studies in the Philosophy of Mind*. P. French, T. E. Uehling, jr. and H. K. Wettstein, Eds. Minneapolis, University of Minnesota Press.
Block, N. and J. A. Fodor (1972). "What Psychological States Are Not." *Philosophical Review* **81**(2): 159-181.
Blum, L., F. Cucker, et al. (1998). *Complexity and Real Computation*. New York, Springer.
Boden, M. (1991). Horses of a Different Color? *Philosophy and Connectionist Theory*. W. Ramsey, S. P. Stich and D. E. Rumelhart, Eds. Hillsdale, LEA**:** 3-19.
Brandom, R. B. (1994). *Making it Explicit: Reasoning, Representing, and Discursive Commitment*. Cambridge, MA, Harvard University Press.
Bringsjord, S. (1998). "The Narrational Case Against Church's Thesis." *Jounal of Philosophy*.
Brooks, R. A. (1997). Intelligence without Representation. *Mind Design II*. J. Haugeland, Ed. Cambridge, MA, MIT Press**:** 395-420.
Brouwer, L. E. J. (1975). *Collected Works, Vol. 1*. Amsterdam, North-Holland.
Burge, T. (1986). "Individualism and Psychology." *Philosophical Review* **95**: 3-45.
Burks, A. W., J. von Neumann, et al. (1946). Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Princeton, Institute for Advanced Studies.
Butler, K. (1996). "Content, Computation, and Individuation in Vision Theory." *Analysis* **56**: 146-154.
Carnap, R. (1928). *Der Logische Aufbau der Welt*. Leipzig, Verlag.
Carnap, R. (1932-3/1959). Psychology in Physical Language. *Logical Positivism*. A. J. Ayer, Ed. Glencoe, Ill., Free Press**:** 165-198.
Chalmers, D. J. (1996a). *The Conscious Mind: In Search of a Fundamental Theory*. Oxford, Oxford University Press.
Chalmers, D. J. (1996b). "Does a Rock Implement Every Finite-State Automaton?" *Synthese* **108**: 310-333.
Chomsky, N. (1957). *Syntactic Structures*. The Hague, Mouton.
Chomsky, N. (1965). *Aspects of a Theory of Syntax*. Cambridge, MA, MIT Press.
Chrisley, R. L. (1995). "Why Everything Doesn't Realize Every Computation." *Minds and Machines* **4**: 403-430.
Church, A. (1932). "A Set of Postulates for the Foundation of Logic." *Annals of Mathematics* **33**: 346-366.
Church, A. (1936). "An Unsolvable Problem in Elementary Number Theory." *The American Journal of Mathematics* **58**: 345-363.
Church, A. (1937). "Review of Turing (1936-7)." *Journal of Symbolic Logic* **2**: 42-43.
Church, A. (1940). "On the Concept of a Random Sequence." *American Mathematical Society Bulletin* **46**: 130-135.
Church, A. (1956). *Introduction to Mathematical Logic*. Princeton, Princeton University Press.
Churchland, P. M. (1979). *Scientific Realism and the Plasticity of Mind*. Cambridge, Cambridge University Press.
Churchland, P. M. (1982). "Is 'Thinker' a Natural Kind?" *Dialogue* **21**: 233-238.
Churchland, P. M. and P. S. Churchland (1990). "Could a Machine Think?" *Scientific American* **CCLXII**: 26-31.
Churchland, P. S., C. Koch, et al. (1990). What is Computational Neuroscience? *Computational Neuroscience*. E. L. Schwartz, Ed. Cambridge, MA, MIT Press**:** 46-55.
Churchland, P. S. and T. J. Sejnowski (1992). *The Computational Brain*. Cambridge, MA, MIT Press.
Clark, A. (1997). *Being There*. Cambridge, MA, MIT Press.
Cleland, C. E. (1993). "Is the Church-Turing Thesis True?" *Minds and Machines* **3**: 283-312.
Cleland, C. E. (1995). "Effective Procedures and Computable Functions." *Minds and Machines* **5**: 9-23.
Cleland, C. E. (2001). "Recipes, Algorithms, and Programs." *Minds and Machines* **11**: 219-237.

Cleland, C. E. (2002). "On Effective Procedures." *Minds and Machines* **12**.
Copeland, B. J. (1996a). The Church-Turing Thesis. *Stanford Encyclopedia of Philosophy*. E. N. Zalta, Ed., URL = <http://plato.stanford.edu>.
Copeland, B. J. (1996b). "What is Computation?" *Synthese* **108**: 224-359.
Copeland, B. J. (2000a). "Narrow Versus Wide Mechanism: Including a Re-Examination of Turing's Views on the Mind-Machine Issue." *The Journal of Philosophy* **XCVI**(1): 5-32.
Copeland, B. J. (2000b). The Modern History of Computing. *Stanford Encyclopedia of Philosophy*. E. N. Zalta, Ed. ISSN 1095-5054, URL = <http://plato.stanford.edu>.
Copeland, B. J. (2002a). "Hypercomputation." *Minds and Machines* **12**: 461-502.
Copeland, B. J. (2002b). "Accelerating Turing Machines." *Minds and Machines* **12**: 281-301.
Copeland, B. J. and D. Proudfoot (1996). "On Alan Turing's Anticipation of Connectionism." *Synthese* **108**: 361-377.
Cotogno, P. (2003). "Hypercomputation and the Physical Church-Turing Thesis." *British Journal for the Philosophy of Science* **54**: 181-223.
Crane, T. (1990). "The Language of Thought: No Syntax Without Semantics." *Mind and Language* **5**(3): 187-212.
Craver, C. (2001a). "Role Functions, Mechanisms, and Hierarchy." *Philosophy of Science* **68**(March 2001): 53-74.
Craver, C. (2001b). Remembering Mechanisms: The Long-Term Discovery of Long-Term Potentiation, presented at the 2001 Meeting of the International Society for History, Philosophy, and Social Studies of Biology (ISHPSSB).
Craver, C. and L. Darden (2001). Discovering Mechanisms in Neurobiology. *Theory and Method in the Neurosciences*. P. Machamer, R. Grush and P. McLaughlin, Eds. Pittsburgh, PA, University of Pittsburgh Press**:** 112-137.
Crevier, D. (1993). *AI: The Tumultuous History of the Search for Artificial Intelligence*. New York, HarperCollins.
Cummins, R. (1975). "Functional Analysis." *Journal of Philosophy* **72**(20): 741-765.
Cummins, R. (1977). "Programs in the Explanation of Behavior." *Philosophy of Science* **44**: 269-287.
Cummins, R. (1983). *The Nature of Psychological Explanation*. Cambridge, MA, MIT Press.
Cummins, R. (1989). *Meaning and Mental Representation*. Cambridge, MA, MIT Press.
Cummins, R. and G. Schwarz (1991). Connectionism, Computation, and Cognition. *Connectionism and the Philosophy of Mind*. T. Horgan and J. Tienson, Eds. Dordrecht, Kluwer**:** 60-73.
Davies, E. B. (2001). "Building Infinite Machines." *British Journal for the Philosophy of Science* **52**(4): 671-682.
Davis, M. (1958). *Computability and Unsolvability*. New York, McGraw-Hill.
Davis, M. (1965). *The Undecidable*. Ewlett, NY, Raven.
Davis, M. (1982). "Why Gödel Didn't Have Church's Thesis." *Information and Control* **54**: 3-24.
Davis, M. (2000). *The Universal Computer: The Road from Leibniz to Turing*. New York, Norton.
Davis, M., R. Sigal, et al. (1994). *Computability, Complexity, and Languages*. Boston, Academic.
Dayan, P. and L. F. Abbott (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA, MIT Press.
Dean, W. (2002). What Algorithms Could Not Be. Pittsburgh, PA, presented at the Computing and Philosophy Conference.
Dennett, D. C. (1969). *Content and Consciousness*. London, Routledge and Kegan Paul.
Dennett, D. C. (1971). "Intentional Systems." *Journal of Philosophy* **LXVIII**(4): 87-106.
Dennett, D. C. (1975). "Why the Law of Effect Will Not Go Away." *Journal of the Theory of Social Behavior* **5**: 169-187.
Dennett, D. C. (1978a). *Brainstorms*. Cambridge, MA, MIT Press.
Dennett, D. C. (1978b). The Abilities of Men and Machines. *Brainstorms*. D. C. Dennett, Ed. Cambridge, MA, MIT Press**:** 256-266.

Dennett, D. C. (1978c). Skinner Skinned. *Brainstorms*. D. C. Dennett, Ed. Cambridge, MA, MIT Press**:** 53-70.
Dennett, D. C. (1978d). Artificial Intelligence as Philosophy and as Psychology. *Brainstorms*. D. C. Dennett, Ed. Cambridge, MA, MIT Press**:** 109-126.
Dennett, D. C. (1987). *The Intentional Stance*. Cambridge, MA, MIT Press.
Deutsch, D. (1985). "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer." *Proceedings of the Royal Society of London A* **400**: 97-117.
Deutsch, J. A. (1960). *The Structural Basis of Behavior*. Chicago, University of Chicago Press.
Dietrich, E. (1989). "Semantics and the Computational Paradigm in Cognitive Psychology." *Synthese* **79**: 119-141.
Dretske, F. I. (1981). *Knowledge and the Flow of Information*. Cambridge, MA, MIT Press.
Dretske, F. I. (1986). Misrepresentation. *Belief: Form, Content, and Function*. R. Bogdan, Ed. New York, Oxford University Press**:** 17-36.
Dreyfus, H. L. (1979). *What Computers Can't Do*. New York, Harper & Row.
Dupuy, J. (2000). *The Mechanization of Mind: On the Origins of Cognitive Science*. Princeton, Princeton University Press.
Earman, J. and J. Norton (1993). "Forever is a Day: Supertasks in Pitowski and Malament-Hogarth Spacetimes." *Philosophy of Science* **60**: 22-42.
Egan, F. (1989). "Discussion: What's Wrong with the Syntactic Theory of Mind." *Philosophy of Science* **56**: 664-674.
Egan, F. (1992). "Individualism, Computation, and Perceptual Content." *Mind* **101**(403): 443-459.
Egan, F. (1995). "Computation and Content." *Philosophical Review* **104**: 181-203.
Egan, F. (1999). "In Defence of Narrow Mindedness." *Mind and Language* **14**(2): 177-194.
Enç, B. (1983). "In Defense of the Identity Theory." *Journal of Philosophy* **80**: 279-298.
Fancher, R. E. (1973). *Psychoanalytic Psychology: The Development of Freud's Thought*. New York, Norton.
Feigl, H. (1958). The 'Mental' and the 'Physical'. *Minnesota Studies in the Philosophy of Science, II*. H. Feigl, M. Scriven and G. Maxwell, Eds. Minneapolis, University of Minnesota Press**:** 370-497.
Field, H. (1978). "Mental Representation." *Erkenntnis* **13**: 9-61.
Fitch, F. (1944). "Review of McCulloch and Pitts 1943." *Journal of Symbolic Logic*.
Floridi, L. (1999). *Philosophy and Computing: An Introduction*. London, Routledge.
Fodor, J. A. (1965). Explanations in Psychology. *Philosophy in America*. M. Black, Ed. London, Routledge and Kegan Paul.
Fodor, J. A. (1968a). *Psychological Explanation*. New York, Random House.
Fodor, J. A. (1968b). "The Appeal to Tacit Knowledge in Psychological Explanation." *Journal of Philosophy* **65**: 627-640.
Fodor, J. A. (1975). *The Language of Thought*. Cambridge, MA, Harvard University Press.
Fodor, J. A. (1978). "Tom Swift and His Procedural Grandmother." *Cognition* **6**.
Fodor, J. A. (1980). "Methodological Solipsism Considered as a Research Strategy in Cognitive Psychology." *Behavioral and Brain Sciences* **3**(1).
Fodor, J. A. (1981). *Representations*. Cambridge, MA, MIT Press.
Fodor, J. A. (1987). *Psychosemantics*. Cambridge, MA, MIT Press.
Fodor, J. A. (1990). *A Theory of Content and Other Essays*. Cambridge, MA, MIT Press.
Fodor, J. A. (1991). Afterthoughts: Yin and Yang in the Chinese Room. *The Nature of Mind*. D. M. Rosenthal, Ed. Oxford, Oxford University Press**:** 524-525.
Fodor, J. A. (1997). *Special Sciences: Still Autonomous after All These Years*. Ridgeview, CA.
Fodor, J. A. (1998). *Concepts*. Oxford, Clarendon Press.
Fodor, J. A. (2000). *The Mind Doesn't Work That Way*. MIT Press, Cambridge, MA.
Fodor, J. A. and Z. W. Pylyshyn (1988). "Connectionism and Cognitive Architecture." *Cognition* **28**: 3-71.
Folina, J. (1998). "Church's Thesis: Prelude to a Proof." *Philosophia Mathematica* **6**: 302-323.

Frank, R. (1994). "Instruments, Nerve Action, and the All-or-None Principle." *Osiris* **9**: 208-235.
Freud, S. (1895). *Project for a Scientific Psychology*. Unpublished manuscript.
Gandy, R. (1980). Church's Thesis and Principles for Mechanism. *The Kleene Symposium*. J. Barwise, H. J. Keisler and K. Kuhnen, Eds. Amsterdam, North-Holland**:** 123-148.
Gandy, R. (1988). The Confluence of Ideas in 1936. *The Universal Machine: A Half-Century Survey*. R. Herken, Ed. New York, Oxford University Press**:** 55-111.
Gardner, H. (1985). *The Mind's New Science: A History of the Cognitive Revolution*. New York, Basic Books.
Geach, P. T. (1956). *Mental Acts*. London, Routledge & Paul.
Giunti, M. (1997). *Computation, Dynamics, and Cognition*. New York, Oxford University Press.
Glymour, C. (1990). Philosophy and the Academy. *Acting and Reflecting: The Interdisciplinary Turn in Philosophy*. W. Sieg, Ed. Dordrecht, Kluwer**:** 63-71.
Glymour, C., K. M. Ford, et al. (1995). The Prehistory of Android Epistemology. *Android Epistemology*. K. M. Ford, C. Glymour and P. J. Hayes, Eds. Menlo Park, AAAI Press**:** 3-21.
Gödel, K. (1931). "On Formally Undecidable Propositions of Principia Mathematica and Related Systems I." *Monaschefte für Mathematik und Physik* **38**: 173-198.
Gödel, K. (1934). On Undecidable Propositions of Formal Mathematical Systems. *The Undecidable*. M. Davis, Ed. Ewlett, NY, Raven**:** 41-71.
Gödel, K. (1936). "Über die Lange von Beweisen." *Ergebnisse eines mathematischen Kolloquiums* **7**: 23-24.
Gödel, K. (1965). Postscriptum. *The Undecidable*. M. Davis, Ed. New York, Raven**:** 71-73.
Gödel, K. (1972). Some Remarks on the Undecidability Results. *Collected Works, Vol. II*. S. Feferman and e. al, Eds. Oxford, Oxford University Press, 1990**:** 305-306.
Goldstine, H. H. and J. von Neumann (1946). On the Principles of Large Scale Computing Machines. Princeton, Institute for Advanced Studies.
Grush, R. (2001). The Semantic Challenge to Computational Neuroscience. *Theory and Method in the Neurosciences*. P. Machamer, R. Grush and P. McLaughlin, Eds. Pittsburgh, PA, University of Pittsburgh Press**:** 155-172.
Hallett, M. (1994). Hilbert's Axiomatic Method and the Laws of Thought. *Mathematics and Mind*. G. Alexander, Ed. New York, Oxford University Press**:** 158-200.
Hamkins, J. D. (2002). "Infinite Time Turing Machines." *Minds and Machines* **12**: 521-539.
Harel, D. (2000). *Computers Ltd: What They Really Can't Do*. Oxford, Oxford University Press.
Harman, G. (1968). "Three Levels of Meaning." *Journal of Philosophy* **65**: 590-602.
Harman, G. (1973). *Thought*. Princeton, Princeton University Press.
Harman, G. (1987). (Nonsolipsistic) Conceptual Role Semantics. *New Directions in Semantics*. E. LePore, Ed. London, Academic Press**:** 55-81.
Harman, G. (1988). Wide Functionalism. *Cognition and Representation*. S. Schiffer and S. Steele, Eds. Boulder, Westview**:** 11-20.
Harnad, S. (1994). "Special Issue on What is Computation?" *Minds and Machines* **4**(4).
Hartley, R. V. (1929). "Transmission of Information." *Bell System Technical Journal* **7**(535-553).
Haugeland, J. (1978). "The Nature and Plausibility of Cognitivism." *Behavioral and Brain Sciences* **2**: 215-260.
Haugeland, J. (1981). "Analog and Analog." *Philosophical Topics* **12**: 213-225.
Haugeland, J. (1997). *Mind Design II*. Cambridge, MA, MIT Press.
Heims, S. J. (1980). *John von Neumann and Norbert Wiener: From Mathematics to the Technologies of Life and Death*. Cambridge, MIT Press.
Heims, S. J. (1991). *Constructing a Social Science for Postwar America: The Cybernetics Group, 1946-1953*. Cambridge, MIT Press.
Hempel, C. (1935/1949). The Logical Analysis of Psychology. *Readings in Philosophical Analysis*. H. Feigl and W. Sellars, Eds. New York, Appleton-Century-Crofts**:** 373-384.
Hilbert, D. and W. Ackermann (1928). *Grunzüge der theoretischen Logik*. Berlin, Springer.

Hodges, A. (1983). *Alan Turing: The Enigma*. New York, Simon and Schuster.
Hodges, A. (1997). *Turing; a Natural Philosopher*. London, Phoenix.
Hogarth, M. (1994). "Non-Turing Computers and Non-Turing Computability." *PSA 1994*: 126-138.
Hopfield, J. J. (1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of the National Academy of Sciences* **79**: 2554-2558.
Horgan, T. and J. Tienson, Eds. (1991a). *Connectionism and the Philosophy of Mind*. Dordrecht, Kluwer.
Hughes, R. I. G. (1999). The Ising Model, Computer Simulation, and Universal Physics. *Models as Mediators*. M. S. Morgan and M. Morrison, Eds. Cambridge, Cambridge University Press**:** 97-145.
Humphreys, P. (1990). "Computer Simulations." *PSA 1990* **2**: 497-506.
Ince, D., Ed. (1992). *Mechanical Intelligence*. The Collected Works of Alan Turing. Amsterdam, North-Holland.
Irvine, A. D. (2001). Bertrand Russell. *The Stanford Encyclopedia of Philosophy (Fall 2001 Edition)*. E. N. Zalta, Ed. URL = <http://plato.stanford.edu/archives/fall2001/entries/russell/>.
Irvine, A. D., Ed. (2002). *Principia Mathematica*. Stanford Encyclopedia of Philosophy (Spring 2002 Edition). URL = <http://plato.stanford.edu/archives/spr2002/entries/principia-mathematica/>.
Israel, D. (2002). "Reflections on Gödel's and Gandy's Reflections on Turing's Thesis." *Minds and Machines* **12**: 181-201.
Jackson, F. and P. Pettit (1988). "Functionalism and Broad Content." *Mind* **XCVII**: 381-400.
Jacquette, D. (1991). The Myth of Pure Syntax. *Topics in Philosophy and Artificial Intelligence*. L. Albertazzi and R. Poli, Eds. Bozen, Istituto Mitteleuropeo di Cultura**:** 1-14.
Jeannerod, M. (1985). *The Brain Machine: The Development of Neurophysiological Thought*. Cambridge, MA, Harvard University Press.
Jeffress, L. A., Ed. (1951). *Cerebral Mechanisms in Behavior*. New York, Wiley.
Kálmar, L. (1959). An Argument Against the Plausibility of Church's Thesis. *Constructivity in Mathematics*. A. Heyting, Ed. Amsterdam, North-Holland**:** 72-80.
Keeley, B. (2000). "Shocking Lessons from Electric Fish: The Theory and Practice of Multiple Realizability." *Philosophy of Science* **67**: 444-465.
Kieu, T. D. (2002). "Quantum Hypercomputation." *Minds and Machines* **12**: 541-561.
Kim, J. (1989). "The Myth of Nonreductive Materialism." *Proceedings and Addresses of the American Philosophical Association* **63**: 31-47.
Kim, J. (1992). "Multiple Realization and the Metaphysics of Reduction." *Philosophy and Phenomenological Research* **52**: 1-26.
Kim, J. (1996). *Philosophy of Mind*. Boulder, Westview.
Kleene, S. C. (1935). "A Theory of Positive Integers in Formal Logic." *American Journal of Mathematics* **57**: 153 - 173 and 219 - 244.
Kleene, S. C. (1938). "On Notation for Ordinal Numbers." *Journal of Symbolic Logic* **3**: 150-155.
Kleene, S. C. (1952). *Introduction to Metamathematics*. Princeton, Van Nostrand.
Kleene, S. C. (1956). Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*. C. E. Shannon and J. McCarthy, Eds. Princeton, NJ, Princeton University Press**:** 3-42.
Kleene, S. C. (1979). Origins of Recursive Function Theory. *20th Annual Symposium on Foundations of Computer Science*. New York, IEEE**:** 371-382.
Kleene, S. C. (1987a). Gödel's impression on students of logic in the 1930s. *Gödel Remembered*. P. Weingartner and L. Schmetterer, Eds. Napoli, Bibliopolis**:** 49-64.
Kleene, S. C. (1987b). "Reflections on Church's Thesis." *Notre Dame Journal of Formal Logic* **28**(4): 490-498.
Koch, C. (1999). *Biophysics of Computation: Information Processing in Single Neurons*. New York, Oxford University Press.
Koch, C. and I. Segev (2000). "The role of single neurons in information processing." *Nature Neuroscience Supplement* **3**: 1171-1177.
Köhler, W. (1938). *The Place of Value in a World of Fact*. New York, Liveright.

Kreisel, G. (1972). "Which Number Theoretic Problems can be Solved in Recursive Progressions on $\Phi\frac{1}{1}$-Paths Through O?" *Journal of Symbolic Logic* **37**(2): 311-334.
Kreisel, G. (1974). "A Notion of Mechanistic Theory." *Synthese* **29**: 11-26.
Kreisel, G. (1987). "Church's Thesis and the Ideal of Informal Rigour." *Notre Dame Journal of Formal Logic* **28**(4): 499-519.
Kubie, L. (1930). "A Theoretical Application to some Neurological Problems of the Properties of Excitation Waves which Move in Closed Circuits." *Brain* **53**(2): 166-177.
Kubie, L. (1941). "The Repetitive Core of Neuroses." *Psychoanalytic Quarterly* **10**: 23-43.
Leiber, J. (1991). *An Invitation to Cognitive Science*. Cambridge, MA, Basil Blackwell.
Lettvin, J. L. (1989a). Introduction. *Collected Works of Warren S. McCulloch, Vol. 1*. R. McCulloch, Ed. Salinas, CA, Intersystems**:** 7-20.
Lettvin, J. L. (1989b). Warren and Walter. *Collected Works of Warren S. McCulloch, Vol. 2*. R. McCulloch, Ed. Salinas, CA, Intersystems**:** 514-529.
Lettvin, J. L., H. R. Maturana, et al. (1959). "What the Frog's Eye Tells the Frog's Brain." *IRE*: 1940-1959.
Lettvin, J. L. and W. H. Pitts (1943). "A Mathematical Theory of Affective Psychoses." *Bulletin of Mathematical Biophysics* **5**: 139-148.
Lewis, D. K. (1966). "An Argument for the Identity Theory." *Journal of Philosophy* **63**: 17-25.
Lewis, D. K. (1969). "Review of *Art, Mind, and Religion*." *Journal of Philosophy* **66**(22-27).
Lewis, D. K. (1972). "Psychophysical and Theoretical Identifications." *Australasian Journal of Philosophy* **50**: 249-258.
Lewis, D. K. (1980). Mad Pain and Martian Pain. *Readings in Philosophy of Psychology, Volume 1*. N. Block, Ed. Cambridge, MA, MIT Press**:** 216-222.
Loar, B. (1981). *Mind and Meaning*. Cambridge, Cambridge University Press.
Loewer, B. and G. Rey, Eds. (1991). *Meaning in Mind: Fodor and his Critics*. Oxford, Blackwell.
Lorente de Nó, R. (1938). "Analysis of the Activity of the Chains of Internuncial Neurons." *Journal of Neurophysiology* **I**: 207-244.
Lorente de Nó, R. (1947). *A Study of Nerve Physiology*. New York, Rockefeller Institute.
Lycan, W. (1981). "Form, Function, and Feel." *Journal of Philosophy* **78**: 24-50.
Lycan, W. (1987). *Consciousness*. Cambridge, MA, MIT Press.
Machamer, P. K., L. Darden, et al. (2000). "Thinking About Mechanisms." *Philosophy of Science* **67**: 1-25.
Machtey, M. and P. Young (1978). *An Introduction to the General Theory of Algorithms*. New York, North Holland.
Magnus, R. (1930). *Lane Lectures on Experimental Pharmacology and Medicine*. Stanford University, Stanford University Press.
Mahner, M. and M. Bunge (2001). "Function and Functionalism: A Synthetic Perspective." *Philosophy of Science* **68**(March 2001): 75-94.
Mancosu, P. (1998). *From Brower to Hilbert: The Debate on the Foundations of Mathematics in the 1920s*. New York, Oxford University Press.
Marr, D. (1982). *Vision*. New York, Freeman.
Maudlin, T. (1989). "Computation and Consciousness." *Journal of Philosophy* **86**(8): 407-432.
McCorduck, P. (1979). *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. Freeman, S. Francisco, CA.
McCulloch, R., Ed. (1989). *Collected Works of Warren S. McCulloch, 4 voll.* Salinas, CA, Intersystems.
McCulloch, W. S. (1940). "Joannes Gregorius Dusser de Barenne." *Yale Journal of Biology and Medicine* **12**: 743-746.
McCulloch, W. S. (1944a). "The Functional Organization of the Cerebral Cortex." *Physiology* **Rev. 24**(3): 390-407.
McCulloch, W. S. (1944b). Cortico-Cortical Connections. *The Precentral Motor Cortex*. P. C. Bucy, Ed. Urbana, IL, University of Illinois Press**:** 211-242.

McCulloch, W. S. (1949). "The Brain as a Computing Machine." *Electrical Engineering* **68**: 492-497.
McCulloch, W. S. (1961). "What Is a Number, that a Man May Know It, and a Man, that He May Know a Number?" *General Semantics Bulletin* **26/27**: 7-18.
McCulloch, W. S. (1965). *Embodiments of Mind*. Cambridge, MA, MIT Press.
McCulloch, W. S. (1974). "Recollections of the Many Sources of Cybernetics." *ASC Forum* **VI**(2): 5-16.
McCulloch, W. S. and W. H. Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* **7**: 115-133.
McGee, V. (1991). "We Turing Machines Aren't Expected-Utility Maximizers (Even Ideally)." *Philosophical Studies* **64**: 115-123.
Mendelson, E. (1963). "On Some Recent Criticism of Church's Thesis." *Notre Dame Journal of Formal Logic* **4**: 201-205.
Mendelson, E. (1990). "Second Thoughts about Church's Thesis and Mathematical Proofs." *The Journal of Philosophy* **88**: 225-233.
Miller, G. A., E. H. Galanter, et al. (1960). *Plans and the Structure of Behavior*. New York, Holt.
Millikan, R. G. (1984). *Language, Thought, and Other Biological Categories: New Foundations for Realism*. Cambridge, MA, MIT Press.
Millikan, R. G. (1993). *White Queen Psychology and Other Essays for Alice*. Cambridge, MA, MIT Press.
Moor, J. H. (2001). "The Status and Future of the Turing Test." *Minds and Machines* **11**: 77-93.
Nelson, R. J. (1987). "Church's Thesis and Cognitive Science." *Notre Dame Journal of Formal Logic* **28**(4): 581-614.
Newell, A. (1980). "Physical Symbol Systems." *Cognitive Science* **4**: 135-183.
Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA, Harvard University Press.
Newell, A. and H. A. Simon (1976). "Computer Science as an Empirical Enquiry: Symbols and Search." *Communications of the ACM* **19**: 113-126.
Newman, M. H. A. (1954). "obituary notice for Alan Turing." *The Times*(16 June).
Newman, M. H. A. (1955). Alan Mathison Turing. *Biographical Memoirs of Fellows of the Royal Society*. London, The Royal Society. **253-263**.
Northrop, F. S. C. (1940). "The Method and Theories of Physical Science and Their Bearing Upon Biological Organization." *Supplement, Second Symposium on Development and Growth* **4**: 127-154.
Odifreddi, P. (1989). *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. Amsterdam, North-Holland.
O'Donohue, W. and R. Kitchener (1999). *Handbook of Behaviorism*. San Diego, Academic.
Oppenheim, P. and H. Putnam (1958). Unity of Science as a Working Hypothesis. *Minnesota Studies in the Philosophy of Science, Volume II Concepts, Theories, and the Mind-Body Problem*. H. Feigl, M. Scriven and G. Maxwell, Eds. Minneapolis, University of Minnesota Press**:** 3-36.
Patterson, D. A. and J. L. Hennessy (1998). *Computer Organization and Design: The Hardware/Software Interface*. San Francisco, Morgan Kauffman.
Peacocke, C. (1992). *A Study of Concepts*. Cambridge, MA, MIT Press.
Peacocke, C. (1994a). "Content, Computation, and Externalism." *Mind and Language* **9**: 303-335.
Peacocke, C. (1994b). Content. *A Companion to the Philosophy of Mind*. S. Guttenplan, Ed. Oxford, Blackwell**:** 219-225.
Peacocke, C. (1999). "Computation as Involving Content: A Response to Egan." *Mind and Language* **14**(2): 195-202.
Penrose, R. (1994). *Shadows of the Mind*. Oxford, Oxford University Press.
Péter, R. (1959). Rekursivität und Konstructivität. *Constructivity in Mathematics*. Amsterdam**:** 226-233.
Piccinini, G. (2000). "Turing's Rules for the Imitation Game." *Minds and Machines* **10**(4): 573-582.
Piccinini, G. (2002). "Review of Jean-Pierre Dupuy's The Mechanization of Mind: On the Origins of Cognitive Science." *Minds and Machines* **12**(3): 449-453.
Piccinini, G. (2003a). "Alan Turing and the Mathematical Objection." *Minds and Machines* **13**(1): 23-48.

Piccinini, G. (2003b). "Review of John von Neumann's The Computer and the Brain." *Minds and Machines* **13**(2): 327-332.
Piccinini, G. (forthcoming). "Epistemic Divergence and the Publicity of Scientific Methods." *Studies in the History and Philosophy of Science*.
Pitowski, I. (1990). "The Physical Church Thesis and Physical Computational Complexity." *Iyyun* **39**: 81-99.
Pitts, W. H. (1942a). "Some Observations on the Simple Neuron Circuit." *Bulletin of Mathematical Biophysics* **4**: 169-175.
Pitts, W. H. (1942b). "The Linear Theory of Neuron Networks: The Static Problem." *Bulletin of Mathematical Biophysics* **4**: 176.
Pitts, W. H. (1943). "The Linear Theory of Neuron Networks: The Dynamic Problem." *Bulletin of Mathematical Biophysics* **5**: 23-31.
Pitts, W. H. and W. S. McCulloch (1947). "How We Know Universals: The Perception of Auditory and Visual Forms." *Bulletin of Mathematical Biolophysics* **9**: 127-147.
Place, U. T. (1956). "Is Consciousness a Brain Process?" *British Journal of Psychology* **47**: 44-50.
Porte, J. (1960). Quelques Pseudo-paradoxes de la "Calculabilité Effective". *Actes du 2me Congrès International de Cybernetique*. Namur, Belgium**:** 332-334.
Post, E. (1936). "Finite Combinatorial Processes. Formulation I." *Journal of Symbolic Logic* **1**: 103-105.
Pour-El, M. B. and J. I. Richards (1989). *Computability in Analysis and Physics*. Berlin, Springer Verlag.
Prasse, M. and P. Rittgen (1998). "Why Church's thesis still holds: Some notes on Peter Wegner's tracts on interaction and computability." *Computer Journal* **41**(6): 357-362.
Proudfoot, D. and B. J. Copeland (1994). "Turing, Wittgenstein and the Science of the Mind." *Australasian Journal of Philosophy* **72**(497-519).
Putnam, H. (1960). Minds and Machines. *Dimensions of Mind: A Symposium*. S. Hook, Ed. New York, Collier**:** 138-164.
Putnam, H. (1961). Some Issues in the Theory of Grammar. *Proceedings of Symposia in Applied Mathematics*, The American Mathematical Society. **12:** 25-42.
Putnam, H. (1963). Brains and Behavior. *Analytical Philosophy*. R. J. Butler, Ed. New York, Barnes and Noble**:** 1-20.
Putnam, H. (1964). "Robots: Machines or Artificially Created Life?" *Journal of Philosophy* **LXI**(November 1964): 668-691.
Putnam, H. (1967a). The Mental Life of Some Machines. *Intentionality, Minds, and Perception*. H. Castañeda, Ed. Detroit, Wayne State University Press**:** 177-200.
Putnam, H. (1967b). Psychological Predicates. *Art, Philosophy, and Religion*. Pittsburgh, PA, University of Pittsburgh Press.
Putnam, H. (1988). *Representation and Reality*. Cambridge, MA, MIT Press.
Putnam, H. (1997). "A Half Century of Philosophy, Viewed from Within." *Daedalus* **Winter 1997**: 175-208.
Pylyshyn, Z. W. (1984). *Computation and Cognition*. Cambridge, MA, MIT Press.
Rashevsky, N. (1936). "Physico-mathematical Methods in Biology." *Biological Reviews* **11**: 345-363.
Rashevsky, N. (1937). "Physico-mathematical Methods in Biological and Social Sciences." *Erkenntnis* **6**(5/6): 357-365.
Rashevsky, N. (1938). *Mathematical Biophysics: Physicomathematical Foundations of Biology*. Chicago, University of Chicago Press.
Rey, G. (1997). *Contemporary Philosophy of Mind: A Contentiously Classic Approach*. Cambridge, MA, Blackwell.
Rohrlich, F. (1990). "Computer Simulation in the Physical Sciences." *PSA 1990* **2**: 507-518.
Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain." *Psychological Review* **65**: 386-408.
Rosenblueth, A., N. Wiener, et al. (1943). "Behavior, Purpose, and Teleology." *Philosophy of Science* **10**: 18-24.

Rubel, L. A. (1989). "Digital Simulation of Analog Computation and Church's Thesis." *Journal of Symbolic Logic* **54**(3): 1011-1017.
Rumelhart, D. E. and J. M. McClelland (1986). *Parallel Distributed Processing*. Cambridge, MA, MIT Press.
Russell, B. A. W. (1914). *Our Knowledge of the External World*. Chicago and London, Open Court.
Ryle, G. (1949). *The Concept of Mind*. London, Hutchinson.
Scheutz, M. (1999). "When Physical Systems Realize Functions ..." *Minds and Machines* **9**: 161-196.
Schlosser, G. (1998). "Self-re-Production and Functionality: A Systems-Theoretical Approach to Teleological Explanation." *Synthese* **116**(3): 303-354.
Schmaus, W. (1999). "Functionalism and the Meaning of Social Facts." *Philosophy of Science* **66**: S314-S323.
Searle, J. R. (1980). "Minds, Brains, and Programs." *The Behavioral and Brain Sciences* **3**: 417-457.
Searle, J. R. (1991). Yin and Yang Strike Out. *The Nature of Mind*. D. M. Rosenthal, Ed. Oxford, Oxford University Press**:** 525-526.
Searle, J. R. (1992). *The Rediscovery of the Mind*. Cambridge, MA, MIT Press.
Segal, G. (1989). "Seeing What is Not There." *Philosophical Review* **98**: 189-214.
Segal, G. (1991). "Defence of a Reasonable Individualism." *Mind* **100**: 485-493.
Selfridge, O. G. (1955). *Pattern Recognition and Modern Computers*. Proceedings of the 1955 Western Joint Computer Conference, IRE.
Selfridge, O. G. (1959). *Pandemonium, a Paradigm for Learning*. Proceedings of the Symposium on Mechanisation of Thought Processes, London, H. M. Stationery Office.
Seligman, J. (2002). "The Scope of Turing's Analysis of Effective Procedures." *Minds and Machines* **12**: 203-220.
Sellars, W. (1954). "Some Reflections on Language Games." *Philosophy of Science* **21**: 204-228.
Sellars, W. (1956). Empiricism and the Philosophy of Mind. *Minnesota Studies in the Philosophy of Science, Vol. I, The Foundations of Science and the Concepts of Psychology and Psychoanalysis*. H. Feigl and M. Scriven, Eds. Minneapolis, University of Minnesota Press.
Sellars, W. (1963). *Science, Perception, and Reality*. Atascadero, Ridgeview.
Sellars, W. (1974). "Meaning as Functional Classification." *Synthese* **27**: 417-437.
Shagrir, O. (1997). "Two Dogmas of Computationalism." *Minds and Machines* **7**(3): 321-344.
Shagrir, O. (1998). "Multiple Realization, Computation and the Taxonomy of Psychological States." *Synthese* **114**: 445-461.
Shagrir, O. (1999). "What is Computer Science About?" *The Monist* **82**(1): 131-149.
Shagrir, O. (2002). "Effective Computation by Humans and Machines." *Minds and Machines* **12**: 221-240.
Shagrir, O. and I. Pitowski (2003). "Physical Hypercomputation and the Church-Turing Thesis." *Minds and Machines* **13**(1): 87-101.
Shanker, S. G. (1987). "Wittgenstein versus Turing on the Nature of Church's Thesis." *Notre Dame Journal of Formal Logic* **28**: 615-649.
Shanker, S. G. (1995). "Turing and the Origins of AI." *Philosophia Mathematica* **3**: 52-85.
Shannon, C. E. (1938). "A Symbolic Analysis of Relay and Switching Circuits." *Transactions of the American Institute of Electrical Engineers* **57**: 713-723.
Shannon, C. E. (1948). "A Mathematical Theory of Communication." *Bell System Technical Journal* **27**: 379-423 and 623-656.
Shannon, C. E. (1952). Presentation of a Maze Solving Machine. *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Eighth Conference*. H. von Foerster, M. Mead and H. L. Teuber, Eds. New York, Macy Foundation**:** 169-181.
Shannon, C. E. and J. McCarthy (1956). *Automata Studies*. Princeton, NJ, Princeton University Press.
Shannon, C. E. and W. Weaver (1949). *The Mathematical Theory of Communication*. Urbana, University Illinois Press.
Shapiro, L. (1997). "A Clearer Vision." *Philosophy of Science* **64**: 131-153.

Shapiro, S. (1983). "Remarks on the Development of Computability." *History and Philosophy of Logic* **4**: 203-220.
Shapiro, S. (1993). "Understanding Church's Thesis, Again." *Acta Analytica* **11**: 59-77.
Shapiro, S. (1995). "Reasoning, Logic and Computation." *Philosophia Mathematica* **3**(3): 31-51.
Sherrington, C. S. (1940). *Man on His Nature*. Cambridge, Cambridge University Press.
Shoemaker, S. (1984). *Identity, Cause and Mind*. Cambridge, Cambridge University Press.
Sieg, W. (1994). Mechanical Procedures and Mathematical Experience. *Mathematics and Mind*. G. Alexander, Ed. New York, Oxford University Press**:** 71-117.
Sieg, W. (1997). "Step by Recursive Step: Church's Analysis of Effective Calculability." *Bulletin of Symbolic Logic* **3**(2): 154-180.
Sieg, W. (2001). Calculations by Man and Machine: Conceptual Analysis. *Reflections on the Foundations of Mathematics (Essays in Honor of Solomon Feferman)*. W. Sieg, R. Sommer and C. Talcott, Eds, Association for Symbolic Logic. **15:** 387-406.
Sieg, W. (forthcoming). Calculation by Man and Machine: Mathematical Presentation. *Proceedings of the International Congress of Logic, Methodology and Philosophy of Science*. Cracow, 1999, Kluwer**:** 246-260.
Sieg, W. and J. Byrnes (1996). K-graph Machines: Generalizing Turing's Machines and Arguments. *Gödel '96*. P. Hájek, Ed. Berlin, Springer Verlag.
Sieg, W. and J. Byrnes (1999). "An Abstract Model for Parallel Computation: Gandy's Thesis." *The Monist* **82**(1): 150-164.
Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Boston, MA, Birkhäuser.
Siegelmann, H. T. (2003). "Neural and Super-Turing Computing." *Minds and Machines* **13**(1): 103-114.
Smalheiser, N. R. (2000). "Walter Pitts." *Perspectives in Biology and Medicine* **43**(2): 217-226.
Smart, J. J. C. (1959). "Sensations and Brain Processes." *Philosophical Review* **68**: 141-156.
Smith, B. C. (1996). *On the Origin of Objects*. Cambridge, MA, MIT Press.
Smolensky, P. (1988). "On the Proper Treatment of Connectionism." *Behavioral and Brain Sciences* **11**: 1-23.
Sober, E. (1990). Putting the Function Back into Functionalism. *Mind and Cognition*. W. Lycan, Ed. Malden, MA, Blackwell**:** 63-70.
Sober, E. (1999). "The Multiple Realizability Argument Against Reductionism." *Philosophy of Science* **66**: 542-564.
Stannett, M. (1990). "X-Machines and the Halting Problem: Building a Super-Turing Machine." *Formal Aspects of Computing* **2**: 331-341.
Stannett, M. (2003). "Computation and Hypercomputation." *Minds and Machines* **13**(1): 115-153.
Stich, S. (1983). *From Folk Psychology to Cognitive Science*. Cambridge, MA, MIT Press.
Strawson, P. (1959). *Individuals*. London, Methuen.
Strogatz, S. H. (1994). *Nonlinear Dynamics and Chaos*. Cambridge, MA, Perseus.
Strogatz, S. H. (2003). *Sync: The Emerging Science of Spontaneous Order*. New York, Hyperion.
Tamburrini, G. (1988). Reflections on Mechanism. New York, Columbia University.
Tamburrini, G. (1997). Mechanistic Theories in Cognitive Science: The Import of Turing's Thesis. *Logic and Scientific Method*. M. L. Dalla Chiara and e. al., Eds, Kluwer**:** 239-257.
Thelen, E. and L. Smith (1994). *A Dynamic Systems Approach to the Development of Cognition and Action*. Cambridge, MA, MIT Press.
Turing, A. (1948). Intelligent Machinery. *Collected Works of A. M. Turing: Mechanical Intelligence*. D. C. Ince, Ed. Amsterdam, North Holland**:** 87-105.
Turing, A. M. (1936-7 [1965]). On computable numbers, with an application to the Entscheidungsproblem. *The Undecidable*. M. Davis, Ed. Ewlett, Raven.
Turing, A. M. (1939). "Systems of Logic Based on Ordinals." *Proceedings of the London Mathematical Society, Ser. 2* **45**: 161-228.

Turing, A. M. (1945). Proposal for Development in the Mathematical Division of an Automatic Computing Engine (ACE). *Mechanical Intelligence*. D. Ince, Ed. Amsterdam, North-Holland**:** 20-105.
Turing, A. M. (1947). Lecture to the London Mathematical Society on 20 February 1947. *Mechanical Intelligence*. D. Ince, Ed. Amsterdam, North-Holland**:** 87-105.
Turing, A. M. (1948). Intelligent Machinery. *Mechanical Intelligence*. D. Ince, Ed. Amsterdam, North-Holland**:** 87-106.
Turing, A. M. (1950). "Computing Machinery and Intelligence." *Mind* **59**: 433-460.
Turing, A. M. (1951). *Programmers' Handbook for the Manchester Electronic Computer*, University of Manchester Computing Laboratory.
Turing, A. M. (1953). Digital Computers Applied to Games. *Faster Than Thought*. B. V. Bowden, Ed. London, Pittman**:** 286-310.
Turing, S. E. (1959). *Alan M. Turing*. Cambridge, Heffer.
van Gelder, T. (1995). "What Might Cognition Be, if not Computation?" *The Journal of Philosophy* **XCII**(7): 345-381.
van Heijenoort, J., Ed. (1967). *From Frege to Gödel*. Cambridge, MA, Harvard University Press.
Vendler, Z. (1972). *Res Cogitans: An Essay in Rational Psychology*. Ithaca, Corness University Press.
von Domarus, E. (1967). Logical Structure of Mind. *Communication: Theory and Research*. L. Thayer, Ed. Springfield, Thomas.
von Foerster, H. (1950). *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Sixth Conference*. New York, Macy Foundation.
von Foerster, H., M. Mead, et al., Eds. (1951). *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Seventh Conference*. New York, Macy Foundation.
von Foerster, H., M. Mead, et al., Eds. (1952). *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Eighth Conference*. New York, Macy Foundation.
von Foerster, H., M. Mead, et al., Eds. (1953). *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Ninth Conference*. New York, Macy Foundation.
von Foerster, H., M. Mead, et al., Eds. (1955). *Cybernetics: Circular Causal and Feedback Mechanisms in Biological and Social Systems. Transactions of the Tenth Conference*. New York, Macy Foundation.
von Neumann, J. (1945). First Draft of a Report on the EDVAC. Philadelphia, PA, Moore School of Electrical Engineering, University of Pennsylvania.
von Neumann, J. (1951). The General and Logical Theory of Automata. *Cerebral Mechanisms in Behavior*. L. A. Jeffress, Ed. New York, Wiley**:** 1-41.
von Neumann, J. (1956). Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. *Automata Studies*. C. E. Shannon and J. McCarthy, Eds. Princeton, NJ, Princeton University Press**:** 43-98.
von Neumann, J. (1958). *The Computer and the Brain*. New Haven, Yale University Press.
von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. Urbana, University of Illinois Press.
Wang, H. (1974). *From Mathematics to Philosophy*. New York, Humanities Press.
Watson, A. G. D. (1938). "Mathematics and Its Foundations." *Mind* **47**: 440-451.
Webb, J. C. (1980). *Mechanism, Mentalism, and Metamathematics*. Dordrecht, Reidel.
Wegner, P. (1999). "Towards Empirical Computer Science." *The Monist* **82**(1): 58-108.
Wells, A. J. (1998). "Turing's Analysis of Computation and Theories of Cognitive Architecture." *Cognitive Science* **22**(3): 269-294.
Whitehead, A. N. and B. Russell (1910-13). *Principia Mathematica*. Cambridge, Cambridge University Press.

Wiener, N. (1948). *Cybernetics or Control and Communication in the Animal and the Machine*. Cambridge, MA, MIT Press.
Wiener, N. (1958). "My Connection with Cybernetics." *Cybernetica* **1**: 1-14.
Wiener, N. (1976). *Collected Works*. Cambridge, MA, MIT Press.
Wilson, R. A. (1994). "Wide Computationalism." *Mind* **103**: 351-372.
Wittgenstein, L. (1922). *Tractatus Logico-Philosophicus*. New York, Harcourt, Brace and Company.
Wittgenstein, L. (1953). *Philosophical Investigations*. New York, Macmillan.
Wittgenstein, L. (1976). *Wittgenstein's Lectures on the Foundations of Mathematics Cambridge, 1939*. Ithaca, NY, Corness University Press.
Wittgenstein, L. (1980). *Remarks on the Philosophy of Psychology, Vol. 1*. Chicago, University of Chicago Press.
Wolfram, S. (1985). "Undecidability and Intractability in Theoretical Physics." *Physical Review Letters* **54**: 735-738.
Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL, Wolfram Media.
Woodger, J. H. (1937). *The Axiomatic Method in Biology*. Cambridge, Cambridge University Press.
Zangwill, N. (1992). "Variable Realization: Not Proved." *The Philosophical Quarterly* **42**(167): 214-219.