PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE PSICOLOGÍA

CONSEQUENCES OF THEORETICALLY MODELING THE MIND AS A COMPUTER

ESTEBAN HURTADO LEÓN

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor in Psychology

Advisor:
CARLOS CORNEJO ALARCÓN

Santiago de Chile, August 2017

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE PSICOLOGÍA

CONSEQUENCES OF THEORETICALLY MODELING THE MIND AS A COMPUTER

ESTEBAN HURTADO LEÓN

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor in Psychology

Santiago de Chile, August 2017

To Carmen and Fulvio

ACKNOWLEDGEMENTS

I would like to thank the School of Psychology at Pontificia Universidad Católica de Chile for taking me in and walking me through the diversity of the study of the mind. I am in debt to all the teachers who kindly and passionately shared their knowledge with me, and very specially to the kind and helpful work of the administrative staff.

I took my first steps in theoretical computer science at the School of Engineering of the same university, with Dr. Álvaro Campos, who is no longer with us. His passion for knowledge, dedication and warmth continue to inspire those of us who where lucky enough to cross paths with him.

The generous and theoretically profound support of the committee members has been fundamental to the production of this text. I am deeply thankful to all of them. I would like to specially thank Dr. Luis Dissett, from whom I learnt many of the issues discussed in this work. From him I also learnt to see logic as a beautiful form of art. His love for maths reverbs in the hearts of many of his students, myself included.

I thank my dear friend Dr. Agustín Ibañez, for introducing me to cognitive science, stimulating my development of an individual critical position in the area, and encouraging me to pursue an academic career. This work would have not been possible without his inspiration and support. I also thank Roberto Musa and David Carré, for the stimulating conversations we have had have been a real contribution to this thesis.

I want to thank my thesis advisor, Dr. Carlos Cornejo, in a very special way. This work, its approach, and my whole interest in the questions embedded in it, would have not been possible without his humble and theoretically profound guidance. With humility and true admiration, I hope this is only the beginning of a long collaboration.

I would like to thank CONICYT for funding my doctoral studies. I also thank the generous spiritual and administrative support I received from Jan Krebs, Héctor Bernales, David Huepe, and Gorka Naverrete, all people I have had the privilege to work for.

I dedicate this work to my parents, Carmen and Fulvio. I thank them deeply for their encouragement, dedication and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The computational theory of mind holds that the mind is a computer. It does not restrict how the computer is to be programmed, but provides a metatheoretical framework for particular theories to propose different ways in which computer systems can support mental processes. It also functions as a claim about the nature of the mind. Its main goal is to explain mental phenomena based on computational processes.

This work argues that well known limitations of computers require more attention in order to understand the possibilities and limitations of the computational theory of mind itself. To that end, a revision of the Turing halting problem and Gödel's incomplenteness theorems is included as a foundation for arguments about what computational models of the mind could and could not achieve.

Contrary to traditional images, computational procedures can be diverse, flexible, adaptable, and deal with the uncertain in successful ways. Therefore, naive criticism of the computational theory of mind could be misplaced. We argue that the main limitations that computational modeling imposes on psychology are: that some important questions have no general answer; that theories are underdetermined by evidence in important ways; and that theories cannot answer specific questions about the nature of their object of study. We also show that the same limitations apply to any theory that restricts itself to making systematic predictions based on finite observations.

Keywords: computational theory of mind, Turing halting, Gödel's incompleteness, psychological theories, philosophy of mind

RESUMEN

La teoría computacional de la mente señala que la mente es un computador. No impone de qué manera ese computador debe ser programado, sino que establece un marco metateórico desde el cual diferentes teorías pueden proponer maneras en que procesos mentales se implementarían como sistemas computacionales. También cumple el rol de una afirmación acerca de la naturaleza de lo mental. Su principal fin es explicar fenómenos mentales con base en procesos computacionales.

El presente trabajo defiende que es necesario dar mayor atención a limitaciones de la computación que ya han sido estudiadas, con el fin de entender las posibilidades y limitaciones de la teoría computacional de la mente. Para ello, se presenta una revisión del problema de la detención de la máquina de Turing y de los teoremas de incompleción de Gödel, en tanto fundamento de argumentos acerca de lo que es o no posible lograr mediante modelos computacionales.

En oposición a nociones tradicionales, los procedimientos computacionales pueden ser diversos, flexibles, adaptables y lidiar con la incertidumbre de manera exitosa. Por lo tanto, las críticas ingenuas la teoría computacional de la mente podrían estar equivocadas. Aquí se argumenta que las principales limitaciones impuestas por el modelamiento computacional sobre la psicología son: la imposibilidad de dar respuesta general a ciertas preguntas; una indeterminación muy relevante de las teorías por la evidencia; y la imposibilidad de responder ciertas preguntas acerca de la naturaleza del objeto de estudio. También se muestra que las mismas limitaciones están presentes en cualquier teoría que se limite a realizar predicciones sistemáticas basadas en conjuntos finitos de observaciones.

Palabras Claves: teoría computacional de la mente, detención de la máquina de Turing, incompleción de Gödel, teorías psicológicas, filosofía de la mente

1. INTRODUCTION: A HIDDEN PROBLEM

When digital computers were invented in the mid-20th century, they were unlike any previously existing machine. Calculating tools, like the abacus, already existed. Autonomous mechanisms, like clocks, were also present. But never had those two aspects been together in a man-made object. The new invention was capable of doing arithmetic, performing logical operations, sorting records, applying transformations to text, and several other tasks that, at the time, were considered exclusive of intelligent beings. And it was capable of doing all that on its own. It was natural to think that those were just early demonstrations of what could become a bigger achievement: the fabrication of a human-like intelligence.

As a consequence, several efforts began to explore ways in which computer machines could display intelligent behavior (e.g., Turing, 1950; Chomsky, 1957; Putnam, 1967; Fodor, 1975; Newell & Simon, 1976, and several others). The notion that computers could explain mental capabilities was attractive to psychology. For the first time the nature of mental processes could be understood and predictions could be made with swiss-clock precision. And the only requirement was to make an assumption that seemed reasonable: that the mind is indeed some kind of computer. This thesis is known as the *computational theory of mind* (CTM) and has been subject of much debate (Searle, 1980; Putnam, 1988; Penrose, 1999, etc.).

It is not the purpose of the present work to answer if the mind is a computer. We will focus on a related question: what are the metatheoretical consequences of modeling the mind as a computer system? We have known, for almost a century, that computer systems are subject to hard theoretical limitations that will not be overcome by better technology. We have to live with them. However, not much has been discussed on how such limitations impact the construction of psychological theories. The reason for that is not lack of interest. It is probably the fact that relationships between computer science and cognition are not always evident, and that researching them requires a trip along metamathematical issues that shape, as they should, a different discipline from cognitive science. We face a scenario in which important metatheoretical consequences of the CTM are solid enough to be provable, but are also implicit and effectively hidden in clear light. Before going deep into the issue, we will make an outline of the aforementioned consequences.

There is an intimate relationship between how a computer system is formally defined and what capabilities it has, which will be the focus of chapter 2. We can define a series of increasingly powerful computer systems, in the sense that each model can perform computations that the previous one cannot. But there is a limit among models of which an actual implementation can be built. And we have very compelling evidence that it is a hard one that cannot be overcome by any means. In other words, there is a *most powerful* effective model of computation. Odds are that the prize goes to what we know as the *Turing machine* (1936). Our current and strongly supported understanding is this: if the Turing machine cannot compute it, nothing will.

The story becomes complicated when we realize that there are several ways to define that *most powerful* computer system. For instance, while Post (1936) independently reached a similar definition to that of Turing, Church (1936) reached a radically different one. It took some work to prove that Turing's and Church's definitions were equivalent (Turing, 1937). In fact, to any class of computer model there are several different ways to define it. We are talking about definitions that look completely dissimilar, that use different words and concepts, and therefore you would not suspect they define systems with the exact same capabilities until you mathematically prove it. Having the exact *same* computational power has a strong meaning for the purposes of theorization, because it means that theories that adhere to one or the other model of computation have the same potential predictive power.

The realization that a computer system can be multiply defined comes together with some problems. First, two seemingly different metatheoretical approaches to cognitive theorization could end up being the same. For instance, we could think that artificial neural networks are a game changer with respect to classical formal-logic-driven schemes of early artificial intelligence. The former are flexible enough to learn patterns and generalize in order to successfully adapt to previously unseen examples (Hinton, 1986), just like humans and other living beings do. On the other hand, classic AI schemes are thought of as rigid structures that have a hard time dealing with new cases. It would be easy to state that neural networks are an improvement over previous theories. *Generalizing flexibility* would be the function that sets them apart. But we should analyze the case more carefully. We can observe flexibility in neural networks. But how do we prove that classical formal reasoning does not have that kind of flexibility? It is possible that the old scheme shares the same capabilities with neural networks, and that we are inclined to think otherwise because it is simply not evident how that could be. Actually, the usual way of implementing a neural network is by means of a programing language that uses the same old rigid formal rules. So, actually, neural networks, as a theoretical mind modeling device, do not add any new capability. They only show us that old rigid formal rules have some human-like capabilities we previously did not know of. In other words, after the development of neural networks, the predictive power of the CTM did not improve, it only became better understood.

Would it be enough if we just were more careful and worked more exhaustively when thinking about cognitive theories and their differences? Metamathematics hits us with a resounding *no*. Whether a given function is actually within the capabilities of a specific computational model is not a question that can be answered in general. In some cases we get to be so unlucky that we cannot even prove that the question has no answer. We are only left with the alternative of trying and hoping, knowing that it is possible that we never find what we search for. We can hold Church (1936), Turing (1936), Gödel (1931), and others responsible for delivering the bad news, as we will discuss on chapters 4 and 5.

A second problem is that a theory that is not explicitly computational may actually be unknowingly under the umbrella of the CTM. That would be the case if an alternative and explicitly computational theory can be proposed that generates exactly the same predictions. In that case, both imply that a computer has the necessary means to produce the mental functions that are being described.

We will approach such problems according to the following objectives. First, to clarify what a computer is and what its capabilities are. Second, to make a detailed review of mathematical concepts that are necessary to speak about the power and limitations of computers. Third, to review and question traditional views on the consequences of computer limitations for psychology. Fourth, make a new proposal about what the consequences are. Fifth, to debunk criticism of the CTM based on inaccurate traditional images of what a computer is. Sixth, to reveal limitations of the CTM on a different ground.

2. WHAT IS A COMPUTER SYSTEM?

Before we explore the consequences of modeling the mind as a computer system, it is fundamental that we agree on what a computer system is. In this effort we have two main goals. First, we aim to define a computer system that is as powerful as possible. Therefore, if our analysis concludes that a computationally modeled mind has limitations, it will not be possible to overcome it by considering more powerful computer systems. This way we will be able to explore limitations of CTM itself. Second, we must define a *mechanical* system, in the sense that it must be only governed by clear, unambiguous formal rules. This means that, just like Searle's (1980) Chinese room, our powerful computer system must not rely on creativity, interpretation, feelings, meaning, etc. Because, if it did rely on such preexisting mental capabilities, how could it be used to explain them?

In the most general way, when we talk about a computer system, we are talking about a *symbol processor*. In order to use it, we write a question or problem as a sequence of symbols, which we then feed to it. After processing, the symbol processor outputs a different sequence of symbols which is the answer. A digital calculator is a common example of this. If we want to ask how much is 7 plus 18, we feed the machine this sequence of five symbols by pressing them on the keyboard.

```
7+18=
```

After processing, the calculator will output this sequence of two symbols.

```
25
```

The input is allowed to include the symbol + because it is on the keyboard of the calculator. But many calculators cannot write that + symbol as part of the output. In general, the input and output do not need to use the same symbol sets, or the same *alphabets* technically speaking. Not every input sequence is valid. For instance ++−= will not yield an answer because it does not have the form of a well written arithmetic problem. In other words, it is syntactically incorrect. Even if an input sequence has a well written form, it may be semantically invalid, like `10/0=`. This will result in the calculator displaying an error message, because the rules of division do not allow division by zero.

Semantics are important to the user of a computer system. We know what `7`, `18` and `25` mean; that is why the answer from the calculator is useful to us. But there is no need for the calculator to understand meaning as part of its inner working. And that is fortunate, because regardless of the possibility of building meaning into an electronic device, we definitely did not know how to do it when calculators were invented. Even if it had some kind of understanding, we would not care if the calculator *thought* that `7` means half a dozen, or that `+` means 'apple', as long as the result, `25` is formally correct. The calculator does not work by understanding maths, but by carrying out an algorithm, which is a meaningless mechanical procedure, in the same fashion as we do arithmetic on paper.

So, of every meaning of the word *symbol*, we have chosen the least meaningful one. For our purposes, all that there is to a symbol is that it is distinct from other symbols. Nothing more. This property alone allows a symbol processor to perform processes that are sensitive to which symbols are fed as input and in which order. We can relate this to the meaning of *information* in the classic approach of information theory (Shannon, 1948). In this sense, information is simply something that had the possibility of being something else.

The word *system* is important in *computer system*. It means that we can make a distinction between hardware and software. Software works like a food recipe and hardware works like the sum of the cook and the kitchen. The recipe dictates how to produce a dish, as long as the kitchen has all that is needed. In the same fashion, a computer program dictates how the machine will proceed in order to process input symbols, as long as it has the capabilities needed to execute the instructions in the program. A thermostat for a heating system can be seen as a symbol processor. It is fed temperature difference information, which can be represented by two different input symbols: `hotter-than-desired` and `colder-than-desired`. Corresponding outputs are `heating-off` and `heating-on`, respectively. But the fact that it is a symbol processor does not mean it is a computer system. It is not, because it is not intended to be programmed in order to perform a different function, and therefore there is no hardware/software distinction. We could force the situation by wiring the thermostat backwards, which would produce the ill function of heating until malfunction. But that is as much flexibility as we get. In contrast, a smartphone or a laptop are full blown computer systems, in the sense that they are not hard wired to perform a single function, but are built for following the instructions in a computer program. Application software determines the function that will be performed by the system, as long as it is within its hardware capabilities. In sum, every computer system is a symbol processor, but not every symbol processor is a computer system.

The computer models we will see here relate to that notion of computer system. What we will actually describe is *classes* of computers, which are a theoretical analogue to kinds of hardware. And how we describe them will affect what the capabilities of the class are. But any particular instance within the class, will have a particular *programming*. For example, the first class we will see is that of finite-state machines, which can be used to perform several different functions. Instances of that classes will be theoretical models of an elevator or a vending machine, each of which is programmed in a particular way so to perform one or the other function. The way we decide to wire an instance of a computer class in order to perform a particular function is the analogue of software. Universal Turing machines will provide us with a more useful notion of programmability, since they allow a *program* to be fed as part of their input.

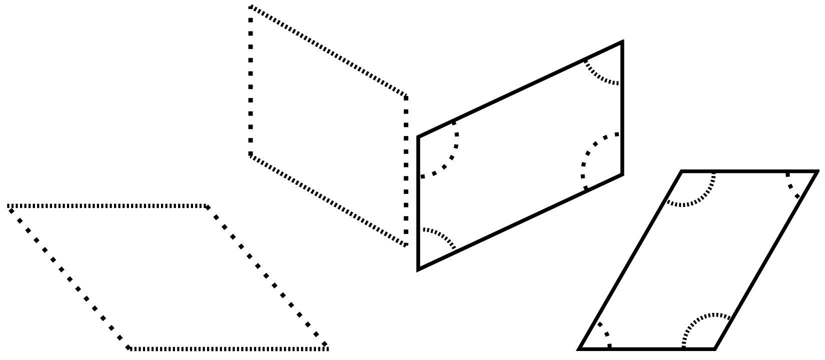
2.1. Multiple definition, multiple implementation, and computational power

We should note that two different definitions can be *equivalent* in the sense that they define the exact same thing. Consider the following example[1]: we will define p-quadrilaterals as quadrilaterals (polygons made up of four sides) with the property that opposite sides in it are parallel (Figure 2.2a). There are also a-quadrilaterals defined by the property that their opposite angles are of the same value (Figure 2.2b). In principle, p-quadrilaterals do not

[1]Thanks to Luis Dissett for sharing this useful example with me.

seem to be the same as a-quadrilaterals, because definitions are differently worded. However, although it may not seem evident, one definition does imply the other. In other words, whenever a quadrilateral satisfies the definition of an a-quadrilateral, it necessarily satisfies that of a p-quadrilateral and *viceversa*. Despite having different definitions, a-quadrilaterals and p-quadrilaterals are the exact same class of polygons. The take-home message is that two classes of objects could actually be the same one, even if they have differently worded definitions, or even if definitions focus on different properties.

FIGURE 2.1. Example of equivalent definitions. The defining property of p-quadrilaterals is having parallel opposite sides, while that of a-quadrilaterals is having same value opposite angles. Despite having differently worded definitions, both define the exact same class of polygons, because one property implies the other.


(A) p-quadrilaterals (B) a-quadrilaterals

A *computer program* is a precise specification of how a symbol processing task is to be effectively carried out. For the purposes of this text, what defines a program is what it does to symbol sequences; in other words, the mathematical function that maps input to output sequences. How that is accomplished is not relevant as long as it can be done. For instance, you could turn right either by turning 90^o to the right or 270^o to the left. This would certainly make an important difference to a robot, because its motion is part of its observable output. But as long as differences are kept internal, not affecting the result, programs can be considered equivalent. Why do we not need to care about internal differences? Because, if CTM results to be limited in the mental capabilities it can explain when we do not care about inner workings, it will still be limited if we include the additional difficulty that internals matter. Efficiency will not be a concern for us. Certainly, turning right is more efficient if we turn 90^o to the right instead of 270^o to the left. How a particular computer problem is solved affects efficiency so much, that an inefficient solution may become impractical. This will not affect our argument, because the kind of limitations we will explore hold even if our computer system had all the speed and all the time in the world. In computer scientific terms, we are concerned with computability, not complexity.

We have emphasized that a computer program will work as long as the computer has all capabilities needed to run it. This is relevant because different models of computation offer different amounts of *computational power*. Contrary to its use in tech advertising, computational power is not about how fast a computer can run, but about what tasks it can accomplish. Model M_2 is computationally more powerful than model M_1 if and only if M_2 can solve all problems that M_1 can, and then some more. Currently, there is no computer model more powerful than what is known as the Turing machine, and there is good reason to believe there will never be.

2.2. Some well known computer models

We will review some computer models, starting from the relatively simple finite-state machine. The recurring scheme will be that each model has some capabilities but also some limitations that can be overcome by adding new features. In the end, we will reach the very powerful Turing machine.

2.2.1. Finite-state machines

One relatively simple theoretical model of computing is the class of *finite-state machines* (FSM, see Hopcroft, Motwani, & Ullman, 2007, ch. 2). The importance of this class is twofold. First, it is useful on its own, since it is a good model for elevators, vending machines, washing machines, and other common apparatuses. Second, it can be used as a building block in more powerful models of computing.

A FSM has a finite set of possible states. For each state, it clearly defines what the next state is as a function of the current state and the input symbol that is currently being

FIGURE 2.3. Simplified finite state machine for two floor elevator. It models how position changes as a result of pressing up or down buttons.

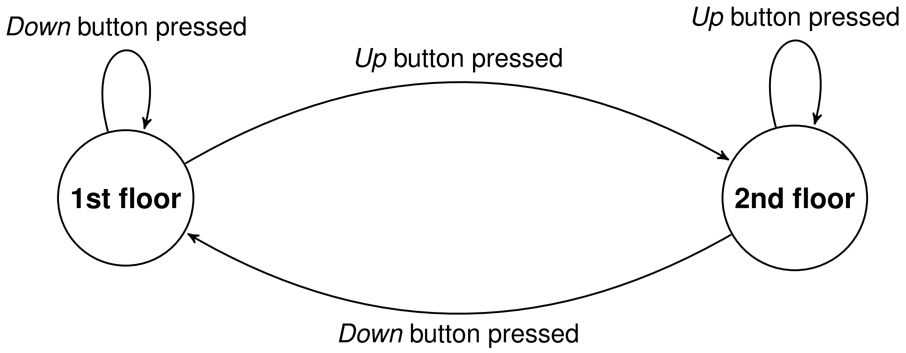


TABLE 2.1. State transition table for the finite-state machine in Figure 2.3. It contains all the information needed to completely define the machine.

Current state	Current input symbol	Next state
1st floor	*Up* button pressed	Go to state: 2nd floor
1st floor	*Down* button pressed	Stay in state: 1st floor
2nd floor	*Up* button pressed	Stay in state: 2nd floor
2nd floor	*Down* button pressed	Go to state: 1st floor

processed. Therefore, a particular instance of a state machine can be defined by a state transition table (see Table 2.1 for an example). Figure 2.3 shows a simplified model for the behaviour of a two floor elevator. It can be summarised as follows: pressing a button results in the elevator moving to the requested floor as long as it is not already there. Figure 2.4 models the rules of a three question quiz TV show in which a wrong answer results in loosing all money.

It is possible to consider the last state of the machine as its output, after it has processed the whole sequence of input symbols. Or, if needed, the full history of the machine can be its output. Figure 2.5 shows an example that takes a sequence of characters as input and outputs whether it includes an even number of ‘a’. Each time any symbol is *read* from the input, a machine state is yielded as output. But only the last state in the output matters, which occurs at the end of the processing.

FIGURE 2.4. Finite state machine for a three question quiz show. States circled in double lines are final. Prize can be up to $3,000 if all three questions are answered correctly. Retiring without an answer yields an amount that depends on the number of previous right answers. A wrong answer finishes the game with no prize.

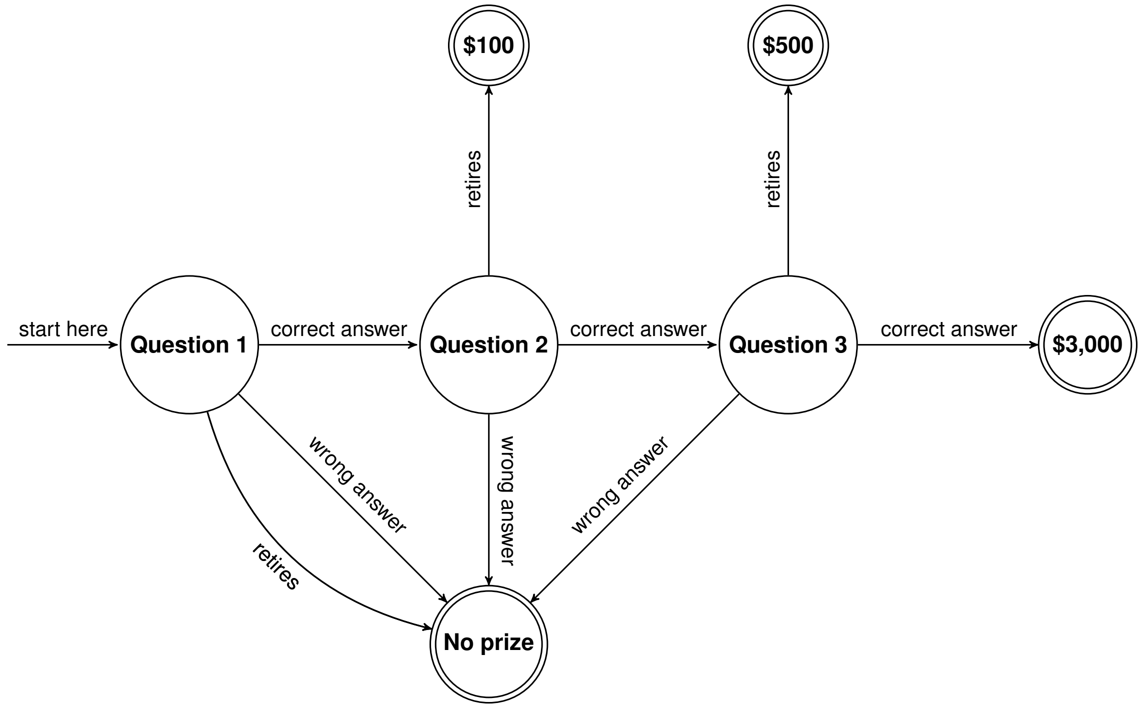
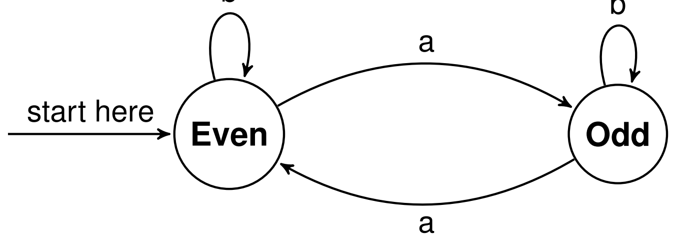
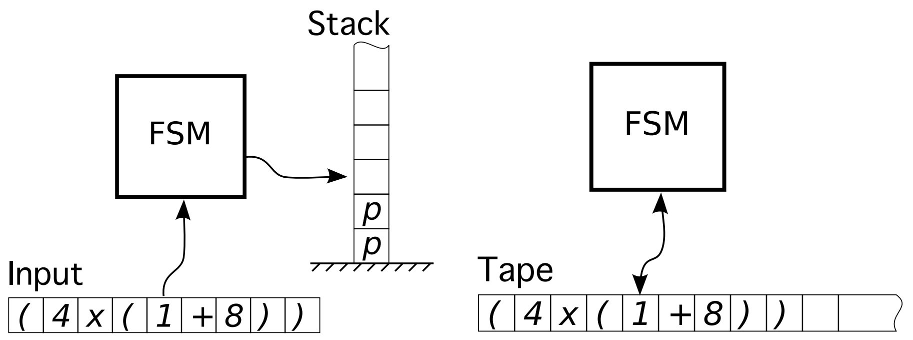


FIGURE 2.5. Finite state machine for telling whether a sequence of characters contains an even amount of 'a' symbols. It has essentially the same form as the elevator model from Figure 2.3. For instance, processing characters "aabaaz" goes through several state transitions (arrows) each one resulting in a new output. The last output indicates that the input has an even number of 'a'.



Sample input: `aabaab`
Output:
Even
Odd
Even
Even
Odd
Even
Even

FIGURE 2.6. Both pushdown automata and Turing machines are the combination of a finite-state machine and some additions. In the case of a PDA the addition is a stack for piling up symbols. Instead of a stack, a Turing machine has a tape and a read/write head that transfers symbols between the tape and the FSM. The tape is infinite, in the sense that there is a beginning to it but no end. In constrast to a PDA, a TM does not need separate input and output, because the input can be written on the tape at the begining, and the output can be read from the same tape after processing has finished.



(A) Pushdown automaton. (B) Turing machine

2.2.2. Pushdown automata

Finite state machines can be very powerful. Among other capabilities, they can detect regular expression patterns (see Hopcroft et al., 2007, ch. 3) which have many important applications in computer science. However, they have important limitations when facing problems that require counting. It is easy for a FSM to determine whether a symbol occurs an even amount of times in some input, as long as there is no need to count exactly how many are there.

One problem that requires actual counting is to check whether an arithmetic expression closes all the parentheses it opens. A FSM for keeping track of at most three open parentheses will need to have at least four different states: 0, 1, 2, and 3 open parentheses. And it will fail with an expression that has four or more parentheses open at a given point. If the maximum number of parentheses is not known beforehand, there is no way a FSM can be built for checking that all parentheses are properly closed.

A *pushdown automaton* (PDA) has no such limitation (see Hopcroft et al., 2007, ch. 6). It corresponds to a FSM with the addition of a stack of symbols. The stack works like a pile of dishes. A symbol can be put (pushed) on top, or removed (popped) from the top. After pushing several symbols; e.g., A, B and C; the first symbol that can be popped is the last one that was pushed (C). This scheme is known as last-in-first-out. Just like the input and the output of a machine, a stack has its own alphabet. The same symbol can appear several times, and there is no limit to how many symbols can be pushed.

In order to check if every open parenthesis is eventually closed, a PDA can use the following strategy. It will add a symbol, 'p' to the top of the stack each time a '(' is read from the input. And it will pop a 'p' from the top of the stack each time a ')' is read. If the input expression properly closes al parentheses it opens, each pushed 'p' will eventually be popped, leaving the stack empty when the input sequence has been completely processed. If the stack is not empty at the end, it means that at least one parenthesis was opened but not closed. If an attempt is made to pop a symbol when the stack is empty, it means that a parenthesis that was not opened is being closed.

Simply adding a stack allows unlimited counting, which opens a whole range of possibilities. A PDA includes a FSM. Therefore, any problem that can be solved with a FSM can also with a PDA. And then, there are additional problems that a PDA can solve and a FSM cannot (unlimited parentheses checking for instance). This means that PDA are computationally more powerful than FSM.

One important application is the kind of processing done by *computer language compilers* (Aho & Ullman, 1977), which are fundamental tools for creating computer software. Their purpose is to translate source code written by a programmer to another language, usually machine code that can be run on the processor of a digital computer system. Despite the fact that PDA have been known for a while (Newell, Shaw, & Simon, 1959) they still play an important role today in software development. This is probably because PDA lend themselves naturally to the processing of formal languages used for computer programming. They also work well for extracting structural information from some natural language patterns (Chomsky & Miller, 1963).

2.2.3. The Turing machine

Despite being powerful, PDA also have some important limitations. Their ability to count without bounds is countered by the fact that they *forget* a count when they use it, because in order to retrieve the count, symbols from the stack have to be popped and discarded one by one. For instance, a parenthesis checking PDA can count opening parenthesis and, then use that count exactly once to match closing parentheses. By the time the machine has confirmed that parentheses were correctly matched, the stack has been emptied and the count has been lost.

An elegant solution is to provide a PDA with a second stack, so that symbols popped from the first one can be stored to the second one instead of being discarded. This allows the automaton not only to give a second use to the count, but to actually use it as many times as desired without limits, switching symbols from one stack to the other. Since an original PDA could not do this, a two-stack PDA is more powerful. It reaches exactly the same computational power as another important computer model: the Turing machine (Turing, 1936, Hopcroft et al., 2007, ch. 8).

A Turing machine (TM) includes a FSM that functions as control device and a tape of infinite length on which symbols can be written and read (see Figure 2.7b). Like rectangle delimited fields in a paper form, the tape in a TM has one position for each symbol. There is a first position, but not a last one, since there is no end to the tape. The included FSM reads and writes symbols through a single head that, at any time, points to some position on the tape. At each step of its operation, the FSM takes the symbol under the head as input. Just like an ordinary FSM, the one inside a TM has its next state fully determined by the current input symbol and the current state. Upon reaching a new state, a symbol and a direction are produced. The symbol is written to the tape, and can even be the same as the one already there. The direction indicates if the head shall move one position to the right on the tape or to the left. By moving one position at a time, the head of a Turing machine has the effective capability of reaching any position of the tape.

Despite being more sophisticated than a FSM, a TM can also be specified by means of a relatively simple table. Table 2.2 shows the specification of the FSM that controls a Turing machine. This particular machine reads a sequence of ceros and ones from its tape. It deletes the ceros and moves the ones to the left. At the end, all the ones are next to each other at the beginning of the tape. For instance, sequence ‘1101001’ becomes ‘1111’ after running the machine. Table 2.3 displays the process.

TABLE 2.2. Example of a Turing machine. The table shows what should be done in the last three columns, as a function of the situation indicated in the first columns. For instance, the first row shall be read as follows: if the machine is in state A and the head is reading a ‘0’ from the tape; then write an ‘x’ to the tape, move the head one step to the right, and make B the new current state.

rule	state	symbol under head	what to write	where to move	next state
R1	A	0	x	right	B
R2	A	1	1	right	A
R3	A	blank	blank	stay	stop
R4	B	0	0	right	B
R5	B	1	0	left	C
R6	B	blank	blank	left	D
R7	C	0	0	left	C
R8	C	x	1	right	A
R9	D	0	blank	left	D
R10	D	x	blank	stay	stop

This is a model of great relevance for at least two reasons. First, attempts to extend its capabilities by including additional devices do not result in additional computational power. This differs from less powerful models like FSMs or PDA which can be made more powerful by adding devices, as we saw before. We can add more heads to the tape of a TM; or even several tapes, each with several heads. Also we can give it non-determinism, which means that given a machine state, the next one is not fully determined, but could be any from a list of possible ones. Regardless of all those capabilities, it is always possible

TABLE 2.3. Execution of the Turing machine of Table 2.2 on the input ‘101001’. At each step, the position of the head corresponds to the underlined symbol. Note that at step 1, the tape hast the input sequence writtin on it. At step 21, the machine stops and the content of the tape corresponds to the output of the process.

step	machine state	tape	applicable rule
1	A	$\underline{1}$101001	R2: write ‘1’, move right, next state A
2	A	1$\underline{1}$01001	R2: write ‘1’, move right, next state A
3	A	11$\underline{0}$1001	R1: write ‘x’, move right, next state B
4	B	11x$\underline{1}$001	R5: write ‘0’, move left, next state C
5	C	11$\underline{\text{x}}$0001	R8: write ‘1’, move right, next state A
6	A	111$\underline{0}$001	R1: write ‘x’, move right, next state B
7	B	111x$\underline{0}$01	R4: write ‘0’, move right, next state B
8	B	111x0$\underline{0}$1	R4: write ‘0’, move right, next state B
9	B	111x00$\underline{1}$	R5: write ‘0’, move left, next state C
10	C	111x0$\underline{0}$0	R7: write ‘0’, move left, next state C
12	C	111x$\underline{0}$00	R7: write ‘0’, move left, next state C
13	C	111$\underline{\text{x}}$000	R8: write ‘1’, move right, next state A
14	A	1111$\underline{0}$00	R1: write ‘x’, move right, next state B
15	B	1111x$\underline{0}$0	R4: write ‘0’, move right, next state B
16	B	1111x0$\underline{0}$	R4: write ‘0’, move right, next state B
17	B	1111x00$\underline{\ }$	R6: write blank, move left, next state D
18	D	1111x0$\underline{0}$	R9: write blank, move left, next state D
19	D	1111x$\underline{0}$	R9: write blank, move left, next state D
20	D	1111$\underline{\text{x}}$	R10: write blank, stay, stop
21	stop	1111$\underline{\ }$	none

to build an ordinary (deterministic, one tape, one head) TM that successfully performs the exact same task on the input (Hopcroft et al., 2007, ch. 8).

In other words, attempts to make a TM better have only resulted in possibly faster or less tape-hungry devices, but in the end, the computational power is no bigger than that of an ordinary TM. This is good enough reason to consider that the power of the TM could be a hard limit, in the sense that a more computationally powerful device may not be possible to build. This conjecture is known as the *Church-Turing thesis* (Kleene, 1943, 1967). Apart from making a TM more complex, other completely different approaches have been taken with the aim of producing very powerful computing models (e.g. Church, 1936). No attempt has surpassed the power of the Turing machine, but very interestingly, some have matched it. It is not possible to prove the Church-Turing thesis, but as we will see in chapters 5 and 4, there is strong support for it and it is widely believed to be true.

2.2.4. Universal Turing machines

A second reason why TMs are important is because they can simulate the execution of another TM. A universal TM (UTM) is designed to read a formal description of a particular TM from its tape, along with some input sequence for it, I. The UTM takes both the description of the TM and its input I as a whole which it swallows as its own input. Then it simulates what would happen if I was fed to the TM. As a result, it outputs what the TM would have written as its own output. The fact that this is possible (Turing, 1936, see Hopcroft et al., 2007, ch. 9) makes the Turing machine a remarkable design. Less formally, Turing machines have the capability of simulating themselves. This is a mathematical basis for the notion of programmable computer system. There is no need to physically build one TM for each single purpose. A UTM is a general purpose computer device that can be fed the description of a single purpose TM (the application software) and data (the input) so that it can run the TM on the input. Just like current digital computers do.

This capability of TMs of simulating themselves adds support to the Church-Turing thesis. For machine X to simulate machine Y, we would expect X to be in a more powerful class than Y. Because, X would be capable of doing all that Y can. And then some more, because it could probably simulate other machines. But when it is a TM that we want to simulate, the "more powerful machine" that is up to the job is just another TM, actually belonging to the same class of computing power. As a consequence, a UTM can simulate another UTM, which is how digital computers can run emulator software in order to behave like other, usually older, digital computers or videogame devices.

2.3. Actual implementation of computing models

The infinite tape of a TM leaves the taste of *fiction* in mouth. There is no such thing as an actual infinite tape. Does that mean that a physical TM cannot be effectively built? Actually it can, if we rely on the following fact. If a TM finishes its task, it never uses an infinite amount of tape in the process. Because if it did, it would need to perform an infinite amount of write operations, and it would never finish. So we do not actually need an infinite amount of tape. We only need the length of the tape not to be limited a priori. For instance, the machine could be designed so that it pauses when it reaches the end of the tape, for us to glue additional length for the work to continue, and everything would be fine.

In practice, we use digital computers with more than enough amounts of memory, so that the limit is never seen. Theoretically, a personal computer has the power of a FSM, because it has a limited amount of memory, unlike PDA or TMs, meaning that it has a finite set of states it can be in. But in practice the number of possible states is unimaginably big. As long as we do not run out of memory, a personal computer will behave as a Turing machine. And when we do run out of memory, we can upgrade it or to buy a whole new computer, which is not that different from glueing additional length at the end of a TM's tape.

A human being could look at the table that specifies a Turing machine and follow the rules in order to perform the same computations on paper. It could be difficult to perform a long computation without making mistakes. But it can be done, specially if redundant work is done by several people for checking. If we look at the human mind from a computational angle, we may question if it is more powerful than a TM. But one thing is sure. It is not less powerful.

Can we effectively implement more powerful systems? Hypercomputation is a research area that explores computing models with more power than a TM. It originates on Turing's (1939) idea of adding an oracle device to a TM. This oracle is a sort of black box that computes functions that a TM cannot, like predicting if a machine will halt. We currently do not know if such oracles can be physically implemented. If we believe the Church-Turing thesis, we must conclude that what the oracle computes cannot be mechanically computed, and then the oracle machine cannot be built. But Turing used this theoretical construct to prove that even with the help of an oracle, a TM would still have a limited computing power. Therefore, despite the skepticism of some authors about the Church-Turing thesis (Gandy, 1980; Ord, 2002), the fact remains that computer power is limited.

Other way to transcend the power of a TM would be to stretch time, in order to perform an infinite amount of operations without having to wait forever. This is the approach of accelerated Turing machines (Copeland, 2002). Quantum computing holds the promise of accelerating the execution of TMs. Some (but not all) processes that currently require exponential amounts of operations with respect to the size of the problem could be solved in polynomial time (see Bennett, Bernstein, Brassard, & Vazirani, 1997). This would be a very good thing, since in actual computing exponential time usually means *impractical*, while polynomial time usually means *doable*. It would also be very bad for banking security, which relies on the fact that cracking their cryptography is currently believed to require exponential time. In any case, turning infinite time into finite time would be harder than turning exponential time into polynomial time, if achievable at all (Hodges, 2005).

There are several reasons to consider the Turing machine as the *most powerful computing model* that we need in order to understand what it is to model the mind as a computer. In practice, it is the most powerful model that has been physically implemented, and there is good reason to think there will never be a more powerful one. Skeptics about the Church-Turing thesis have not been able to implement a more powerful system, and their theoretical models of choice often rely on non-existing capabilities like oracles or the execution of an infinite amount of operations in finite time.

Even if the TM was not the limit, it is a fact that there is a limit. Not only did Turing (1939) prove that oracle machines still have limitations. As we will see in chapter 3, there are as many mathematical functions that map input sequences to output sequences as there are real numbers. But there are as many formal descriptions of computing procedures as there are natural numbers. And we will see that there are much more real numbers than natural numbers. Therefore, there are much more functions than it is possible to describe machines for computing them. Consequently, if some model had the power to compute *any* possible mapping from inputs to outputs, it would have to transcend the definition of a machine that can be formally described, and therefore, would hardly correspond to the concept of *computer*. This is the notion that the Church-Turing thesis captures.

Perhaps the most important reason to stick with the TM for the purposes of this work is the links it has to logic and axiomatic systems, which will be clarified in chapter 5. Science is transparent, in the sense that theoretical models and findings can be shared, explained and substantiated. Arguments need to have steps, and the transition from each step to the next has to be explicable with more than a leap of faith, except for axioms with very strong intuitive support. Ultimately, a theory needs to make predictions. Otherwise, it would not be possible to check if it matches observations, or to challenge it, making it useless for science. If a model of a mental phenomenon transcended the power of TMs, its behavior would go beyond the computable, and we would have to do without the notion of predictions.

3. COUNTABLE AND UNCOUNTABLE SETS

Turing halting and Gödel incompleteness are the next stops in our journey. The former is an important example of a problem that is clearly and formally well defined, that definitively has an answer, but that answer cannot be computed. The latter shows a limitation of axiomatic (formal-logic) systems. Undecidability of Turing halting is a stronger and more recent result than Gödel incompleteness. Both are intimately related, but as usual, this is not evident at first. We can still benefit from reviewing Gödel's work, because it helps with understanding the limitations of TMs and it will shift our focus from calculators to theorization, which is a central topic to this work.

Before we go into those two topics, it is beneficial to talk about the cardinality of infinite mathematical sets. The notion that there are different *kinds of infinite* is important in order to understand why formalisms that can literally solve infinite different problems, cannot solve all problems. We will talk about countable and uncountable sets, of which natural numbers and real numbers are examples, respectively. Both are infinite. But one is much bigger than the other.

The fundamental reason why computational power has a limit can be stated as follows. There are uncountable ways to map input to output sequences. But there are only countable ways to describe a machine. Therefore, there are countless input-output mappings for which a machine cannot be described. However, let us not get ahead of ourselves.

3.1. How to define a mathematical set

What defines a mathematical set is what belongs in it. We can specify that by giving a full list of every object it includes. Unfortunately this does not work for sets that have too many or an infinite amount of objects in them. For those cases, we need a logical expression that is only true of every object that belongs in the set. With that kind of definition the cardinality of the set, which is the amount of objects it includes, is not always clear. The set of "the 10 fastest runners in the World" clearly has a cardinality of 10. But finding the cardinality of the set of "runnners that can run 100 meters in 11 seconds or less" requires some research.

Sometimes it is not evident if there is even a limit to the amount of objects in a set. For instance, how many *prime numbers* are there? In other words, how many objects are included in the set of prime numbers? Remember that a prime number is a positive integer number that has exactly two positive divisors[1]: 1 and itself. Which is the biggest prime number? There is no such thing, because it is always possible to find a bigger prime. Euclid's well known proof (Euclid & Williamson, 1788, book IX, proposition 20) provides the following argument. Take any finite set of prime numbers S. Then find number N which is the product of all numbers in S plus 1.

$$N = \prod_{p \in S} p + 1$$

This number can be used to prove that there is at least one prime number that is not in S. If N is prime, then trivially N is a prime number bigger than any other in S. But if N is *composite* (i.e., not prime), that still means that there is a prime number not included in S. Being a composite number it must be possible to express N as the product of prime numbers. But none of those prime numbers can be in S, because dividing N by any prime in S will yield the product of the other numbers in S as result, leaving a reminder of 1. The conclusion is that given any finite set of prime numbers there is always at least one prime number that is not included. Since no finite set includes all prime numbers, the set of prime numbers is infinite.

While the information is there in the definition of prime numbers, a mathematical proof was required in order to find the cardinality of the set. It gets more interesting when we realize that not all infinite sets have the same cardinality.

[1] D is a divisor of N if the division N/D leaves no reminder

3.2. The cardinality of infinite sets

3.2.1. Natural numbers

Natural numbers $\mathbb{N}$ are those we use to count. A fundamental concept in them is that of *successor*. Every natural number n is followed by its only successor which we call $S(n)$. And that successor $S(n)$ is followed by another one $S(S(n))$ and so on. This can go on forever as long as there is a beginning. There has to be a first natural number. Often 0 is considered to be the first natural number[2]. What identifies a natural number is how many times the successor operator has to be applied to reach it. For instance, number 3 is different and actually bigger than number 2, because they are built as $S(S(S(0)))$ and $S(S(0))$ respectively.

This two principles, that there is a first natural number, and that every natural number has a bigger successor, have the consequence that there are infinite natural numbers. If there was a finite set containing all natural numbers, then a logical contradiction would occur, because taking the biggest number in that set and finding its successor (which always exists) would produce an even bigger natural number, that therefore cannot be in the original set, meaning that it did not include all natural numbers in the first place.

Whether we start counting from 0 or 1 does not make a difference to natural numbers in terms of cardinality. It is just a naming issue. Replace 1 for 0, 2 for 1, 3 for 2, etc., and since there are always more natural numbers available, there is the same amount of them regardless of the name of the first one.

The set of natural numbers is big enough to include an infinite amount of elements, but small enough for its elements to be countable. That means that there is a strategy for visiting each of them, one after the other, without leaving a single one out. It is true that we will never have enough time to count them all. But it must be noted that any natural number, no matter how big, will eventually be counted if enough time is given, and that

[2]Some mathematicians start natural numbers at 1. Whenever there is a need to avoid ambiguity, *non-negative integer numbers* is used if 0 is to be included, or *positive integer numbers* if not.

amount of time is always finite. The cardinality of $\mathbb{N}$ is often called $\aleph_0$ and is the "smallest" infinite set cardinality. Sets with bigger cardinality do exist, as we will see.

3.2.2. Some subsets of natural numbers

Infinite cardinalities are not always intuitive. For instance, consider a set T which includes every natural number and also the irrational number $\pi = 3.14159265...$ Since T has every object in $\mathbb{N}$ and then an additional one, intuition would suggest that the cardinality of T is bigger than the cardinality of $\mathbb{N}$. However we must remember that cardinality is not about which elements are in a set, but about how many. A single additional element makes no difference to an infinite cardinality, just like it made no difference whether 0 was included in $\mathbb{N}$.

Consider sets $X = \{\bigcirc, \triangle, \bigtriangledown\}$, $Y = \{\pentagon, \hexagon, \square\}$. The fact that X has objects not in Y does not mean that X has more objects. In order to find whether two sets have the same cardinality we need a criterion that is not based on object identities. Two sets, A and B have the same cardinality if and only if there is a way to match objects in A with objects in B in pairs, with no object of A or B left out. Such a pairing is possible in the previous example; e.g,., $\bigcirc$–$\pentagon$, $\triangle$–$\hexagon$, $\bigtriangledown$–$\square$. But it would not be possible to match X to $\mathbb{N}$, because only three natural numbers could be matched by shapes in X, leaving infinite natural numbers out of the matching. This means that $\mathbb{N}$ has bigger cardinality than X.

Removing a finite amount of numbers from $\mathbb{N}$ does not alter its cardinality. Lets call the resulting set M. Note that the infinite natural numbers that remain in M can be counted. That means they can be matched to natural numbers in pairs without leaving any number out, either from M or $\mathbb{N}$. M is a strict subset of $\mathbb{N}$ but that does not mean its cardinality is smaller, which would be the case if we were talking about finite sets. M has the same cardinality as $\mathbb{N}$. The same happens when adding a finite amount of numbers instead of removing. Therefore, when infinite cardinalities are involved, it is possible for a set and a strict subset of it to have the same cardinality.

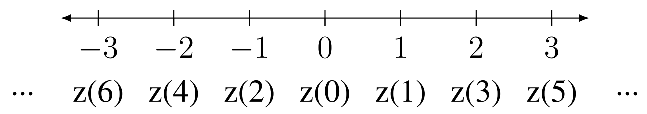


FIGURE 3.1. Matching of integer numbers and natural numbers on the number line.

In a similar fashion, adding or removing an infinite but countable amount of numbers from a countable set does not necessarily alter its cardinality. An exception would be to remove every number in the set, leaving an empty set. For instance, consider positive even numbers. We can map all of them to natural numbers as follows. To every natural number n, there is a positive even number $m = 2 \cdot n$. Since there is a one to one mapping, natural numbers and positive even numbers have the same cardinality. A similar argument can be used to show that there are as many odd numbers as natural numbers.

3.2.3. Integer numbers

Integer numbers, denoted by $\mathbb{Z}$, add zero and negative numbers to $\mathbb{N}$. It turns out that every number z in $\mathbb{Z}$ can be matched by a number n in $\mathbb{N}$ by means of the following mathematical function.

$$z(n) = \begin{cases} -n/2 & \text{if } n \text{ is even} \\ (n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

It works like this for the first six natural numbers.

$$\begin{aligned} z(0) &= 0 \\ z(1) &= 1 \\ z(2) &= -1 \\ z(3) &= 2 \\ z(4) &= -2 \\ z(5) &= 3 \\ z(6) &= -3 \end{aligned}$$

and so on...

Figure 3.1 shows a graphical representation of the mapping. By plugging all natural numbers into function $z(n)$, all integer numbers can be generated. Odd naturals match positive integers, and even naturals match the rest (negative and zero). It also works the other way round. Each number n in $\mathbb{N}$ can be matched by a number z in $\mathbb{Z}$.

$$n(z) = \begin{cases} z \cdot 2 - 1 & \text{if } z \text{ is positive} \\ -z \cdot 2 & \text{if } z \text{ otherwise} \end{cases}$$

In this case, positive integers generate odd naturals, and the remaining integers generate even naturals. The result is that $\mathbb{Z}$ has the same cardinality as $\mathbb{N}$.

3.2.4. Rational numbers

Rational numbers $\mathbb{Q}$ are those that can be expressed as a fraction of two integer numbers (e.g., $\frac{1}{2}$, $\frac{3}{4}$, $\frac{-3}{5}$, $\frac{13}{7}$, etc.). Of the two integers, the one at the top is called *numerator* and the other *denominator*. An alternative writing for rational numbers is an integer followed by decimal digits (e.g., $\frac{3}{4} = 0.75$). Note that some numbers require an unlimited amount of decimal digits, like $\frac{1}{3} = 0.333333...$. Some numbers (like $\pi = 3.14159265...$) cannot be written as a division, meaning that they are not rational numbers. Rational numbers are useful to measure amounts in daily life, like distances, weights, etc. Intuition would suggest that there are more rational than natural numbers, because $\mathbb{N}$ is a strict subset of $\mathbb{Q}$. But we already discussed that this argument is misleading. We know that $\mathbb{Q}$ cannot be *less* than $\mathbb{N}$. But are they more? The question becomes, are rational numbers countable?

Actually, there is a strategy to count rational numbers. Consider figure 3.2. It is infinite to the right and to the bottom, and it includes every positive rational number. The table displays arrows showing an order for counting all rational numbers without leaving out a single one. With that strategy, some will be counted more than once. For instance, $\frac{1}{1}$ and $\frac{2}{2}$ are the same number. This can be solved simply by ignoring numbers that have already appeared in the count. Those are only positive rational numbers. We could make a similar table for negative rational numbers. Since positive and negative rational numbers can be

FIGURE 3.2. Countability of rational numbers. This array includes all positive rational numbers (some appear more than once). It extends infinitely to the right and to the bottom. In order to build it, each number in a cell has the row number as numerator and the column number as denominator. All numbers in the table can be counted, starting from $1/1$.

$$\begin{tikzcd}
\frac{1}{1} \arrow[d] & \frac{1}{2} \arrow[r] & \frac{1}{3} \arrow[dl] & \frac{1}{4} \arrow[r] & \frac{1}{5} \arrow[dl] & \frac{1}{6} \arrow[r] & \frac{1}{7} \arrow[dl] \\
\frac{2}{1} \arrow[ur] & \frac{2}{2} \arrow[dl] & \frac{2}{3} \arrow[ur] & \frac{2}{4} \arrow[dl] & \frac{2}{5} \arrow[ur] & \frac{2}{6} \arrow[dl] & \\
\frac{3}{1} \arrow[d] & \frac{3}{2} \arrow[ur] & \frac{3}{3} \arrow[dl] & \frac{3}{4} \arrow[ur] & \frac{3}{5} \arrow[dl] & & \\
\frac{4}{1} \arrow[ur] & \frac{4}{2} \arrow[dl] & \frac{4}{3} \arrow[ur] & \frac{4}{4} \arrow[dl] & & & \\
\frac{5}{1} \arrow[d] & \frac{5}{2} \arrow[ur] & \frac{5}{3} \arrow[dl] & & & & \\
\frac{6}{1} \arrow[ur] & \frac{6}{2} \arrow[dl] & & & & & \\
\frac{7}{1} & & & & & &
\end{tikzcd}$$

counted, the union of both sets with the addition of zero is also countable. This follows from the same arguments used in Section 3.2.2. As a result $\mathbb{Q}$ has the same cardinality as $\mathbb{N}$.

3.2.5. Real numbers

We have reviewed different infinite sets, and all resulted to have the same cardinality $\aleph_0$, which is the cardinality of $\mathbb{N}$. But bigger cardinalities are also possible. Cantor (1874), proved that real numbers $\mathbb{R}$ are not countable, and therefore have a bigger cardinality than natural numbers. That proof rests on concepts beyond the scope of this text. However, he later published an article using a *diagonal argument* (Cantor, 1891), which is a simpler strategy for proving that $\mathbb{R}$ is not countable. The same argument is also at the core of Gödel's incompleteness theorems, and Turing halting undecidability, which in turn are fundamental for understanding the limitations of logic and computing.

TABLE 3.1. Uncountability of real numbers. If real numbers between 0 and 1 were countable, they could be displayed on a table like this, with each natural number associated to a real number, and not leaving out a single real number. Each number can be written as a 0 followed by a decimal separator and an infinite series of digits d_{ij} in which i is the row and j is the column.

Natural number	Corresponding real number
0	$0\ .\ \mathbf{d_{00}}\ d_{01}\ d_{02}\ d_{03}\ d_{04}\ d_{05}\ d_{06}\ d_{07}$ ...
1	$0\ .\ d_{10}\ \mathbf{d_{11}}\ d_{12}\ d_{13}\ d_{14}\ d_{15}\ d_{16}\ d_{17}$ ...
2	$0\ .\ d_{20}\ d_{21}\ \mathbf{d_{22}}\ d_{23}\ d_{24}\ d_{25}\ d_{26}\ d_{27}$ ...
3	$0\ .\ d_{30}\ d_{31}\ d_{32}\ \mathbf{d_{33}}\ d_{34}\ d_{35}\ d_{36}\ d_{37}$ ...
4	$0\ .\ d_{40}\ d_{41}\ d_{42}\ d_{43}\ \mathbf{d_{44}}\ d_{45}\ d_{46}\ d_{47}$ ...
5	$0\ .\ d_{50}\ d_{51}\ d_{52}\ d_{53}\ d_{54}\ \mathbf{d_{55}}\ d_{56}\ d_{57}$ ...
6	$0\ .\ d_{60}\ d_{61}\ d_{62}\ d_{63}\ d_{64}\ d_{65}\ \mathbf{d_{66}}\ d_{67}$ ...
7	$0\ .\ d_{70}\ d_{71}\ d_{72}\ d_{73}\ d_{74}\ d_{75}\ d_{76}\ \mathbf{d_{77}}$ ...
...	...

The argument follows a *reductio ad absurdum* scheme. Assume that real numbers between 0 and 1 are countable. Then it is possible to build Table 3.1, in which each natural number appears next to a corresponding real number. Look at digits in boldface, which correspond to a diagonal line across digits. We can use those digits to build a new number:

$$D = 0\ .\ \mathbf{d_{00}\ d_{11}\ d_{22}\ d_{33}\ d_{44}\ d_{55}\ d_{66}\ d_{77}}\ ...$$

Replace each digit in this number for a different one in order to build another number D'.

$$D' = 0\ .\ \mathbf{d'_{00}}\ \mathbf{d'_{11}}\ \mathbf{d'_{22}}\ \mathbf{d'_{33}}\ \mathbf{d'_{44}}\ \mathbf{d'_{55}}\ \mathbf{d'_{66}}\ \mathbf{d'_{77}}\ ...$$

D' is not in the table, because its first digit differs from the first digit of the first number. The second digit differs from the second digit of the second number. And so on. It differs from every number in the table in at least one digit. But D' is a real number between 0 and 1. This generates a contradiction, because we assumed the table already had every real number between 0 and 1. No matter how we try to match numbers, there will always be at least one real number between 0 and 1 left out. The conclusion is that the matching was not possible in the first place. This means the two sets cannot have the same cardinality. What is being left out is real numbers, not natural numbers. In consequence, the cardinality of real numbers between 0 and 1 is bigger than that of $\mathbb{N}$. Consequently, the whole of $\mathbb{R}$ has bigger cardinality than $\mathbb{N}$.

Cantor's theorem states that the cardinality of a set A is strictly less than the cardinality of its power set 2^A. The power set of A is the set of all possible subsets of A. For instance, the power set of $S = \{1, 2, 3\}$ is $2^S = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. Cantor's theorem also works for infinite sets. So $2^{\mathbb{N}}$ is bigger than $\mathbb{N}$, $2^{2^{\mathbb{N}}}$ is bigger than $2^{\mathbb{N}}$, and so on.

3.3. Countability of computational procedures

Computational procedures (i.e., algorithms) are infinite but countable. The universal Turing machine (UTM, see section 2.2.4) is useful in order to see that. Remember that we could write the description of a Turing machine (TM) on the tape of a UTM along with input data. In other words, a description of a TM is a symbol sequence. We can enumerate every possible sequence of that kind, and therefore every possible TM. In order to do that, we should first count every possible symbol sequence of length 1 (only one symbol). Then by combining every possible symbol with every sequence of length 1, we get every possible sequence of length 2. Combinations of every possible symbol with sequences of length 2 yield every possible sequence of length 3. We can continue this strategy in order to generate every possible symbol sequence of any possible length. And we can assign a new natural number to each new sequence we generate in the process. The previous argument is implicitly based on the Church-Turing thesis, because we are assuming that every effective computational procedure has a TM that performs it. But what if we are skeptical about that? The same argument not only works for descriptions of TMs, but also for any description that is written as a finite sequence of symbols, where symbols are taken from a finite alphabet[3].

On the other hand, the number of possible mathematical functions of integer numbers is uncountable. This is already true for functions f that take a natural number as argument and output either 0 or 1:

$$f : \mathbb{N} \to \{0, 1\}$$

In order to see this, consider that to any such function, a subset S_f of $\mathbb{N}$ corresponds: the set of natural numbers n for which $f(n)$ is 1.

$$S_f = \{n \in \mathbb{N} \mid f(n) = 1\}$$

And to each subset S_f there is a corresponding function f. So there are as many of such functions as subsets of $\mathbb{N}$. Let F be the set of binary functions of integer numbers.

$$F = \{f \mid f : \mathbb{N} \to \{0, 1\}\}$$

and $2^{\mathbb{N}}$ the power set of $\mathbb{N}$ (the set of all subsets of $\mathbb{N}$)

$$2^{\mathbb{N}} = \{S \mid S \subset \mathbb{N}\}$$

What we have found is that the cardinality of $2^{\mathbb{N}}$ and F is the same. From Cantor's theorem (see section 3.2.5), the cardinality of $2^{\mathbb{N}}$ is strictly bigger than that of $\mathbb{N}$, and $2^{\mathbb{N}}$ is uncountably infinite. The conclusion is that F is uncountably infinite: binary functions of natural numbers are uncountable. And since such functions are just a subset of integer functions,

[3]In a formal language, the alphabet is the set of symbols used for writing symbol sequences (strings).

the latter are also uncountable. In summary, algorithms are countable, but functions of integer numbers are not.

4. TURING HALTING

Imagine you requested a calculation from a computer machine. It is a very difficult one, so the machine is taking some time. After a few hours there is no answer. You better let it run for the night, hoping to have a result tomorrow. But the next day there is still no result. The machine is working intensely. Maybe the machine will never stop. Is it worth letting it run for another day?

Turing halting problem is about determining if a machine will ever stop for a given input. Some tasks are particularly sensitive to this issue. For instance, a machine built for scanning signals from the sky in search of extraterrestrial intelligent life (ETIL) has the possibility of producing a positive result, if such a signal is ever found. But it would not be capable of producing a negative result. Failure to produce a positive result must not be interpreted as a negative result. It could be that there is no ETIL. Or it could be that there is and we have not been able to find it, so the scanning must go on. If we were talking about a problem that can be formally written for a UTM, we would call it a *semidecidable problem*. Theoretically, there are two possible outcomes. But only one result can inform what the outcome is. The alternative result leaves us with nothing but uncertainty.

In a more general way, a *total function* produces an output for any valid input. In contrast, a *partial function* only provides an output for some input instances. Modeling the EITL scanning machine as a mathematical function clearly corresponds to a case of a partial function (of signals from the sky). Other situations are not as clear. In general, it is not easy to know if a total function can be found for solving a given problem.

We described the Turing machine (TM) model in section 2.2.3. In brief, it is the combination of a finite-state machine, and an endless tape with a head that can read and write symbols on it. Among physically realizable computing models, no other has surpassed its power. A TM can be specified by means of a table (see table 2.2 for an example). Also, there are as many TMs as there are natural numbers. We will use that fact during this chapter: a TM can be identified by the natural number that corresponds to it, as long as we agree on a strategy for numbering TMs. It is also possible to identify a TM with a sequence of symbols, which works as a formal description that can be fed as input to a universal Turing machine (UTM). A UTM is a TM with the capability of simulating any other TM that is formally described to it (see section 2.2.4). The fact that UTMs exist, speaks eloquently of the power of TMs. But now it is time to speak about their limitations

4.1. The Turing halting problem

The TM model was introduced to show that there are *undecidable* problems in mathematics (Turing, 1936). This means that some mathematical problems, despite being stated in a clear and perfectly rigorous formal language, simply do not have and will never have a solution. The Turing halting problem is about finding a function that tells if some TM, which we will call M, halts when its tape has input I initially written on it. Formally, the Turing halting function h looks like this:

$$h(M, I) = \begin{cases} 1 & \text{if machine } M \text{ halts on input } I \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Turing elegantly proved that a computable function that satisfies that specification cannot exist. In conclusion, there is at least one mathematical function of natural numbers that can be formally specified in a rigorous fashion, but cannot be computed effectively.

4.1.1. Undecidability proof

Given some computer program[1] M and input I, an actual implementation of h shall analyze those data and then output 1 if the result is that M will halt on I. The other possible case is that M would not halt on I, in which case the output shall be 0. We expect this to work and effectively give an answer for any program/input combination. In other words, h shall be a total function of integer numbers. In a *reductio ad absurdum* scheme, we will

[1]That is, a TM formal description for a UTM.

assume that h exists. Consider the following partial function, based on the total function h.

$$g(x) = \begin{cases} 0 & \text{if } h(x,x) = 0, \\ \uparrow & \text{otherwise} \end{cases}$$

Here the $\uparrow$ symbol means *undefined* which repesents not giving an answer. An actual computer implementation of g would first calculate $h(x, x)$. Since h is a total function, we have no risk of waiting forever, even if it takes some time. If the result was $h(x, x) = 0$, then g must also output 0 by its definition. But in the case of $h(x, x) = 1$ the function g is designed to not give an answer. In order to do so, the implementation can enter an infinite loop.

If h could be actually implemented as a computer program, then a program for g could definitely be implemented, since the latter is a simple construction that adds an infinite loop on top of the former. Now we can sketch Turing's proof. Let e be the computer program (represented by its corresponding integer number) that implements g. Does the program e halt on input e? In other words is it true that $h(e, e) = 1$? There are only two possible cases.

- If $h(e, e) = 0$ then by definition $g(e)$ must output 0. But this means that g produces an actual output on argument e, instead of working forever. In other words, program e halts on input e. This means, by definition, that $h(e, e) = 1$, which contradicts the initial statement that $h(e, e) = 0$. Therefore this cannot be the case.
- If $h(e, e) = 1$ then $g(e)$ must be undefined, which means program e never halts on input e. As consequence $h(e, e) = 0$, which leads to a contradiction again.

Consequently, function h cannot exist. The Turing halting problem is unsolvable. There is a naive point of view from which this could be a trivially expected result. A simple solution attempt would be to use something similar to a UTM (see section 2.2.4) to run a simulation of M on I. If M halts on I, the UTM simulation will eventually end. Otherwise, the simulation will run forever. We would never know if the simulation just needs a little more time or if it will actually never end. Is this why the Turing halting problem is not decidable? Not quite, since running a simulation may not be the only way to tell if the machine halts. Perhaps, by analyzing a description of the machine and its input it could be possible to determine the halting result. But this strategy will not work either. Turing proved that h cannot exist, regardless of whether we use a naive strategy or any other to compute it.

4.2. Computable numbers

Since there are more real numbers than computing procedures (of which there are as many as natural numbers), it follows that some real numbers can have their digits calculated and others cannot. A computable number is one that can be calculated to any desired precision, which means any desired digit of the number can be found computationally. In other word, a number is computable if a there is a computable function that computes its digits. *One third in decimal notation* is a computable number, which happens to be the digit zero, followed by a decimal separator and then the digit 3 repeated forever ($0.3333333...$). Ask me any digit of that number and I can tell you what it is. I can do that mechanically by simply following algorithm 1.

Algorithm 1 Computes a digit of $1/3$ at the given position (negative positions are before the decimal separator)

```
procedure DIGIT(position)
    if $position < 0$ then
        return 0
    else
        return 3
```

Note that we do not need to fully know an algorithm for a number in order to know it exists. For instance, how many human pregnancies have occurred during 19th century? It would be difficult to accurately find that number, but it exists and it is an integer number. So regardless of which number it is, an algorithm can be build that tells any desired digit of it. Certainly we do not know the number, and therefore we do not know which of the many algorithms we could build is the right one. In that sense we know the number is computable, but we do not know which algorithm computes it. This does not make it less computable.

Not only integer or rational numbers can be computed. For instance, the famous irrational number π is computable, because a computational method exists for finding any desired digit of it (Berggren, Borwein, & Borwein, 2004). Intuition would suggest that any number is computable as long as an unambiguous definition of it is given. Turing (Turing, 1936) used Cantor diagonalization to show that at least one definable number is not computable.

The issue of uncomputable numbers gives us a taste of how big the problem is. It is not only that we cannot compute the Turing halting function. Turing's proof opens a Pandora's box full of an infinite amount of undecidable problems.

4.3. Consequences

4.3.1. A few undecidable problems

One important application of Turing halting is proving that other problems are also undecidable by the strategy of reduction. If a solution to a problem P would allow us to build a solution to the Turing halting problem, then we know that P must not have solution, because Turing halting cannot be solved. In other words, if Turing halting can be reduced to another problem, then the latter is undecidable. Here are some undecidable problems of Turing machines and proof sketches.

(i) *General halting*: given a TM, does it always halt or is there any input sequence that will make it run forever?
Consider machine M' that halts on any input if and only if machine M halts on input I. How do we built M'? We program it so that it first erases whatever input was written on the tape, then it writes I, and moves the head to the beginning of the tape; the rest of the machine is a copy of the functionality of M', so that if M halts on I, M' will halt. If the problem was decidable we could apply its

solution to M' in order to decide if M halts on input I. But if we could decide if M halts on I, that would be a solution for Turing halting.

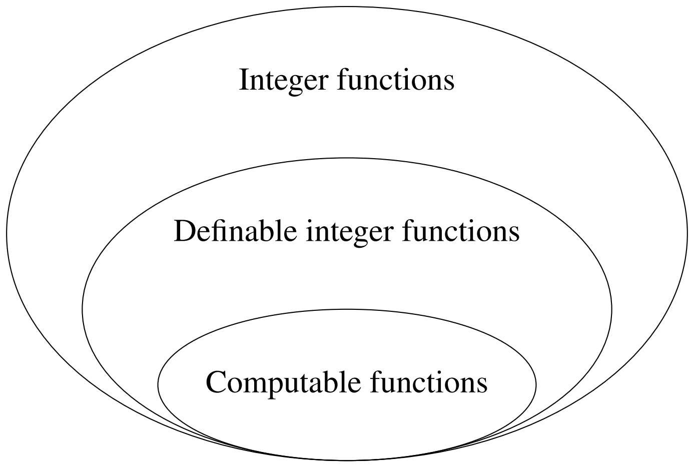
(ii) *Non-emptiness of halting*: given a TM, does it halt for any input at all?
Consider machine M' that halts on input I if machine M halts on I, and halts on no input if M does not halt on I. In order to built M' we program it to check if the input written on the tape is I. If it is, it takes the head to the beginning and follows the program of M. Otherwise it enters an infinite loop. If the problem was decidable, after applying its solution to M' we would have a solution for Turing halting.

(iii) *Finiteness of halting*: given a TM, does it only halt for a finite set of inputs, or does it for infinitely many different input sequences?
We use the same machine M' that we used for general halting. If machine M halts on input I, then M' halts on each one of the infinite possible input sequences. Then by applying this problem's solution to M' we would have a solution for Turing halting.

4.3.2. Undecidability of Turing machine equivalence

We must give special attention to the problem of testing if two Turing machines are equivalent in the sense that given the same input they both give the same output. Just as we did in the previous section, we sketch the proof that this problem is undecidable by reducing Turing halting to it. If we could test two Turing machines for equivalence, then we could test any given machine M for equivalence with machine M_0, that never halts because its program is no more than entering an infinite loop. This means that we would have a solution for testing if machine M halts for any input, which is the non-emptiness of halting problem. As we saw in previous section, if we had a solution for the non-emptiness of halting problem, we would have a solution for deciding the Turing halting problem. And since the Turing halting problem is undecidable, the machine equivalence problem must be undecidable.

FIGURE 4.1. Computable functions are definable integer functions, which in turn are integer functions. In the opposite direction, there are integer functions that are not definable. Also, Turing (1936) proved that there are definable functions that are not computable.



This fundamental result is closely related to another result from Turing's thesis advisor, Alonzo Church, that is known as *undecidability of lambda equivalence*. Actually, Church's was the first undecidable problem to be discovered, just before Turing halting. Lambda calculus is a relatively simple formalism for computation that is based on functions, and has the same power as Turing machines (Turing, 1937). Given two expressions of lambda calculus, there is no general solution for testing if they compute the same mathematical function (1936).

4.3.3. Not all definable functions are computable

We already knew that there are more integer functions than can be finitely defined (see section 3.3). That is because definitions are countable but integer functions are not. We also knew that not all integer functions can be computed, because descriptions of computer machines are also countable, which means there are not different machines to cover them. We did not need Turing halting to know that.

But until now this was not a problem *per se*. Maybe functions that cannot be defined are useless. Maybe functions that cannot be computed would be bizarre things that we would not be interested in. Maybe functions that can be computed are exactly those that can be defined. After all, we are saying that we have *no language* to define those functions. But there is still an infinite amount of functions that *can* be defined. Maybe definable ones are about things that can occur in reality, and undefinable ones are just contradictory, weird mathematical paradoxes that have no real world correlate.

Turing halting tells us that there is an actual problem. First, it shows us that the class of definable functions is not the same as the class of computable functions (see figure 4.1). The Turing halting function h can be precisely defined, but cannot be computed. Just like even numbers are a subset of integers, but both sets have the same cardinality (section 3.2.3); computable and definable functions exist in the same countably infinite amount, but the former are a strict subset of the latter. All computable functions are definable, because computer programs that calculate them work as a definitions. But it does not hold the other way round: not all definable functions are computable.

Another lesson from Turing halting undecidability is that functions that cannot be computed are not necessarily paradoxical entities unrelated to real world issues. Within computer science, Turing halting is already a relevant issue. We wish we could provide computer machines with the means to detect dead ended processes, so we did not waste resources on them. But the issue transcends computer science. The machine with its input, as an extended object in the world, is either a halting or a non halting device. The fact that there is no general way to know which it is, even if we know everything that determines it (the program and the input), means that there is an unsurmountable gap between what the machine *is* and what we can *know* about it. In other words, Turing halting is an example of a formal-logic expression that interrogates about a real world issue, but cannot be answered with formal-logic rigor, even when in possession of the whole data that determines the answer.

5. GÖDEL'S INCOMPLETENESS

Gödel's incompleteness theorems are intimately linked to Turing halting. In principle, we would not need to review them, because Turing's results are stronger. However, as we have already mentioned, this kind of problems can be worded in very dissimilar ways. This time we can use that to our advantage. While Turing speaks of computer machines and functions of integer numbers, Gödel speaks about formal logic. Taking a tour of Gödel's incompleteness is a good way to observe links between the limitations of computing and those of formal theorization.

We could argue that psychological or cognitive theories are not necessarily formal in the way mathematical theories are. Therefore, the limitations of formal theorization may seem not apply to CTM. However, when the mind is modeled as a computer system, then the kinds of relevant questions that we can effectively answer about it become limited, as we saw in chapter 4. At risk of being redundant, we need to be clear about the model/object distinction that is taking place here. Under the umbrella of CTM, a theory may not be equivalent to a computer system, but the object under study (the mind) is assumed to be. Therefore, even if a theory may transcend the limitations of formal theorization, the object of study will not. Limitations originate at an ontological level from a computational nature imposed to the mind. This would not be as big of an issue if it were not for the fact that a cognitive scientist also has a mind. As long as we assume the mind is a computer, we are faced with a hard choice. Either the mind of the researcher is subject to the same formal limitations as the computationally modeled mind under study, or the former belongs to a fundamentally different (strictly more powerful) ontology than the latter, which would make the theory fundamentally incomplete.

As we face this problem, it is relevant to ask what we are losing when we model the mind as a computer. Is it something we would care about when we approach research questions in cognitive science? Or is it just a theoretical curiosity with no important consequences? We need to dig deeper. And before we do, it will be beneficial to discuss what an *axiomatic system* is.

5.1. Axiomatic systems and formal languages

In formal logic, an axiomatic system consists of a set of initial statements, which we call axioms, and a set of logical inference rules that produce new statements from previously established ones. Both statements and rules are written as symbol sequences. In order to do that, we need to agree on an *alphabet* (i.e., a finite symbol set), and a *syntax* that determines whether an arbitrary symbol sequence is well formed. Syntax must be computable, in the sense that there must be mechanical process which allows us to tell the correctness of statement without ambiguity. In other words, we are talking about a formal language. *Semantics* must also be computable: the process that determines the result of applying a rule must not be ambiguous. Although it may help, *understanding* the rules of of an axiomatic system is never a necessity. A TM should be able to apply the rules in order to generate new statements. This dispensability of understanding is not unlike the intuition captured by the *Chinese room* metaphor (Searle, 1980).

5.1.1. Theories and proofs

A formal theory is the set of all statements that belong in an axiomatic system. This includes the axioms we start with, and all the statements that can be generated by applying inference rules. Usually there is a finite amount of axioms, and an infinite amount of generated statements. In this context, a *proof* for a statement p is a sequence of statements that begins with axioms, and continues as a series of statements, each of which can be formally derived from statements that appeared before. Eventually, the proof reaches statement p, which can only occur if p can be formally derived from axioms.

As a first example, without too much formalization, we shall use commonly known arithmetic principles too prove that

$$(a+b)^2 = a^2 + 2ab + b^2$$

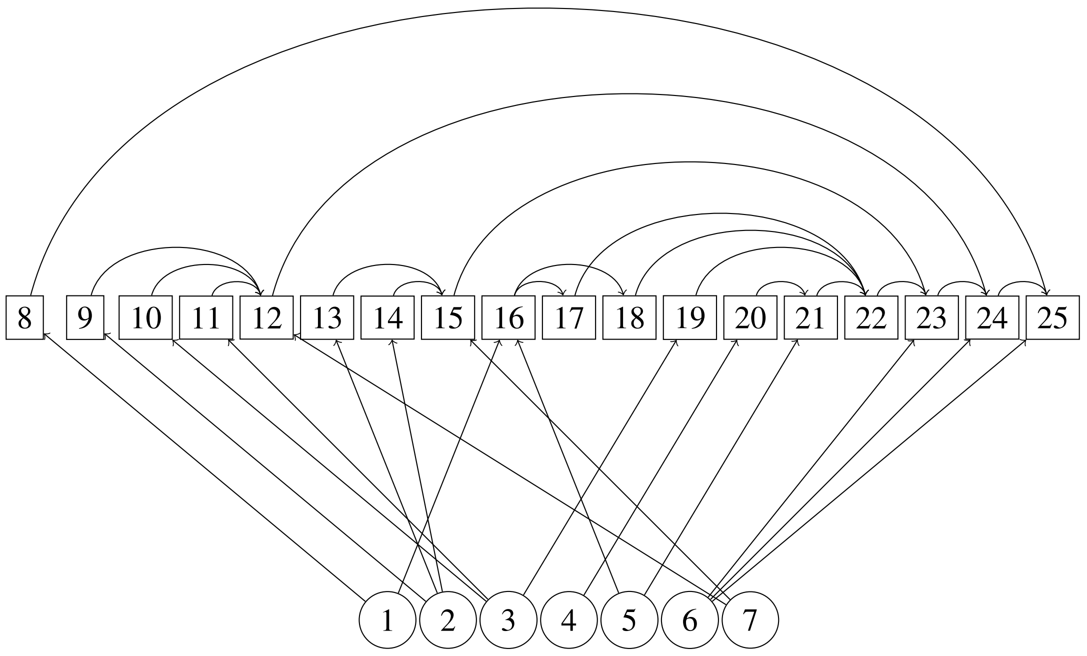
The proof is in table 5.1. Note that the first statements are axioms. Each statement is completely supported by statements that appear strictly before; except for axioms, which

TABLE 5.1. Proof for the statement $(a+b)^2 = a^2 + 2ab + b^2$.

#	statement	support
1	$x^2 = xx$	axiom: definition of squaring
2	$z(x+y) = zx + zy$	axiom: distributivity of $\times$ over $+$
3	$xy = yx$	axiom: commutativity of $\times$
4	$nx = x + x + ... + x$ (repeated n times)	axiom: definition of $\times$
5	$x = y \to y = x$	axiom: symmetry of equality
6	if $x = y$ and $y = z$ then $x = z$	axiom: transitivity of equality
7	if $x = y$ then y can be substituted for x	axiom: substitution of equals
8	$(a+b)^2 = (a+b)(a+b)$	statement #1
9	$(a+b)(a+b) = (a+b)a + (a+b)b$	statement #2
10	$(a+b)a = a(a+b)$	statement #3
11	$(a+b)b = b(a+b)$	statement #3
12	$(a+b)(a+b) = a(a+b) + b(a+b)$	statements #7, #9, #10, and #11
13	$a(a+b) = aa + ab$	statement #2
14	$b(a+b) = ba + bb$	statement #2
15	$a(a+b) + b(a+b) = aa + ab + ba + bb$	statements #7, #13, and #14
16	$xx = x^2$	statements #1, and #5
17	$aa = a^2$	statement #16
18	$bb = b^2$	statement #16
19	$ba = ab$	statement #3
20	$2ab = ab + ab$	statement #4
21	$ab + ab = 2ab$	statements #5, and #20
22	$aa + ab + ba + bb = a^2 + 2ab + b^2$	statements #17, #18, #19 and #21
23	$a(a+b) + b(a+b) = a^2 + 2ab + b^2$	statements #6, #15, and #22
24	$(a+b)(a+b) = a^2 + 2ab + b^2$	statements #6, #12, and #23
25	$(a+b)^2 = a^2 + 2ab + b^2$	statements #6, #8, and #24

are accepted without proof. Also note that the last statement is the *theorem*, which is the statement that we wanted to prove. The fact that it appears at last indicates that it was successfully proved, meaning that the theorem can be effectively generated by arithmetic theory. Figure 5.1 graphically displays how the axioms support the theorem. A proof in a theory corresponds to a formula in which all statements of the proof appear joined by conjunction (logical *and*). The proof in table 5.1 corresponds to the big formula that results from orderly writing its 25 statements, one after the other, joined by the logical *and* operator.

FIGURE 5.1. Diagram for derivations of statements in table 5.1. An arrow from A to B should be read “A supports B”. Circles represent axioms, and rectangle represent other statements. Note that each rectangle statement is supported by statements on the left and/or axioms.



$$[x^2 = xx] \text{ AND } [z(x+y) = zx + zy] \text{ AND } [xy = yx]\text{AND...}$$

Not only the last statement (the theorem) has been proved true. Each intermediate statement has been also proved true, because each one is either an axiom of arithmetic, or is justified by previously proved statements. This big formula is the conjunction of true statements, and therefore is true as a whole, which is relevant for Gödel’s argument: in an axiomatic system a proof can be written as a single big true formula of the theory.

5.1.2. The natural number system

Dedekind–Peano system (Peano, 1889) is an important formalization of natural numbers. It has the following axioms, in which x, y, and z are natural numbers.

(i) 0 is a natural number.

(ii) Equality is reflexive: $x = x$

(iii) Equality is symmetric: if $x = y$, then $y = x$.

(iv) Equality is transitive: if $x = y$ and $y = z$, then $x = z$.

(v) Natural numbers are closed under equality: if $x = \chi$, then χ is a natural number.

(vi) $S(x)$ (the successor of x) is also a natural number.

(vii) S is an injection: $x = y$ if and only if $S(x) = S(y)$.

(viii) No natural number has 0 as its successor: $S(x) = 0$ is always false.

(ix) Axiom of induction: if the two following conditions hold

- K contains 0,
- n being in K implies $S(n)$ is also in K

then K contains every natural number.

Note that some axioms are actually inference rules, in the sense that they can be used to deduce a statement from others. For instance, rule 4 allows us to produce the statement $x = z$ each time we have both $x = y$ and $y = z$ already in our theory. With the Peano system we can easily prove that 3 is a natural number, which in its formal language would be written $S(S(S(0)))$. First we apply rule number 1, and then we repeatedly apply rule 6 as many times as needed.

- Step 1. Axiom 1. 0 is a natural number.
- Step 2. Axiom 6. S(0) is a natural number.
- Step 3. Axiom 6. S(S(0)) is a natural number.
- Step 4. Axiom 6. S(S(S(0))) is a natural number.

Other proofs can be more involved. For instance, what is the result of summing all natural numbers up to n? We will first do a few instances.

- $n = 0$: 0 = 0
- $n = 1$: 0 + 1 = 1
- $n = 2$: 0 + 1 + 2 = 3
- $n = 3$: 0 + 1 + 2 + 3 = 6

- $n = 4$: 0 + 1 + 2 + 3 + 4 = 10

After a few trials, we could observe that the following formula seems to work well:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

It does work for n up to 4. But does it work for *all* natural numbers? The axiom of induction (axiom 9) allows us to prove that indeed it does. We will, of course, need to assume the notion of addition, which can be included as additional axioms[1]. Let K be the set of natural numbers for which the formula works. We first test that $n = 0$ is in K.

$$\sum_{i=0}^{0} i = \frac{0(0+1)}{2} = 0$$

The first condition of axiom 9 holds: 0 is one of the numbers for which the formula works. Now, we shall see if the second condition holds. If n is in K, does that imply that $S(n)$ is also in K? In other words, if the formula works for χ, does that imply that it works for $\chi + 1$? As the question indicates, we shall assume that the formula works for χ, and see if that assumption leads us to the conclusion that it works also for $\chi + 1$. Our assumption is

$$\sum_{i=0}^{\chi} i = \frac{\chi(\chi+1)}{2} \tag{5.1}$$

This assumption may have the same form as the formula we want to prove. But meanings are different. We want the formula to work for all natural numbers. But we are only assuming here that it works for a single number χ. We use letter χ instead of n to remind us of that. We know that the sum up to $\chi + 1$ is equal to the sum up to χ plus the next number, which is $\chi + 1$.

$$\begin{aligned}\sum_{i=0}^{\chi+1} i = & \left(\sum_{i=0}^{\chi} i\right) + (\chi+1) \\ = & \frac{\chi(\chi+1)}{2} + (\chi+1)\end{aligned}$$

[1]The reader can verify that the two following axioms are sufficient: (1) $a+0 = a$, and (2) $a+S(b) = S(a+b)$. For simplicity, the next few paragraphs will rest on intuitive knowledge the reader already has about addition.

This conclusion holds, because we are assuming equation 5.1. We will work with this expression, using what we know about how addition formally works.

$$\begin{aligned}\frac{\chi(\chi+1)}{2}+(\chi+1) &= \frac{\chi(\chi+1)+2(\chi+1)}{2}\\ &= \frac{(\chi+1)(\chi+2)}{2}\\ &= \frac{[\chi+1]([\chi+1]+1)}{2}\end{aligned}$$

In conclusion

$$\sum_{i=0}^{[\chi+1]} i = \frac{[\chi+1]([\chi+1]+1)}{2}$$

As we have just shown, the assumption that the formula works for χ causes it to also work for $\chi+1$. In other words, the second condition of axiom 9 holds. Now we have checked that its two conditions are true. Therefore, we are allowed to also hold the following true: K includes all natural numbers. In this case, this means, all natural numbers belong to the set of numbers for which the formula works.

5.2. Hilbert's program

Although interest on axiomatization has and old history in mathematics (Ball, 2010), axiomatic systems gained special relevance on 19th century. The Dedekind–Peano system we just presented is an emblematic example. The success of axiomatic systems was followed by a crisis as early 20th century mathematicians asked themselves whether *all* mathematics could be formulated rigorously (Kleene, 1952). Metamathematics became a first class concern.

Hilbert's program was a response to the crisis, and a sign of how highly valued axiomatization was. It demanded to base mathematics on a *formal*, *complete*, and *consistent* axiomatic system. In other words, it was expected from mathematics that its statements could be expressed within a formal system with precise manipulation rules for symbol sequences, that all true statements could be proved within the system, and that no contradiction could be generated (as in Russell's paradox, discussed in section 6.3). An important requirement was that of *decidability*: there shall be an effective method (i.e., a TM) to assess the truth value of any statement in mathematics.

It is commonly believed that Gödel's incompleteness theorems have rendered Hilbert's program impossible. Nevertheless, this is still controversial. It is unanimous that the findings of Gödel and his contemporaries have a deep meaning, but it is not clear which would that be (Dawson, 2006; Pekonen, Franzèn, & Mar, 2007).

5.3. Gödel's incompleteness

Gödel's first incompleteness theorem (Gödel, 1931) includes several technical details. Despite that, its proof is based on a relatively simple idea. It is about a statement that we will call G, and that expresses the following.

$$G : \text{“Statement } G \text{ cannot be proved within the formal system } F\text{”.}$$

It is important to note that symbol G appears within statement G. In other words, the statement refers to itself.

We will first assume that G is false in F. This means that its negation is true, which would read as “statement G has a proof within F”. If we take that literally, it means G is true within F (because it has a proof). But this contradicts our initial statement that G is false. The consequence is that F is an inconsistent theory, because it includes both G and its negation. According to the explosion principle, this means that any statement that can be written in the theory can be proved true and false at the same time, which renders the whole of F useless.

Now, if we assume instead that G is true in F, this literally means that G cannot be proved within F. Therefore, F is an *incomplete* theory; because there is a statement G which is true in F, but cannot be proved within F. From these two attempts we see that the theory must be either incomplete or inconsistent, it cannot be complete and consistent at the same time.

The previous argument only works if the language of F is expressive enough to write such a proposition as G. It would be unfair to accuse a theory of being incomplete because it cannot prove a statement it will not state in the first place. Incompleteness only occurs when a theory provides "enough language" to write a statement, but not the capability of assessing its truth value. Gödel's proof shows that if a theory is capable of expressing basic arithmetic facts (i.e., by including the Dedekind-Peano axioms), then it is capable of formally encoding G. The first incompleteness theorem can then be stated as follows.

> "Any effectively generated theory capable of expressing elementary arithmetic cannot be complete and consistent at the same time." (Kleene, 1967)

In order to show that a theory that includes arithmetic can encode G, Gödel assigns a natural number to each syntactically correct statement that can be written within the theory. Intuitively, this can be accomplished by orderly generating each possible symbol sequence (remember that the system has a finite alphabet) and checking each time whether the formula is syntactically correct. The first correct formula will be assigned number 0. And each time a syntactically correct formula appears, the successor of the last assigned number will be assigned to it. Although this numbering strategy is not exactly the same that Gödel used, what is relevant is that each correct logical formula can be assigned a unique natural number. A proof is also a formula. We can construct the formula of a proof by orderly taking the formulas for all its steps, and joining them by logical conjunction (logical *and* operation). Therefore a proof also has a natural number. By means of Gödel's coding, the statement "G has a proof in F" becomes an arithmetic relation involving the Gödel number of that statement. The relation states that a Gödel number for the proof of G exists. By negating the previous statement, and finding its Gödel number, we get G expressed within natural number arithmetic.

6. CONSEQUENCES OF GÖDEL'S INCOMPLETENESS

6.1. Consequences for axiomatic systems

6.1.1. There is no proof of arithmetic consistency in arithmetic

After Gödel's proof of the first incompleteness theorem, it was soon discovered that it had the following unsettling corollary (Formica, 2011; Gödel, 1931): an axiomatic system powerful enough to include arithmetic cannot prove its own consistency. This is known as Gödel's second inconsistency theorem. The sketch of the proof rests on the fact that, just like arithmetic allows as to build a formula for G, it also allows us to build one for

$$\text{Cons}(F): \text{ “System } F \text{ is consistent”.}$$

The same statement resembles G more closely if we express it this way

$$\text{Cons}(F): \text{“Statement } 0 = 1 \text{ cannot be proved within } F\text{”.}$$

An inconsistent system can prove any formula true, *ex falso quodlibet*, and in particular it can prove that $0 = 1$. Now, if F could prove $Cons(F)$, it would mean that contradictions cannot be derived from F. Therefore the negation of G cannot be derived, because it leads to a contradiction, as we previously saw. This sounds a lot like a proof of G. After a few technicalities, that is the case: if F can prove $Cons(F)$, it means that F can prove G. But we know that from the first incompleteness theorem that this is false. By *reductio ad absurdum*, we must conclude that F cannot prove $Cons(F)$.

6.1.2. There is no definition of arithmetic truth in arithmetic

Tarski's undefinability theorem (Tarski, 1936; Murawski, 1998) uses methods in Gödel's first theorem to prove that a theory that includes arithmetic cannot include a definition of truth. As consequence, a definition of truth sits at a metatheoretical level with respect to such a theory.

In a formal theory, a definition of truth would be a formal relationship that only holds true when applied to a true formula. This is analogue to a machine that can orderly be fed the symbols of a formula as inputs and that outputs whether the formula is true after mechanical processing.

$$\text{True}(g(A)) \leftrightarrow A$$

Here, A is a formula, and $g(A)$ is the Gödel number that identifies it[1]. Let T be the set of all natural numbers n for which $True(n)$ holds. This means

$$\text{True}(n) \leftrightarrow n \in T$$

Once again, details require careful mathematical treatment, but a sketch of the proof is beautifully intuitive. If $True(x)$ was a true statement in system F, then the inverse statement could be proved in F by simply adding the negation symbol $\neg$ in a way that does not change the truth value.

$$\neg\text{True}(n) \leftrightarrow n \notin T$$

In other words, a formula like $True(n)$ would not only be a device for testing if a formula is true in F, but would also implicitly allow us to build a complementary device for testing if a formula is false in F. As a result, any formula could be effectively tested either true or false, which would make the system complete. The first incompleteness theorem proved that this is not the case. In other words, $True(n)$ cannot exist as a definition in F, because that definition would *always* tell if a given formula is true, but we have just seen (in section 5.3) that we cannot always tell.

We should note that Tarski's proof does not only apply to theories that include arithmetic. Actually, it applies to any theory that includes negation and has enough self-referential capabilities to write formulas that refer to other formulas in the system. Developments on the limitations of logic by Tarski (1936), Church (1936), Turing (1936, 1937), Post (1936, 1944), and others, stand on the shoulders of Gödel's (1931) work. A

[1]Remember that any sequence of symbols can be assigned a natural number (see sections 3.3, and 5.3).

fundamental step was Gödel's use of the *diagonal method*, which was later made more explicit by Carnap (1934; 2009) and Kleene (1938, 1952). All those developments, owe a lot to Cantor (Cantor, 1891), and the diagonalization strategy he used to prove that there are more real numbers than natural numbers[2].

6.2. Relationship with Turing halting

Turing halting involves a machine that tries to answer a question about another machine; and the critical issue in the proof involves a program that self-referentially asks about its own halting. A machine for Turing halting has the flavor of a proof for the self-referential Gödel's statement. The relationship between the results of Gödel and Turing is so intimate that Kleene (1952) and Penrose (1994) managed to, anachronically, state the first incompleteness theorem in terms of Turing halting. Penrose's version is as follows.

> "Suppose A is a Turing machine which is such that whenever A halts on an input (q, n) then $Cq(n)$ does not halt. Then for some k, $Ck(k)$ does not halt, yet A does not halt on (k, k). In other words, if the halting of A is a sufficient condition for the non-halting of Turing machines then it is not a necessary condition for that; still more briefly: soundness of A implies incompleteness of A."

A formal system has the property of being sound if it only proves true statements[3]. Penrose's wording of the incompleteness theorem highlights it's relationship to Turing halting through soundness. Since the halting of a TM can be written as a statement in arithmetic, a sound theory must not always be able to deduct whether a given machine–input pair halts. If it always did, that would contradict Turing's undecidability theorem. As we saw in chapter 4 this would produce contradictory (i.e., false) statements, which would in turn contradict the soundness of the theory. Therefore, in order to *correctly* capture the fact that sometimes a TM's halting cannot be decided, a sound theory must be incomplete.

[2]Although he had proven the same before using a different argument (Cantor, 1874)

[3]More precisely, a formal system is sound if it only proves formulas that are true under any possible assignment of meaning to its symbols.

Turing himself commented on Gödel's work in his 1936 paper, giving perhaps the most accurate and straightforward interpretation of Gödel's incompleteness theorems.

> "Gödel has shown that (in the formalism of Principia Mathematica), there are propositions $\mathfrak{A}$ such that neither $\mathfrak{A}$ nor its negation[4] is provable. On the other hand, I shall show that there is no general method which tells whether a formula $\mathfrak{A}$ is provable in the Principia Mathematica[5]."

Let us imagine for a moment that Gödel was wrong, meaning that every logic statement could be formally proved either true of false, no statements being left out of having a systematically assigned truth value. There would be no need to ask which statements can be given a truth value, because answer would always be a trivial yes. As Turing points out, a machine could be built to consecutively prove all provable formulae, eventually reaching either $\mathfrak{A}$ or its negation. In the first case we would know $\mathfrak{A}$ is provable. In the second we would know it is not. End of story. But after Gödel proved that some statements simply could not be effectively given a truth value, things got more complicated. Now we know there is no guarantee that the machine will ever stop, because a real possibility exists that neither $\mathfrak{A}$ or its negation will ever be deduced.

It would have been sensible to believe that any sound formal question could be answered in the future, even if it is a very distant future, if enough effort is put to it. Of course we can accept that there are also paradoxes which are not worth the effort. But we could have thought that those are special cases that can be easily identified and disregarded. Unfortunately there are very reasonable formal questions that simply do not have an answer, not one that can be effectively proved correct at least. Those are more than the ones that can be answered. And the general situation will be that when a formal question has not been answered, it will not be possible to tell whether it is because more effort is needed or because an answer does not exist.

The fact that Turing's approach involves a *machine* gives Gödel's work a little bit of materiality. The issue of Turing halting undecidability corresponds to the following

[4] Actual wording in Turing's paper is "such that neither $\mathfrak{A}$ nor $-\mathfrak{A}$ is provable".

[5] In the original text, Turing uses letter K instead of the words "the Principia Mathematica"

material world issue. Physically realizable machines have the means to process symbolic representations of questions about machines (including themselves), and in many cases, are bound to fail in finding an answer.

6.3. Implications for logic language

Late 19th century efforts for giving mathematics the rigor of arithmetic soon found difficulties in the form of paradoxes in set theory. Cantor's work on infinite cardinalities lead to the construction of a set of cardinal numbers. It included $\aleph_0$, the cardinality of natural numbers, $\aleph_1$, the cardinality of real numbers, and went on an on with as many members as there are natural numbers. Cantor realized that if a set existed that could include all cardinal numbers, then it would be missing a cardinal number, which is a contradiction. The fact that this well defined set could not exist was a red alert for set theorists.

Later the famous Russell's paradox was discovered (Rang & Thomas, 1981). It focuses on the following set.

$$R = \{x \mid x \notin x\}$$

R is the set of all x such that x is not a member of x. Although it seems like a weird definition, and already smells like contradiction, the issue is that the criterion for belonging to this set is a syntactically correct logical expression, and there is no a priori reason that prevents it from having a truth value for a given set x. We could even suspect that the value is always false, since there does not seem to be a way to make x a member of itself. This would be no problem, because it would only mean that no object satisfies the criterion, and therefore R would be just the empty set. The problem appears when we ask if R is a member of R. Once again a strange question, but syntactically valid. If the answer is yes, then $R \in R$, which means R does not satisfy the criterion for belonging to R, which in turn means $R \notin R$. We reached a contradiction. No surprises here; $R \in R$ was a strange proposition anyways. So what if the answer is no. That means that $R \notin R$, which means that R does satisfy the criterion for belonging to R, and therefore $R \in R$. Another contradiction. In consequence,

$$R \in R \iff R \notin R$$

This result concerned Frege (1893), who expressed the following reaction.

> "Is it always permissible to speak of the extension of a concept, of a class? And if not, how do we recognize the exceptional cases?"

This questions are not too different than the ones that later would be answered negatively by Gödel and Turing respectively. Frege concerns were well placed because an undealt inconsistency in set theory was an inconsistency in what was being proposed and built as a foundation for the hole of mathematics. This would trivialize the field by making every statement true and false at the same time. Fortunately, there are ways to deal with Russel's paradox, the simplest of which is to impose the requirement that the definition of a set can only refer to sets that already exist.

Gödel's statement bears a resemblance to Russell's paradox.

G : "Statement G cannot be proved within the formal system F".

This is not a coincidence, since both are applications of Cantor's diagonal argument. However, at the time Russell's paradox was discovered, Gödel's statement would have not been considered a problem, because it would have been considered non-sensical that the symbol for a proposition was contained in the proposition itself. And also, it was expected that solving Russell's paradox would rule out this kind of expressions. Wittgenstein (1921) takes note of this understanding of early 20th century logicians as follows.

> "No proposition can say anything about itself, because the propositional sign cannot be contained in itself (that is the 'whole theory of types')."

But the whole point of Gödel's proof is to show that arithmetic has the power to encode a reference to a statement in the same statement. Just like had happened before with Russell's paradox, there was a distance between intuitive meaning and the possibilities of formalization. We know the rest of the story: G can be formalized in arithmetic, and we are reviewing the consequences of that.

We can learn two things from this story. The first is that Cantor's diagonal arguments is the source of all kinds of beautiful paradoxes. The second is that the meaning and consequences of these issues strongly challenge intuition, in such a way that even early 20th century mathematicians struggled to understand them. This explains why, as we will see next it is easy to underestimate or misinterpret its consequences for psychology.

7. PSYCHOLOGICAL INTERPRETATIONS OF INCOMPLETENESS

7.1. Piaget's reaction to Gödel's theorems

In *Structuralism*, one of his late works, Piaget (Piaget, 1970) presented an organized perspective of his approach to intelligence, and made a brief mention to Gödel's incompleteness. The relevance of this is that Piaget had already published works with deep roots in algebraic structures when Gödel's proof was published. Taking in consideraton that incompleteness touched the core of his theoretical framework, he reacted in a rather optimistic way. In his view, the fact that formal knowledge structures are incomplete makes them constantly open. Since a theory cannot complete itself, it will eventually need to resort to a stronger theory that completes the previous one, but has an incompleteness of its own. In this way, incompleteness would constantly drive development in consistency with Piaget's constructivist view. As we know from the second incompleteness theorem (section 6.1.1), there is no proof of consistency *inside* arithmetic. Piaget (Piaget, 1970) had learned that Gentzen managed to prove the consistency of arithmetic from *outside* by a "stronger" formalism. This served as a base for proposing that incomplete formal structures call for *stronger* ones in order to overcome their incompleteness.

Piaget's reaction to Gödel suffers from a few problems. In some important sense *primitive recursive arithmetic* (PRA), the "stronger" system that Gentzen used for the proof, is not stronger, nor weaker, but actually comparable (Weyl, 1921). We should also note that the proof is subject to the consistency of PRA. If PRA was not consistent, it would prove anything it can express, including the consistency of arithmetic and its inconsistency at the same time, which proves nothing. In this sense, instead of proving the consistency of arithmetic, Gentzen's proof links it to the consistency of PRA. Additionally, there is the question of what drives the development of simpler structures that are not powerful enough to be incomplete. Finally, when a formal structure reaches the limit of a UTM's power, how can it further develop? We can shed some light on these issues by realizing that a formal structure that is maxed out in power to a UTM level, can still evolve to better adaption, because what determines adaption is not a position in a hierarchy of theoretical strength, but the adequacy of the actual definition of the structure.

7.2. Did Wittgenstein misunderstand Gödel?

Here is a paragraph from Wittgenstein (Wittgenstein, von Wright, Rhees, Anscombe, & Anscombe, 1978) on Gödel's incompleteness.

> I imagine someone asking my advice; he says:"I have constructed a proposition (I will use 'P' to designate it) in Russell's symbolism, and by means of certain definitions and transformations it can be so interpreted that it says: 'P is not provable in Russell's system'. Must I not say that this proposition on the one hand is true, and on the other hand unprovable? For suppose it were false; then it is true that it is provable. And that surely cannot be! And if it is proved true, then it is proved also true that it is not provable. Thus it can only be true, but unprovable." Just as we can ask, "*Provable* in what system?", so we must also ask, "*True* in what system?" "True in Russell's system" means, as was said, proved in Russell's system, and "false" in Russell's system means the opposite has been proved in Russell's system.—Now, what does your "suppose it is false" mean? In the Russell sense it means, "suppose the opposite is been proved in Russell's system"; if that is your assumption you will now presumably give up the interpretation that it is unprovable. And by "this interpretation" I understand the translation into this English sentence.—If you assume that the proposition is provable in Russell's system, that means it is true in the Russell sense, and the interpretation "P is not provable" again has to be given up. If you assume that the proposition is true in the Russell sense, the same thing follows. Further: if the proposition is supposed to be false in some other than the Russell sense, then it does not contradict this for it to be proved in Russell's system.

This paragraph has been perceived as a rejection of Gödel's mathematical proof (Hintikka, 2000), starting a debate that suggests that Wittgenstein misunderstood incompleteness. Gödel himself was not fond of Wittgenstein's comment and reacted with strong criticism (Bays, 2004). Floyd and Putnam (Floyd & Putnam, 2000) made a compelling defense of Wittgenstein, claiming that it is the latter who has been misunderstood. They hold that Wittgenstein made an important philosophical contribution to the understanding Gödel's incompleteness. The source of confusion seems to be the wrong assumption that Wittgenstein rejected the mathematics of Gödel's incompleteness. Instead, the actual target of his criticism was philosophical remarks made by Gödel and others (Floyd & Putnam, 2000; Bays, 2004).

Wittgenstein rejects a common interpretation of Gödel according to which incompleteness means that there are *true statements that cannot be formally proven*. The important part here is that statements are assumed to be *true*. This interpretation originates from a direct reading of Gödel's sentence

$$G : \text{"Statement } G \text{ cannot be proved within the formal system } F\text{".}$$

G speaks about itself. It can be expressed in system F but it causes problems whether it is true or false[1]. Therefore Wittgenstein is right in criticizing an interpretation that assumes that G is true. In which sense would an unprovable G be true? Not in the sense that it has been formally proven, because it has not. Nor in the sense that it satisfies a definition of truth, because it does not exist in F, following Tarski's undefinability. We have to look for a truth value for G beyond the limits of F. And if we find it, that does not guarantee that G is true unless the system F' we use to prove it is consistent itself. And we fall in the same problem again, either F' can prove its own consistency or we have to search for a proof outside F'.

What happens here is that formal criteria for truth in axiomatic systems were expected to have enough reach to either prove or disprove any statement in the system. But in F, not all statements can be reached so to assign them a truth value that agrees with those criteria.

[1]See chapter 5 and section 6.1.2 for details.

Consequently, the wordings of Turing (section 6.2) and Hofstadter (Hofstadter, 1980) are cleaner: consistent axiomatizations of number theory include undecidable propositions.

7.3. Lucas-Penrose interpretation of Gödel

Lucas (Lucas, 1961) has used Gödel's incompleteness to lay down an argument against mechanism in congnitive science.

> [...]We now construct a Gödelian formula in this formal system. This formula cannot be proved-in-the-system. Therefore the machine cannot produce the corresponding formula as being true. But we can see that the Gödelian formula is true: any rational being could follow Gödel's argument, and convince himself that the Gödelian formula, although unprovable-in-the-system, was nonetheless—-in fact, for that very reason—true. Now any mechanical model of the mind must include a mechanism which can enunciate truths of arithmetic, because this is something which minds can do: in fact, it is easy to produce mechanical models which will in many respects produce truths of arithmetic far [259] better than human beings can. But in this one respect they cannot do so well: in that for every machine there is a truth which it cannot produce as being true, but which a mind can.

An equivalent position has been presented by Penrose (Penrose, 1994, 1999). One could easily agree with the conclusion and be satisfied enough to overlook a fundamental flaw in the argument. One thing is to say that Gödel's sentence *could* be true. A different one is to claim that it is undoubtedly so. Then the question is, does Lucas sufficiently support that Gödel's sentence is true? He briefly provides two separate pieces of justification. First, he argues that it is what "any rational being" would think, which has to be disregarded as an *ad populum* fallacy. Second, he says that Gödel's sentence is "unprovable-in-the-system", which is by definition what Gödel's sentence means. This is a piece of circular reasoning that can be uncovered by asking, how does Lucas know that the sentence cannot be proved in the system? Because he said it at the beginning. With this, Lucas introduces an important difference with respect to Gödel's incompleteness. Gödel did not introduce an a priori statement holding that G is not provable in F. This is why Lucas can say something that Gödel could not. If Gödel introduced such a fact initially in order to build his proof, he could be accused of already assuming something that should be proved. And how would he formalize it? In Lucas' language it has the appearance of being a harmless statement. But formally it would involve including G as a true fact, therefore trivially proving G which means "proved-in-the-system". As a result, the system would be inconsistent instead of incomplete.

Gödel did not prove that the sentence is true. Instead he proved that the sentence can be formally written in arithmetic. This leads us to the main issue: why does Lucas believe that Gödel's sentence is obviously true? By the arguments at the end of section 7.2, we must insist that this is a wrong assumption. But this important issue still requires a little more attention

7.4. The "truth" of Gödel's sentence

A possible source of confusion is the belief that, in an axiomatic system, any statement must be either true or false, excluding any other possibility. It is a reasonable belief. But now that we are aware of Turing halting, the second incompleteness theorem and Tarski's undefinability, we should consider a third option: a truth value may not not correspond to a statement. We now can explore three instead of two possibilities.

(i) *Gödel's sentence is false*. As we have previously seen, this derives in the consequence that the theory under analysis is inconsistent.

(ii) *Gödel's sentence is true*. We have reviewed a few arguments against this possibility (sections 7.2, 6.1.2, and 6.2). Note that this case does not invalidate Tarski's undefinability theorem. Therefore, if Gödel's sentence is true, then there is no definition in the system for that truth, in a very Russell paradoxical way. Then, under which definition could we arrive at the conclusion that the sentence is true, if we have not discussed any other system?

(iii) *That Gödel's sentence is neither true nor false*. This alternative seems to produce no contradiction.

The third possibility seems to be the interpretation of Turing, Wittgenstein, and Hofstadter. And there is a good reason for it. TMs will allow us to give the problem some materiality.

First, assume for a moment that arithmetic is consistent and sound. If we programmed two UTMs with the axioms and inference rules of arithmetic, a *positive* one with the task of proving G, and a *negative* one with the task of disproving G, none of them would halt. The positive one could not halt because in that case it would have proven that its methods are not strong enough for it to reach a halting state (incompleteness). The negative one could not halt, because that would prove that the positive one would halt, and we already dismissed that possibility (inconsistency). We still have the possibility of giving up hope for consistency: both machines would halt, and arithmetic would be useless until we successfully find a way to repair inconsistencies.

But let us not give up hope yet. What if arithmetic is consistent but not sound? Then the positive machine would be allowed to halt even though that would theoretically prove that it cannot halt. And the negative machine would be allowed to halt, even if that speaks wrongly about the halting of the positive machine. We have three cases to analyze.

(i) *Both machines halt*. Then arithmetic does not only lack soundness but also consistency, because its axioms allow to mechanically prove G and its negation.

(ii) *Only the positive machine halts*. Then it proves G, and at the same time it wrongly proves the statement that the positive machine cannot halt, which does not matter because we abandoned soundness. But also this provides the arithmetic proof of G that G states not to exist. This contradiction occurs regardless of the fact that the negative machine does not halt, since soundness is no longer required. Once again, arithmetic results to be inconsistent.

(iii) *Only the negative machine halts.* Analogue to previous cases, this proves the negation of G, from which we can prove G. Therefore arithmetic is inconsistent.

Our only alternative before giving up the consistency of arithmetic is to accept the possibility that none of the machines halt. In other words, the axioms of arithmetic do not have the effective power to assign a truth value to G.

Someone could say "that situation is exactly what G means, therefore it is true!". If we were to give a name to this proof method it would have to be *proof by common sense*. Lucas and Penrose are right in this: common sense strongly suggests that G is true. But they are not considering that common sense is not 100% bullet proof. It is precisely *this kind* of common sense about formalizations that often proves to be deceiving. It is what lead Wittgenstein and his contemporaries to hold that statements like Gödel's sentence could never be formalized, and what lead early proponents of a naive set theory to think that it roots in formal logic would protect it from contradictions (see section 6.3). The Lucas-Penrose argument suggests that formal proof and common sense are capable of granting the same type of truth, only that the latter can do so in some cases the former cannot (e.g., Gödel's sentence). But common sense and formal proof are not tied together so strongly. More often than not, formal truth is not good enough for common sense, and common sense truth is not good enough for formalization, simply because they can disagree.

Considering this situation, our next step is to explore what happens when some statements have no truth value in an axiomatic system.

8. KNOWLEDGE IN AN INCOMPLETE SYSTEM

A good theory of any phenomenon must make predictions that are consistent with observations. In that sense, the statements it produces must be *true* to empirical data. In a traditional dichotomous view of truth, the only other possibility is that a statement is false. Just like true statements, false ones can be analytically predicted to be so by a theory, or be determined by observation.

Let's add the results of Gödel and Turing to the discussion, as Turing clearly summarizes them (section 6.2). We will call those findings the Gödel–Turing limit. They point to an insufficiency of dichotomous truth at an epistemic level: it is impossible to know the truth value of some statements regardless of the amount of finite empirical data available or the amount of effort devoted to it. This occurs because theories with enough power to express arithmetic become capable of self-reference, and some self-referential statements become formally problematic both if they are true or if they are false. Not all and not only evidently self-referential statements take part in this issue. Statements about the consistency of a theory, a definition of truth for it, and others we will later see, are also dragged into the problem.

Turing halting provides an example of a physically realizable object that has one of two mutually exclusive properties (it either halts or not), and would definitely behave according to the one it has if extended in space–time. The problem is not that the value of the property does not exist. The problem is that, even though it definitely exists, and even if we have all the information that determines its value, it is not possible to *know* it.

8.1. Truth values and world facts

Dichotomous truth values allow us to abuse language speaking of what *is* true and what we can *know* to be true, as if they were the same. In principle, this should work because a one to one mapping between *true* facts of the world and provably true statements is highly hoped to exist. Observations support that logic structures capture the way facts work in the world. Direct experience suggests that things cannot *be* and *not be* at the same time. Also that patterns like mathematical equality, conjunction, disjunction, implication, etc., are true to what actually happens. History suggests that whenever formal structures fall short of capturing the way world facts work, or create trouble with themselves, a solution can be worked out so that better figures of the world can be built with higher hopes of finding a one to one mapping.

What we are talking about is a fundamental metatheoretical issue. The whole point of theorization is that we do not need to limit ourselves to observable facts. We can model the way facts work, record what we observe, process, and *predict*. As a consequence, we can alter our behavior in order to produce the results that better serve our goals and access a kind of understanding about how things work. A one to one object–theory formal relationship is a stronger hope than just being able to theorize. It holds the promise of the ultimate control over the world. Even if recording enough facts proves to be difficult, or the processing happens to be way too resource–hungry for practical use, it would be reassuring to keep the hope that it *could* be done.

We want theories to be *correct* in the sense that what they predict ends up being what actually happens. A relaxed statement of this hope is that we want theories to correctly predict what hapens *most of the time*, with high enough success rate for us to base decisions on an assessment of risks. This would allow us to obtain a favorable outcome most times, so that the global balance is positive with good enough certainty: we loose some battles but we win the war. However, we should note that this does not take us away from a dichotomous truth model. For this relaxed strategy to work, we need a means to calculate the true probability of each possible outcome. In other words we need a TM to prove statements of this kind.

“The probability for outcome A is strictly bigger than 95%”.

Fortunately we have an excellent example in probability theory and statistics which are widely used today in scientific research. Take confidence intervals for instance. We grab a data set, and then we use the axiomatic system of classic inferential statistics (or a TM / stats software) to prove a statement like the following.

"Given dataset D, a confidence interval at a γ confidence for variable X, assuming a probability distribution $f(x)$, is x_{lower} to x_{upper}" .

The `women` dataset provided by the R statistical software records height and weight data for 15 US women, taken from the 1975 World Almanac and Book of Facts. The statement becomes:

"Given dataset `women`, a confidence interval at a 95% confidence for variable `height`, assuming a probability distribution $t(x, \text{df} = 14)$, is 62.5 to 67.5 inches."

The theory is not directly saying that 95% of women at that time and place were expected to be between 65.5 to 57.5 inches. That is our use of the theoretical result, which should consider whether data recording methodology is consistent with axioms and facts assumed for calculations. What the theory says is that the statement in quotation marks is true in the axiomatic system of inferential statistics given the provided facts. And not 95% true, nor 5% true. Just true. If we replaced 62.5 and 67.5 with any other numbers, the theory would not prove the resulting statement.

Back to our desire for correct theories, what we expect of formal models of the world is soundness and consistency. In other words, we respectively want theories to only predicate truth about statements that map to true world facts, and to be free of contradictions that can trigger the principle of explosion. Additionally, we want completeness: that we can find a truth to any statement that maps to a valid question. Otherwise, the theory will be useful, but sometimes it will leave us under the same shadows as if there were no theory at all. So, does Gödel's incompleteness sentence us to coexist with shadows?

One possible interpretation of Gödel's incompleteness is that, in a formal system, some statements could be true but unprovable. But then *true* in which sense? Tarski(Tarski, 1936, see section 6.1.2) already showed that a complete definition of truth is not possible in such a system, because the definition applied to the unprovable statement would work as a proof of the unprovable. We are left with defining truth as a property of statements that the system can prove. But proof does not reach the the unprovable statement, so we should not call it true in that sense. For the purposes of our current endeavor, which is to target theorization, we must conclude that within the limits of a theory, statements come in three flavors: true, false, and neither. Consequently, we need to review the dichotomous truth view.

We will distinguish three analysis levels, which will help us better organize how knowledge in an incomplete formal system relates to facts. At an *epistemic* level, we will discuss truth as an issue of formal knowledge, which is what we have done so far. We will also consider an *ontic* level, which is simply about what *is*. Between them, we will put an *ontological* level, that is *about* what is. In this use, the ontological is not completely independent from the epistemic, and actually functions as a sort of bridge between the other two levels.

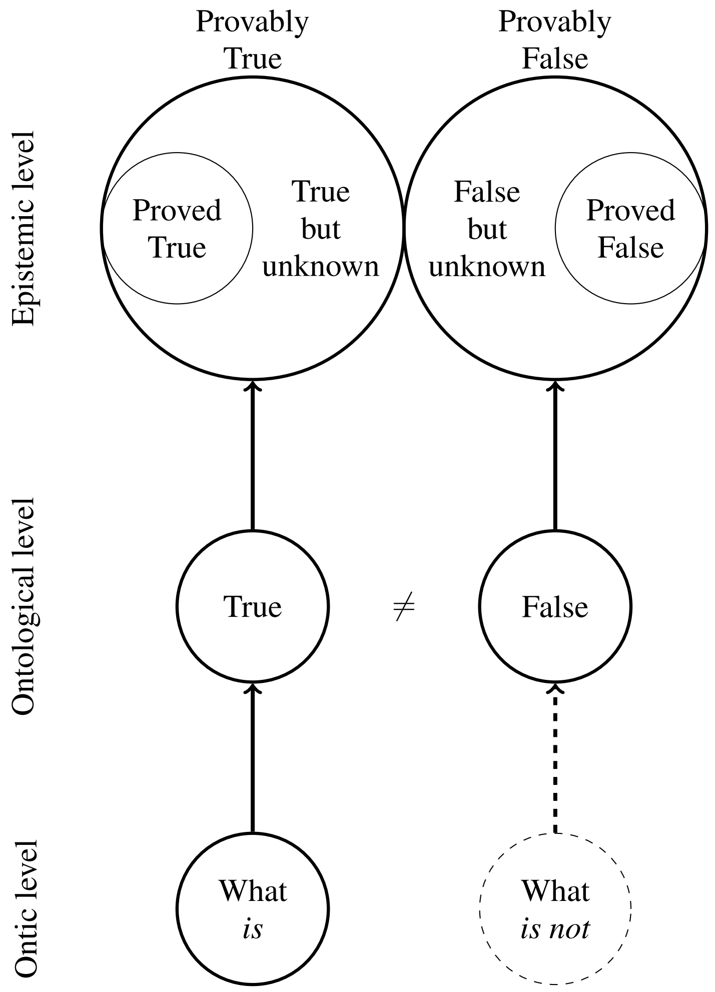
8.2. Dichotomous truth

Figure 8.1 illustrates the most traditional view. At an ontological level, any given statement is either true or false. If our logical model is sound, true statements correspond only to what actually *is* in the world. Since what *is not* does not exist, it cannot be at an ontic level or support false statements. But the relationship is shown in the figure, in order to represent that we believe that *not being* is the cause for a statement to be false. We assign the false truth value to statements that are not true. We assume that our model is complete. Therefore, all true statements are provably true, and false ones are provably false. The epistemic level is slightly more complex than the others, because there is a chance that we do not know the truth value for some statements yet. This dichotomous view of truth values is a good representation of the intuition that, in a sound system, logical truth corresponds to solid world facts; so if something is true, it stands to reason that it should be provable. Note that there is a one to one relationship from each level to each other.

8.3. Trichotomous truth

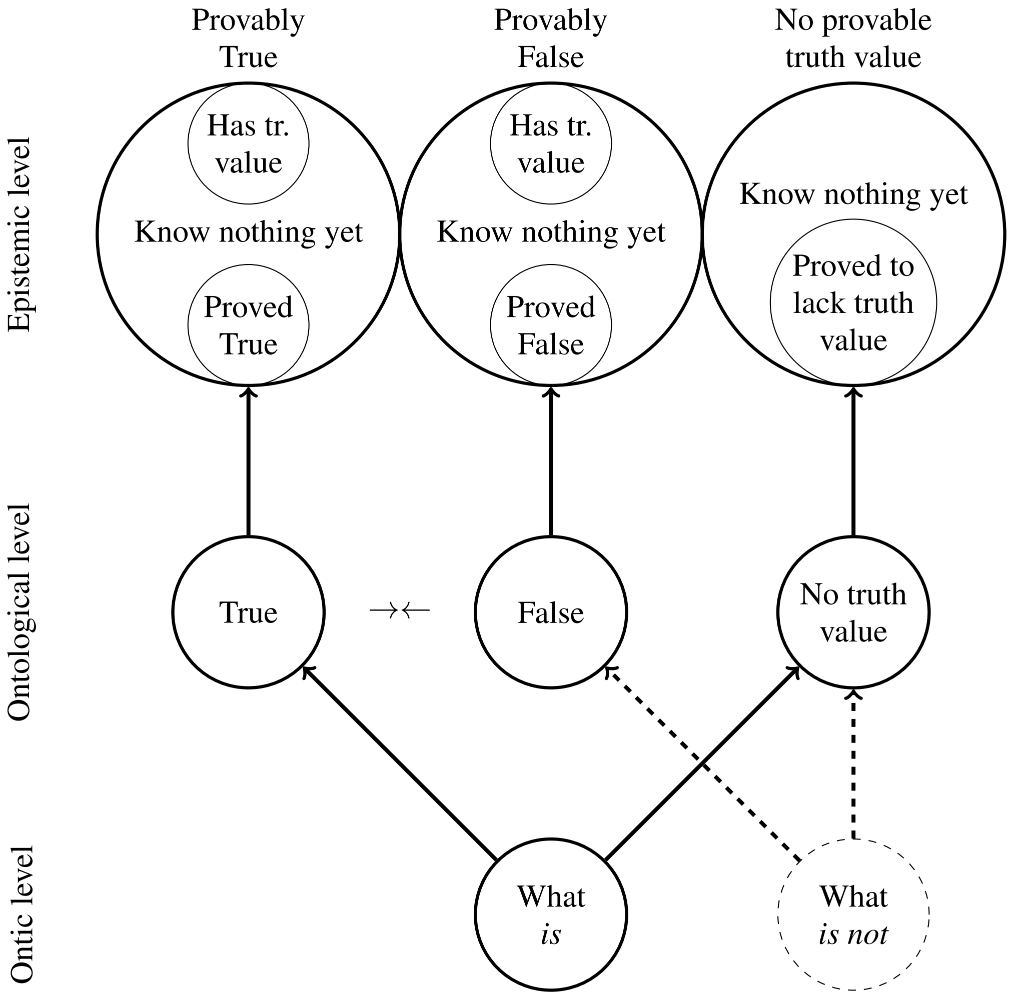
As we previously realized, the Gödel–Turing limit forces us to consider the fact that truth within a theory is no longer dichotomous, at least at an epistemic level. We need to

FIGURE 8.1. Knowledge in a dichotomous truth system. Contrast this with Figure 8.2.



make room for a third truth value which corresponds to no truth value at all. Figure 8.2 shows what occurs within a formal system that is powerful enough to be incomplete. The third truth value opens a Pandora's box of possible fates for the epistemic truth of statements. For instance, a statement can already be proven to have a truth value in the system, although we may not know which one it is yet. This is a very different situation from that of another statement which has already been proven to lack a truth value. In other cases, we will not even know if a truth value can be found. As Turing (1936) points out, the issue is not only that some statements have no truth value, but also that we often cannot know which they are.

FIGURE 8.2. *Epistemically trichotomous view*. Some statements are (1) true, others are (2) false, and others have (3) no truth value.



By separately analyzing the three levels, we get to have a better look at the face of incompleteness. Relationships among levels are no longer one to one. Even though the figure suggests that there is a very close relationship between the ontological an the epistemic level, this time the situation is harder than with complete dichotomous truth. Before, it was guaranteed that a truth value would be eventually found for any statement if we were patient enough. Now, the truth value of a statement we know nothing about could very well be a truth value that cannot be found. And what we are left incapable of knowing could be something that *is* in the world. For our purposes, this is the most important consequence of the Gödel–Turing limit. An incomplete formal system is not only logically incomplete.

It is also bound to be incomplete in capturing what *is*, at least from the point of view of a theoretician.

9. COMPUTERS AND NON-DETERMINISM

After reviewing Gödel's incompleteness (chapter 5) and its epistemological consequences (chapter 6), not much needs to be added in order to suspect that certain facts cannot be known with certainty. However we were dealing with an *epistemic* uncertainty. The Turing halting problem pointed us to an example in which a necessary epistemic uncertainty (halting undecidability) was related to an ontologically certain world fact (actual halting, see section 4.3). In other words, Gödel's incompleteness only proves that knowledge is compromised. The reality that is being modeled is not necessarily affected. But what if the physical world was ontologically uncertain, as the Copenhagen interpretation of quantum mechanics holds (Heisenberg, 1949)?

Whether the physical world is deterministic or not is a difficult question. The sensible approach here seems to be finding out the consequences that an ontologically non-deterministic world would have on modeling the mind as a computer. For that we have a few tools at our disposal, thanks to ingenious mathematical developments. The problem has two parts: how non-determinism affects the physically extended computer machine that implements a mind, and how that computer deals with uncertainty in the world it strives to know.

9.1. Non-deterministic Turing machines and quantum computing

Regular TMs proceed in a deterministic manner. Remember the specification of TMs (section 2.2.3). The state of the full machine is composed of the state of the tape (content and head position) and the state of its control device (a FSM). The next state of the full machine is determined by its current state. Therefore, the input initially on tape fully determines the sequence of states the machine will transit through, and the output. In contrast, a non-deterministic TM (NDTM) can specify more than one possible next state given the current one.

How to run a NDTM? The trick is to think that the machine is not in a single state at a given time, but in a set of states. At the beginning, this set has only one state, which is the initial one. In the next step, the set only includes all the states that could have resulted from the initial one. In general, at each step, the set of states gets updated with all states that could have resulted from the ones present at the previous step. If at some step, some element of the set corresponds to what is considered a final state (which needs to be determined before the machine starts to run), it means that one of the possible execution paths of the machine has reached a result. We can access it by looking at the tape contents inside the final state that was found.

How does the power of a NTDM relate to that of a TM? It is easy to see that NDTMs are at least as powerful as TMs, because a regular deterministic TM is a particular case of NTDM in which the amount of possible next states that can follow the current one is always 1. What is surprising, and a little trickier to prove, is that the same is true in the opposite direction: TMs are at least as powerful as NDTMs, because a TM can be built for simulating a NDTM (Hopcroft et al., 2007, ch. 8). This derives from the following fact. Even though the number of possible states of a NDTM can grow very fast as it runs, it is finite at each step and can be organized in a tree of possible execution paths. A TM can simulate the same thing by orderly generating and going through the execution tree until a final state is found. A three-tape TM can do this by keeping the input string in the first tape, using the second tape to try an execution path, and keeping track of progress in the third tape so that nodes in the tree do not get visited more times than necessary. In turn, as previously mentioned, a several tape TM has the same power as a regular single tape TM (section 2.2.3, Hopcroft et al., 2007, ch. 8).

The conclusion is that a NDTM has not less, not more, but exactly the same power as a regular TM. Something similar (but not exactly equal) occurs with quantum computers which take advantage of the capability that non-deterministic systems have for exploring several execution paths simultaneously. Previously, we briefly mentioned that quantum computers are not more powerful than TMs (section 2.3, Hodges, 2005). They are only faster.

We must however note that NDTMs are not a perfect theoretical correlate of quantum computers. At least, not any more than a regular TM could be. It is true that NDTMs have the same computational power as quantum computers, because TMs have the same power as both (see, Bennett, 1973, for a proof that quantum computers have at least the same power as a TM). But NDTMs can be theoretically *faster*[1] than quantum computers; because harder restrictions operate on the latter (Bernstein & Vazirani, 1997). The point here is that quantum computing takes advantage of quantum physical phenomena that were not considered in the design of classical physics inspired TMs. Nevertheless, just like it happened with previous attempts, adding sophistication to the machine did not produce an advantage in computational power with respect to a regular TM. NDTMs, while not a direct correlate to quantum computers, offer a relatively easy account of how a non-deterministic model of computation happens not to supersede classic TMs. They provide this insight without requiring a dive into quantum computing or quantum mechanics.

So non-determinism applied to a physically extended mind/computer will not affect the limits of theories. We can even stick to a classic single head TM and still have the same predictive power. But how do we deal with the fact that the "outside world" may be non-deterministic too. The Copenhagen interpretation of quantum mechanics (Heisenberg, 1949), draws an important distinction here. It has been known for long that measuring physical quantities is subject to error because there is a limit to the precision an instrument can be built to. Also, because uncontrolled physical variables often affect measurements in a way that, from within the boundaries of a specific theory, can only be described as *random error* or *noise*. Stated this way, this is a methodological issue that affects us at an epistemic level. It can often be mitigated by building more precise instruments and controlling more variables until the precision of predictions is good enough for whatever the purpose is. However, Heisenberg's uncertainty principle (Heisenberg, 1927) points to a different issue: its proposal is that the physical world is ontologically uncertain.

TMs are deterministic. And non-deterministic devices are not more capable than a deterministic TM. It intuitively follows that TMs may be too rigid to model a mind that

[1]In the sense of requiring less steps for reaching a result.

needs to deal with the indeterminacy of the physical world. The answer is this: non-determinism of the world can be dealt with in a computationally deterministic way. And it works very well.

9.2. Bayesian modeling

A fundamental building block in our current theories to model uncertain events is the Bayes theorem. It stands on the following definition from classic probability theory.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{9.1}$$

It reads

> "Probability of event A given that event B is the case, equals to the probability that both events A and B occur, divided by the probability that B occurs in general."

An example may be of help. We know that it is sort of difficult to get two sixes when throwing a dice twice. But if we are in the middle of the exercise and already got one six, we are in a good position. Now we do not need two sixes in order to reach the goal, but just one more. Since outcomes of the two throws are (hopefully) independent, classic probability predicts that the probability of getting two consecutive sixes is obtained by multiplication on the probabilities for the two events. Let A, and B respectively be the events that the first and the second throw of a dice yields an outcome of six.

$$P(\text{two sixes}) = P(A \cap B) = P(A)P(B) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}$$

Sensibly, the probability of obtaining two sixes is 1 in 36. But how does the probability change if we switch to a situation in which we already got the first six? By the definition in equation 9.1, he have

$$\begin{aligned}
P(\text{two sixes}|\text{first throw yielded six}) &= P(A \cap B|A) \\
&= \frac{P(A \cap B \cap A)}{P(A)} \\
&= \frac{P(A \cap B)}{P(A)} \\
&= \frac{(1/36)}{(1/6)} = \frac{1}{6}
\end{aligned}$$

What happened here is that our success in the first throw is past history and no longer uncertain. All we need to do to reach the goal is just to get another six, and that has a probability of 1 in 6.

In our current context, we can learn three things from the previous example. First, we can formally write and process knowledge about uncertain events by performing perfectly deterministic symbol processing. Second, gathering additional knowledge about the state of the world produces an update on the probabilities we assign to events. The third is a clever consequence of equation 9.1. We can rewrite the equation as follows.

$$P(A \cap B) = P(A|B)P(B)$$

And since $P(A \cap B)$ equals $P(B \cap A)$, we have

$$P(A|B)P(B) = P(B|A)P(A)$$

Therefore,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{9.2}$$

Equation 9.2 is known as Bayes' theorem. Formally, it serves the purpose of inverting $P(A|B)$ into $P(B|A)$. Why this is remarkable gets clearer when we think of A as a discrete world state, and of B as a piece of evidence.

$$P(\text{state}|\text{evidence}) = \frac{P(\text{evidence}|\text{state})P(\text{state})}{P(\text{evidence})}$$

It might be difficult to directly assess the probability that the world is in a specific state given the evidence we possess, which is the left side of the equation. But the right side allows us to compute it in terms of probabilities that are easier to obtain by counting empirical frequencies.

- The probability that a piece of evidence emerges from a given world state.

$$P(\text{evidence}|\text{state})$$

- The probability that a given state occurs in the world.

$$P(\text{state})$$

- The probability that a given piece of evidence emerges under general conditions.

$$P(\text{evidence})$$

Bayes' theorem is the starting point to an interesting debate on different interpretations of the notion of probability (Cox, 1946; De Finetti, 2017), which however is not essential for us to go forward. We shall simply say that classic probability is traditionally and mostly associated to the idea of probability as frequency of occurrence. Bayesian analysis opens the door to two families of interpretations; one in which probability quantifies *objective* expectations (Cox, 1961), and other in which probability is a matter of *subjective belief* (De Finetti, 2017). We should also stress that this is just a matter of interpretation. Equations are the same and bayesian analysis is not at all formally incompatible with classic probability, to the extent that it is even derived from classic axioms, as you may have noticed.

9.3. Markovian modeling

In section 8.1 we held that a theory was not only useful for predictions about what always happened, but also for relaxed predictions that spoke about what happens most of the time. Bayesian analysis takes that to a new level, by allowing probabilistic predictions to be continuously updated as more evidence gets available, improving on initial certainty levels. We will specifically focus on Markovian processes (Markov, 1906; Ching, Huang, Ng, & Siu, 2013), which add a time dimension to bayesian inference. They are modeled under the assumption that the current state of a system fully determines the next state, whether this is faithful to the modeled phenomenon or just a convenient simplification.

9.3.1. Bayes' theorem in time: an example

Consider the following example. John has a job that requires him to live and work 24/7 in a basement for a certain period of time. He cannot observe local weather directly, and for some reason forecasts are not available where he is. But he still wants to talk about the weather when he chats with his loved ones through a messaging application. He relies on his pet for that, a turtle whose behavior is somewhat related to weather. John's records indicate the following probabilities for the turtle's behavior.

$$P(\text{sleep}|\text{sunny day}) = 0.1 \tag{9.3}$$

$$P(\text{mild activity}|\text{sunny day}) = 0.3 \tag{9.4}$$

$$P(\text{intense activity}|\text{sunny day}) = 0.6 \tag{9.5}$$

$$P(\text{sleep}|\text{rainy day}) = 0.7 \tag{9.6}$$

$$P(\text{mild activity}|\text{rainy day}) = 0.2 \tag{9.7}$$

$$P(\text{intense activity}|\text{rainy day}) = 0.1 \tag{9.8}$$

These data correspond to $P(\text{evidence}|\text{state})$ at the end of previous section. Note that probabilities for a given weather sum up to 1.0, because any possible behavior of the turtle is always categorized into one of the three shown categories. John has heard that where he is, rainy days are just as frequent as sunny ones.

$$P(\text{sunny day}) = 0.5 \tag{9.9}$$

$$P(\text{rainy day}) = 0.5 \tag{9.10}$$

He also heard that a rainy day tends to be followed by another rainy day 70% of the times and something similar occurs with sunny days.

$$P(\text{sunny tomorrow}|\text{sunny today}) = 0.7 \tag{9.11}$$

$$P(\text{rainy tomorrow}|\text{sunny today}) = 0.3 \tag{9.12}$$

$$\tag{9.13}$$

$$P(\text{sunny tomorrow}|\text{rainy today}) = 0.3 \tag{9.14}$$

$$P(\text{rainy tomorrow}|\text{rainy today}) = 0.7 \tag{9.15}$$

The whole model can be visualized as in figure 9.1. Note how similar it is to a FSM (section 2.2.1), with the sole differences that state transitions indicate the probability for their occurrence next to arrows, and that the output (evidence) is not fully determined by the state (or transition), but is probabilistic.

Introducing evidence

With all this information, John uses equation 9.2 for estimating his *beliefs* on today's weather after observing that his turtle is sleeping.

$$P(\text{sunny today}|\text{sleep}) = \frac{P(\text{sleep}|\text{sunny today})P(\text{sunny today})}{P(\text{sleep})} \tag{9.16}$$

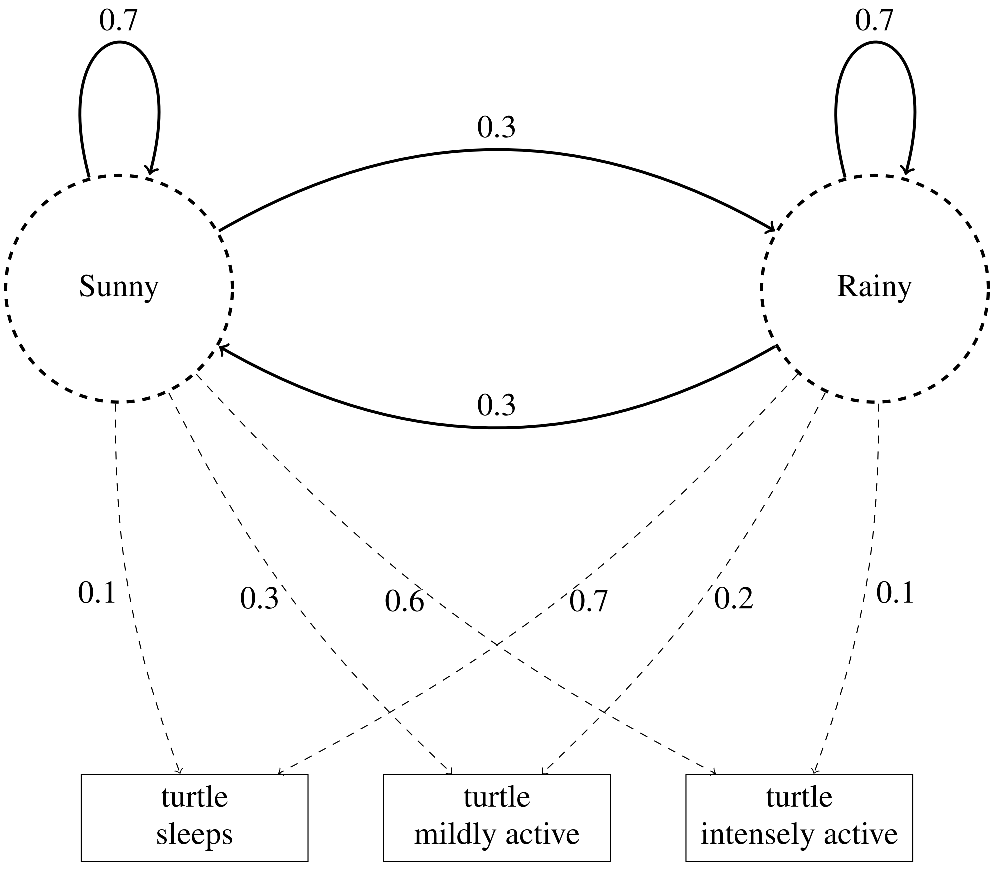
$$P(\text{rainy today}|\text{sleep}) = \frac{P(\text{sleep}|\text{rainy today})P(\text{rainy today})}{P(\text{sleep})} \tag{9.17}$$

We do not directly know the probability that the turtle sleeps. But we have enough information to compute it from equations 9.3, 9.6, 9.9, 9.10. From total probability law[2]

$$\begin{aligned} P(\text{sleep}) =& P(\text{sleep}|\text{sunny day})P(\text{sunny day}) \\ &+ P(\text{sleep}|\text{rainy day})P(\text{rainy day}) \\ =& 0.1 \cdot 0.5 + 0.7 \cdot 0.5 \\ =& 0.4 \end{aligned}$$

[2]Total probability law states that $P(A) = \sum_{i=1}^{n} P(A|B_n)P(B_n)$ as long as B_n are disjoint and their union represents the whole space from which events are sampled.

FIGURE 9.1. John's turtle weather forecast model. Dashed circles are states and cannot be directly observed. Solid rectangles are evidence that can be *emitted* by states. Numbers on solid arrows are state transition probabilities. Numbers on dashed arrows are evidence emission probabilities.



Then equations 9.16 and 9.17 become

$$P(\text{sunny today}|\text{sleep}) = \frac{0.1 \cdot 0.5}{0.4} = 0.125 \tag{9.18}$$

$$P(\text{rainy today}|\text{sleep}) = \frac{0.7 \cdot 0.5}{0.4} = 0.875 \tag{9.19}$$

This is our first result, and corresponds to beliefs for the first day. That is, we have a value for the belief that it is sunny today (0.125), given the evidence that the turtle sleeps. And the same for the belief that it is rainy today (0.875). As expected, the fact that the turtle is sleeping leads to a higher belief that it is a rainy day.

Updating beliefs for the next day

State transition probabilities in equations 9.11 to 9.15 allow us to update beliefs for tomorrow. We need to consider that we do not know the weather for sure, so we have to include both believes for sunny and rainy day.

$$\begin{aligned}
P(\text{sunny tomorrow}) =& P(\text{sunny tomorrow}|\text{sunny today})P(\text{sunny today}) && (9.20)\\
&+ P(\text{sunny tomorrow}|\text{rainy today})P(\text{rainy today}) && (9.21)\\
=& 0.7 \cdot 0.125 + 0.3 \cdot 0.875 && (9.22)\\
=& 0.35 && (9.23)
\end{aligned}$$

$$\begin{aligned}
P(\text{rainy tomorrow}) =& P(\text{rainy tomorrow}|\text{sunny today})P(\text{sunny today}) && (9.24)\\
&+ P(\text{rainy tomorrow}|\text{rainy today})P(\text{rainy today}) && (9.25)\\
=& 0.3 \cdot 0.125 + 0.7 \cdot 0.875 && (9.26)\\
=& 0.65 && (9.27)
\end{aligned}$$

This is our second result, our beliefs for tomorrow are 0.35 sunny and 0.65 rainy. We believe that tomorrow will also be rainy, but with less certainty than today. In order to improve our certainty we need to wait until tomorrow and observe the turtle.

Repeated observations and updates

Note that updates to beliefs are caused by observations of the turtle and by a day passing by. We can repeatedly apply both procedures each day, updating for the turtles behavior, and updating for weather transition probabilities. Assume that the turtles behavior for the next four days is `sleep`, `mild activity`, `intense activity`, `intense activity`. Table 9.1 shows beliefs after each day's observation of the turtle. Note how day three yields very similar beliefs for both weather possibilities, so there is no clear prediction for that day. However, John can be pretty sure of the weather on days 2 and 5, and his estimations are better than those he had on day one after a single observation. This is remarkable, since from John's isolation he never gets to see how the weather actually is.

TABLE 9.1. John's weather beliefs after several days of observing his turtle.

Day	Observation	Sunny day belief	Rainy day belief
1	sleep	0.125	0.875
2	sleep	0.071	0.929
3	mild activity	0.423	0.577
4	intense activity	0.841	0.159
5	intense activity	0.913	0.087

9.3.2. Acting under uncertainty

John's turtle forecast is an example of a hidden Markov model (HMM). It is a useful modeling tool when we are spectators to a process in which states are not directly observable, but can be partially observed by measuring their probabilistic consequences. Variations on this scheme depend on whether we are mere spectators or act upon the world, and if we can observe world states fully or partially as displayed in table 9.2.

TABLE 9.2. Discrete state, discrete time stochastic models. Table extracted from Anthony R. Cassandra's website, http:\\pomdp.org, where he attributes it to Michael Littman.

	Observation only	Observation and action
Completely observable state	**MC** Markov chain	**MDP** Markov decision process
State partially observable through probabilistic evidence	**HMM** Hidden Markov model	**POMDP** Partially observable Markov decision process

When action is included in Markovian modelling, the goal is to map world state to optimal actions in order to reach a goal. The goal is usually to maximize long term rewards, so this requires an assignment of reward values to world states. Partially observable Markov decision processes (POMDPs, Littman, 2009; Astrom, 1965) are specially interesting. They assume that world states are not directly observable, and that the effects of acting in the world are not completely deterministic. In other words, world states are probabilistically related to what can be observed, as in John's weather forecast. And actions are probabilistically related to their actual effects. As an example of this, consider shooting darts to a bullseye target. Under a given circumstance you may choose to take the action of aiming for the center. But after the shoot, there is no guarantee that the state of the world will include the dart within the center region. There is a probability that this happens, but other effects on the world (like fully missing the target) also have a chance.

Since knowledge of the world state is partial in a POMDP, actions are not mapped from *actual* world states (which we cannot know directly); but from a *belief* state, that includes one probability value for each possible world state. This is analogue to John's belief numbers about the two possible weather states. Beliefs on world states are updated both after acting, and after observations, because new partial information can be derived from both kinds of events.

Remarkable emergent properties are associated with POMDPs. An entanglement between action and observation is present, not as a consequence of a design goal, but as a result of seeking optimal rewards with the tools provided by a Bayesian framework. The *programming* of this kind of models is done by setting structure and probabilities, just like at the beginning of John's turtle problem. But this is rarely done by hand picking values. It usually involves fitting parameters automatically to empirical data (Cassandra & Kaelbling, 2016; Atrash & Pineau, 2010). As a consequence, action policies emerge without much intervention of a designer. Even though the global objective of a POMDP is to maximize reward, it could well "delay gratification", and direct a few actions towards reducing uncertainty in its beliefs. In the long term, this may lead to decisions that produce higher rewards. In this context, uncertain belief states are those that assign similar probabilities to all worlds states, in contrast to more certain belief states in which one or a few world states clearly dominate.

Perhaps the most remarkable property that emerges in POMDPs is the necessity of segmenting a continuous belief space into a finite amount of belief *domains*, in order to be able to map beliefs to discrete actions. An explanation is in order. Here we shall not use the term *continuous* as strictly as in the discussion of chapter 3 but in the relaxed sense that is common in data analysis, where the meaning is: dense enough that treating it discretely does not make sense. Such is the case when beliefs are probabilities: a belief is a vector that includes continuous values between 0 and 1 for each possible state. Even if actual world states are considered discrete, belief states are necessarily continuous and often multidimensional. A discretization strategy is needed in order to associate such a space to a few discrete actions. And it stands to reason that, in many cases, similar belief states should be associated to the same action. In a POMDP, such a discretization is automatically learned with the goal of maximizing rewards. We are talking of a self-organizing emergence of discrete beliefs supported by action policy relevance and value. Although this is not even close to explaining why humans organize reality into concepts, it is still noteworthy that a model so overwhelmingly simpler than living beings can exhibit these properties.

9.4. Non-determinism and computational modeling of the mind

The stochastic models we just reviewed do not need to be thought as tools for modeling the mind. Instead, they should be seen as models of non-deterministic phenomena, that *can* be used in a TM as a way of gathering information from and acting on an environment. Whether human knowledge is probabilistic in nature is a separate question (Chater & Oaksford, 2008; Baker, Saxe, & Tenenbaum, 2011). The issue here is that TMs can be programmed to deal simultaneously with multiple kinds of uncertainties in a flexible and successful way, whether uncertainty comes from an ontological lack of determinism or from errors inherent to instrumentation, and regardless of non-deterministic quantum phenomena that may operate on the physical extension of the TM's implementation.

Something to be learned from MDPs, POMDPs, and alternative models of computation like artificial neural networks, is that "Turing computing" does not need be "Turing styled" in the way it is usually perceived: centralized, serial, and rigid. All these models are within the capabilities of traditional TMs. That is why researchers can use their own von Neumann architecture laptops to study and run such kinds of models. First attempts in artificial intelligence tried to analyze problems, divide them into smaller easier tasks, and solve them one by one in an orderly fashion. The main weakness of such a scheme is that success of the whole relies on complete success of the parts. If some step fails, a solution cannot be provided. Stochastic modeling is inspired by a different spirit. We may never be a 100% certain, at any step of the process, and we can still solve the problem well. By doing without the requirement of certainty, information can be better utilized.

But do not forget that this is still a TM, just with a clever program. John never concluded that it was 87.5% true that “the day was rainy”. What he concluded is that he was 100% sure that “his belief was 87.5% for the rainy state”. And he computed that number by deterministically processing symbols under the axiomatic system of classic probability. The morale is: a fully deterministic system can deal with uncertainty quite well. If we search for something computers cannot do in order to build an argument against the computational theory of mind, we need to look somewhere else.

10. SOME CONTROVERSIES AROUND A COMPUTATIONAL MIND

It is no secret that the fundamental claim of CTM is that the mind is a computer. It is stated either directly (McCulloch & Pitts, 1943; Putnam, 1967; Fodor, 1975; Newell & Simon, 1976; Block, 1995, , etc.) or simply assumed (i.e., Chomsky, 1957; Marr, 1983; Fodor, 1983). The mystery lies in what the claim means. Although the debate is as young as digital computers, the field is rich and questions are challenging. There has been enough time to propose a huge amount of philosophical positions, while specifying detailed differences among them. Rather than reviewing several approaches, we will focus on a few controversies that are relevant for this work. Our aim is not to be exhaustive, or to go deep into each issue, but to offer a taste of the diversity of perspectives under CTM and to better understand what it means when it claims that the mind is a computer. We must remember that our goal is not to decide if the mind is a computer, but to explore the consequences of modeling the mind as one that has the limitations we have already exposed.

10.1. Being versus being *like* a computer

We should note that it is possible to use computers, either theoretical models or actual digital devices, as a metaphor for speaking about the mind. How good the metaphor is will be the result of how much it makes it easier to communicate difficult concepts, and how little potential it holds for misleading conclusions. Of course, computers are not exclusive in this. However, lifeless mechanisms hold a special appeal as metaphors of the mind, because we can fully understand how they work, and any life-like capabilities they display must be constructed first. Computers are specially attractive because their mechanism is relatively simple to understand if compared to the enormous complexity of behavior they can achieve.

However, metaphors are not theories, nor fundamental pieces in them. Instead, they play a role in communicating theories and helping us to understand them. It would be problematic to assimilate a metaphor as an essential theoretical device. The vehicle/image *is not* the concept being referred, but only *like* it. This means that some aspects are similar, while other ones are not. In order for this scheme to help intuition without being overcomplicated, we must do without a rigorous or exhaustive delimitation of which aspects are similar. Therefore, if we based theoretical results on metaphors, sometimes we would be making predictions not intended by the theory. How could we tell? The CTM does not claim that the mind is *like* a computer. It claims that it actually *is* a computer. This does not prevent some properties of particular computer mechanisms to be used as metaphors for understanding the theorized mind/computer. For instance, thinking memory processes as storage or retrieval operations on electronic memory devices is common.

Saying that the mind is a computer implies saying that the mind is a physically extended machine. Electronic computers are made from silicon chips, wire, etc.; while humans are made of flesh and bone. CTM does not care, because nothing prevents an implementation of TMs in flesh. However, we should be aware of an implicit identification here between mind and body (e.g., Boring, 1933). If the mind is a computer, and the computer in humans is implemented in flesh, then mind is implemented in flesh. Moreover, it is common to approach this relationship by targeting, not the whole body, but the brain (e.g., Smart, 1959). The mind–body relationship is too big of a topic to review in a comprehensive way here. But we should at least be aware of what kind of claims we are dealing with in CTM.

10.2. Brain and mind

It is immediately problematic to assume that the mind *is* the brain or any physically extended object. A brain has a weight, but how much does the mind weight? Where are mental processes? Can I touch them? The solution in *type identity theory* (e.g., Feigl, 1958; Smart, 1959; D. K. Lewis, 1966; Place, 1970) is that there are types of mental events that are *correlated* to physical events. What does the trick is that, in this view, mental processes are not the brain, but states of the brain. Maybe we cannot attribute weight to a mental process, but neither can we to a *state* of a mechanism.

Functionalism identifies mental states with *functional states* instead of brain states. In this view, what matters the most about a mental state is the role it plays in a system, regardless of the brain states that support it. The notion of mathematical function is involved here in terms of information processing: what matters is the mathematical function, regardless of the choice of TM implementation. Functionalism opposes to type identity theory in the view of an insider (Putnam, 1967), the objection being that the same mental state can have diverse physical realizations. The *multiple realizability thesis* stands on the observation that some mental states like pain, seem to be present in a wide variety of animals with wildly varying physical structures, and in relation to many different physical events. Therefore, a mental state does not seem to be identical to a single brain state. Not every one sees type identity theory as completely incompatible with functionalism. Lewis (1980) believes that both views have problems, but can be mixed into a better identity theory.

10.3. Is there anything beyond the physical?

Work on type identity theory and functionalism presents the mind as something that would not *directly be* a physical object (i.e., the brain) but rather something that *happens to* one (i.e., brain states). It could be argued that this is not a departure from a physical account of the mind, in the sense that what happens to the physical is also physical. *Physicalism* claims that everything is or *supervenes* on the physical. In other words, anything that is not directly physical is a consequence of the physical.

Once again we are in front of a challenging claim. Controversies organize around the meaning of *physical*, the meaning of *everything*, and whether the claim is true (see Davidson, 2001; Moser & Trout, 2002). It does not help that, despite this being a general problem, it has received special attention in philosophy of mind; a field that is rich in challenges to physicalism, but has difficult questions of its own. But, if we are true to the spirit of the question (and to our current purposes), the issue is very understandable. If the mind is a computer, does that mean that there is nothing more than the machine that we can touch and weight on a scale? How does a physical mechanism explain the quality we perceive in feelings that occur in the intimacy of mental life? Is it even *correct* to ask for physical explanations when we talk about feelings? The answer of physicalism must necessarily be that there is nothing beyond the physical. Therefore, if we feel there is something else, there must be an explanation for why we wrongly feel so. Since we arrive to these questions by the powers of reason, it could actually be a logical issue. Perhaps some false inference leads us to thinking that there must be something beyond the physical. That is the opinion of Dennett (1993) in his refutation of Jackson's "Mary's room" argument (Jackson, 1982, 1986). We will revisit this issue in chapter 11.

10.4. Representation of knowledge

Theorization on the mind cannot simply ignore the essential phenomenon of representation. Mental contents usually involve the presence of objects in the world that *present themselves again* in the mind. We want to understand how this is possible and how it works. We could make the stronger claim that the nature of the mind is to perform computations on representations. This would give the theory some clarity about how to implement mental processes. There is more than a single way to approach this problem. Here we will briefly see two opposing views.

10.4.0.1. Representational theory of mind

Fodor (Fodor, 1975) holds that thinking occurs in a *language of thought*, also called *Mentalese*, capable of producing mental representations. We will call this the language of thought hypothesis (LOTH). The idea bears a lot of resemblance to axiomatic systems. Just like complex logic formulas can be produced by combining primitive elements, Mentalese produces complex representations by combining primitive ones. Similarities do not stop there. In both cases, the processing of expressions is carried out strictly by mechanical manipulation of symbols (i.e., by a TM). Advantages of the LOTH can be easily understood in terms of these similarities to axiomatic systems. It explains the fact that human cognition is capable of an infinity of different expressions, and yet it can establish systematic relationships among them without a need for infinite experiences.

Even if we asume that the LOTH is true, there are important controversies involving whether Mentalese is exclusively human (Gallistel & King, 2011), whether it only applies to high-level thought (Fodor, 1983), or if it resembles the propositional character of natural language (Pinker, 2005). Note that the LOTH endorses a computational nature of mental processes. This does not exclude identification with brain states or functional states. Also, it does not require physicalism. However, Mentalese can be implemented in a physically extended TM. Therefore, it provides a solution for physicalists, and is often associated to them.

10.4.0.2. Distributed representations

When compared to the LOTH, connectionism (Rumelhart, McClelland, & PDP Research Group, 1986) offers a radically different proposal on the computational implementation of representations. It starts with McCulloch & Pitts (1943) realizing that formal models of simplified neurons can support logic functions. In Mentalese, a straightforward approach to representation is to use one symbol for each primitive concept. For instance, `JOHN` and `MARY` could represent two people called John and Mary; and `INLOVE(x,y)` could represent the relationship of `x` and `y` being in love. It all can be combined to represent the fact that John and Mary are in love as follows.

```
INLOVE(JOHN, MARY)
```

If we were to produce the same representation with simple models of neurons, the naive approach would be to assign one neuron to each concept. There would be neuron `JOHN`, neuron `MARY`, and neuron `INLOVE`. Neurons `JOHN` and `MARY` would somehow be connected through neuron `INLOVE`. Nevertheless, this model has to be abandoned quickly. For starters, research on biological brains shows a very different reality: there is no one to one relationship between concepts and single brain cell activity. Also, each new learning would require assignment of new neurons to the task and a very fast creation of detailed and specific connections, all to produce a structure that would catastrophically fail with the death of a single neuron. Connectionist models have reached a different strategy that better resembles biological nervous systems: distributed representations.

Such a strategy can be exemplified by NETtalk (Sejnowski & Rosenberg, 1987), an artificial neural network that was built and trained from examples to read english text aloud. It achieved good generalization, in the sense that it can do a good job reading new text that was not presented during the training. Representation of categories like vowels vs consonants emerged from training in a way that does not assign one neuron to each. What happened is that the same group of neurons represented the two categories. Then how could categories be told apart? Each category corresponded to a different activation pattern of the same neurons. Information about categories was stored in a distributed fashion in the connections between artificial neurons, no single connection being exclusively associated to a single category. This strategy is the basis for artificial neural networks resilience: neurons and connections can be removed usually without catastrophic damage to function. It also allows such devices to behave nicely in front of new cases.

We must remark that, in practice, artificial neural networks are implemented as software for current digital computers with a von Neumann architecture (Von Neumann, 1993), which is a practical implementation of a universal Turing machine. In turn, universal Turing machines are Turing machines (see section 2.2.4). Therefore, artificial neural networks are not more powerful than Turing machines. Everything that a connectionist model can do, can also be done by a Turing machine that implements the calculations to run a simulation of the model.

10.5. Architecture of the mind

Connectionism offers the possibility of a massively distributed computational implementation of the mind: everything connected to everything. But it does not enforce it. It is also possible to segment the implementation into *modules* that exhibit liberal flow of information inside, but limited interface to the outside. Fodor (1983) introduced the notion of a modular mind. He described modules as *domain specific*, *informationally encapsulated*, *fast*, and *innate*, among other attributes. Domain specificity refers to the processing of only a well delimited kind of information. So there could be modules for color perception, analysis of visual shapes, natural language grammar, etc. Encapsulation means that information processing that occurs within a module only has access to input information that needs to be processed, and whatever is already stored inside the module, with little or no informational access to or from other modules. Encapsulation and domain specificity would allow very fast processing speeds that could explain why some complex processes of human speech, vision, etc. occur so quickly. Language development in human children (Stromswold, 1999), and visual perception to some extent (Spelke, 1994), suggest modularists that modules should be innate. This is because the acquisition of very sophisticated knowledge takes place in little time, subject to what is considered relatively little exposure to examples, and with a time course that is remarkably constant across different cultures.

As restrictive as modularism looks, it actually corresponds to a more relaxed architecture than that of a single head Turing machine which has a central processing unit (its FSM), just like single processor digital computers based on a von Neumann architecture[1] (Von Neumann, 1993; Null, Lobur, et al., 2014). In this sense, traditional TMs are serial, centrally executing one instruction strictly after the other. Modularity resigns the massive parallelism that could be achieved in fully connected neural networks, but still retains some simultaneous processing. It is the informational encapsulation of modules that impedes liberal interchange with a central location, disallowing a central control of the system. The fact that several encapsulated modules work in parallel in a modular system, provides some of its speed advantage and confer it some resiliency, because failure inside a module should have limited effect over other modules.

In summary, we have a range of architectural options for the computational implementation of mental processes, that goes from the completely centralized single head TM, to

[1]Von Neumann architecture is a practical design for digital computers that closely follows the organization of a Turing machine. It is probably inspired in Turing's work, since von Neumann was a visiting professor at Cambridge and met Turing before publication of the halting problem undecidability.

fully connected artificial neural networks made from several similar and simple processing units. Modularity sits in the middle. But in Fodor's version, it also makes some claims about domain specificity, innateness, and which mental functions are modular; that have encountered resistance (Prinz, 2006; Churchland, 1988; McCauley & Henrich, 2006). If we restrict the debate to architecture, the main question is to which extent the mind is modular, with Fodor (1983) himself limiting modularity to low-level cognitive processes, in opposition to some evolutionary psychologists that see it as a pervasive feature of the mind (Cosmides & Tooby, 1992; Pinker, 1997).

10.6. Body and environment

The perspective of *embodied cognition* observes that, in the past, cognitive science has put too much emphasis on formal processes and too little on how those are situated in a physically extended environment (Anderson, 2003). Ignoring the environment generates both, a positive and a negative bias in theories about the mind. Positively, it makes theories too optimistic by not considering the physical constraints an actual living being is subject to. Negatively, it makes theories more complicated than they need to be, because problems that are hard to solve in isolation could be easier to solve with the memory and processing help an actual environment could provide.

We can see some conflicting views on this topic from the beginning of the artificial intelligence program. Von Neumann's cell automata (Von Neumann, 1951), despite simple, where all about the environment of a modeled agent. In contrast, Turing's attempt to answer if machines can think (Turing, 1950) starts by designing a game that excludes flesh and voice from analysis, so that it is all about symbol processing. At any rate, a clear interest in this debate is more recent (Shapiro, 2007; Mahon & Caramazza, 2008; Clark, 1997).

Embodied cognition gives attention to notions like spacial directions, such as "up" or "front" and realizes that they could be anything for an isolated symbol processing agent, only acquiring meaning when used by a being that stands up and has eyes located so to look into some direction in space. A lot of thought has been given to spatiotemporal conceptual metaphors (Lakoff & Johnson, 1980), which are abundant in everyday language and can prove difficult to explain from formal symbol processing alone.

The approach of *enactive congnition* (Varela, Rosch, & Thompson, 1992) is special in several ways. Two of them are directly relevant to our purposes. First, it challenges the view that cognition requires representation, whether it is classical symbol processing as in LOTH, or distributed connectionist representations. It holds that an a priori division between external world objects and internal representations should not be assumed. Instead cognition should be found in the mutual interactions of body and environment. This is a rejection of traditional symbolic computation as an explanation of the mind. Second, it removes some of the *aboutness* that characterizes cognitive theories. It is not about augmenting theories with a theorized mind–body–environment continuity. It is about actually *having* a body, and actually *being* there.

11. INCOMPLETENESS AND EPIPHENOMENAL QUALIA

The fast picture of the CTM that we presented in chapter 10 reveals that, in general, there are two kinds of issues, although they are sometimes presented in an indistinct way. The first ones we will call *computational*[1]. They are about the relationship between computational procedures and an often implicit formalization of observed behaviors. Can computers implement the behavior? If so, what would be a detailed mechanism for that? What are clear similarities and difference between the proposed mechanism and the formalized behavior? We reviewed a few questions that correspond to this description. For instance, how is it possible that the mind can produce an infinity of expressions and yet establish systematic relationships among them. LOTH answers how this can be implemented computationally (section 10.4.0.1). Issues like the architecture of the mind or how to implement representations in consistency with behavioral observations also fall in this category. The second kind of issues we will call *metaphysical*. They refer to the fundamental nature of the mind. Is the mind a correlate of machine states or functional states or both? Does it involve anything more than its physicality? Can it be explained by a relationship between matter and symbolic representations, or does such a relationship assume an unnecessary separation between the mind and an outside world?

Metaphysical issues of the mind are particularly difficult. Take the difficulty of computational questions and add the fact that we have not agreed on the nature of the objects we are talking about. But they are also the most *psychological* issues in a proper sense. Because, in contrast, computational questions do not require our object of study to actually have a mental life, as we will argue in chapter 12. They just require it to have an observable behavior. A position one could take is that some complex behaviors are a sufficient condition for mental life. Nevertheless, the point is that mental life is not necessary for a computational theory, in the same way that an engineer can calculate a bridge regardless of whether a mind emerges from its physical processes or not. This is not to say that

[1]*Mechanical* may be a better name. However it is important at this point to use a sort of neutral term. As used here, the term *computational*, is consistent with the fact that, at an epistemic level, TMs are as powerful a model as they could be.

computational questions are not psychological at all. Models of behavior are essential for predicting the mind, and offer a fertile terrain for proposing ideas about its nature. We must however recognize that some of the most fascinating and proper psychological issues like consciousness, aesthetic pleasure, or the subjective quality of mental life, are currently metaphysical issues.

11.1. The incompleteness of Mary's knowledge

A frustrating aspect of cognitive science is how difficult or even impossible it is to speak formally about some mental phenomena. Jackson's *Epiphenomenal qualia* (Jackson, 1982) made a controversial attempt, by means of asking a question that attempts to be both computational and metaphysical. His goal was to defend the position of *epiphenomenalism*, according to which mental events are caused by physical events, but not the other way around (Huxley, 1882). Jackson's work presents us with the following thought experiment.

> "Mary is a brilliant scientist who is, for whatever reason, forced to investigate the world from a black and white room via a black and white television monitor. She specialises in the neurophysiology of vision and acquires, let us suppose, all the physical information there is to obtain about what goes on when we see ripe tomatoes, or the sky, and use terms like 'red', 'blue', and so on. She discovers, for example, just which wave-length combinations from the sky stimulate the retina, and exactly how this produces via the central nervous system the contraction of the vocal chords and expulsion of air from the lungs that results in the uttering of the sentence The sky is blue'. [...] What will happen when Mary is released from her black and white room or is given a colour television monitor? Will she learn anything or not? It seems just obvious that she will learn something about the world and our visual experience of it. But then it is inescapable that her previous knowledge was incomplete. But she had all the physical information. Ergo there is more to have than that, and Physicalism is false."

Jackson was not the first to ask if any amount of formal knowledge about physical phenomena can replace first person experience (see Russell, 1918; Tye, 1986, for instance). However Jackson's thought experiment has been widely discussed (Dennett, 1993; Jackson, 1986). There are many aspects to this debate apart from those that are relevant here. A deeper discussion can be found in Ludlow, Nagasawa, & Stoljar (2004).

Dennet (1993) defends physicalism against Jackson's position by claiming that his thought experiment is not a proof. In Dennett's eyes, thought experiments are "intuition pumps" that help us conceive new possibilities which must be later confirmed by systematic methods. He focuses in the asseveration that Mary acquires "all the physical information there is to obtain about [color]", arguing that, since no human being can accurately imagine what would it be like to possess that kind of information, what we imagine to be the outcome when Mary is released could also be inaccurate. In order to make his point clear, Dennett offers the following alternate ending to Mary's story.

> And so, one day, Mary's captors decided it was time for her to see colors. As a trick, they prepared a bright blue banana to present as her first color experience ever. Mary took one look at it and said "Hey! You tried to trick me! Bananas are yellow, but this one is blue!" Her captors were dumfounded. How did she do it? "Simple," she replied. "You have to remember that I know everything–absolutely everything–that could ever be known about the physical causes and effects of color vision. So of course before you brought the banana in, I had already written down, in exquisite detail, exactly what physical impression a yellow object or a blue object (or a green object, etc.) would make on my nervous system. So I already knew exactly what thoughts I would have (because, after all, the 'mere disposition' to think about this or that is not one of your famous qualia, is it?). I was not in the slightest surprised by my experience of blue (what surprised me was that you would try such a second-rate trick on me). I realize

> it is hard for you to imagine that I could know so much about my reactive dispositions that the way blue affected me came as no surprise. Of course it's hard for you to imagine. It's hard for anyone to imagine the consequences of someone knowing absolutely everything physical about anything!"

This new ending shifts the focus from the original question, if Mary would learn anything new from first person color experience, to a new one: will Mary realize the banana trick? Dennett's story makes several unlikely assumptions, just like Jackson's, but we should avoid discussing them because that would deviate us from the spirit of this debate. The question is whether knowledge by direct experience goes beyond the best of formal knowledge about the physical. A non-trivial assumption made by Dennett is that full formal knowledge about the *cognitive machine* (i.e., Mary's nervous system) allows Mary to predict the effects of some input (i.e., information from visualizing the banana). In chapter 4 we could see that there are many things we cannot predict about the behavior of a machine, even if we had full access to its design and input data. This leaves us with the only possibility of *running the machine* and seeing what happens. Since it could well be that Mary cannot (in general) predict what the effects of the banana input would be in her nervous system, Dennett's point is not solid. It may be necessary for Mary's body to be exposed to the experience of color before some particular computational process of her nervous system can take place in her and be observed.

Let us say that Mary possesses *the* axiomatic system of the neurophysiology of vision plus a database of all known facts about it. We will call it system F_V. In other words if something can be proved about vision, Mary can do it. We will also assume that processing time is not a problem, and that Mary owns a device that allows observation of brain activity with all the spacial and temporal resolution she may require for this experiment. Whether the trick banana produces brain activity consistent with the perception of yellow corresponds to a statement S_Y that can be written in the F_V system. In the spirit of Dennett's argument the extensive power we have given to Mary should allow her to formally find the truth value of S_Y in order to determine if the banana is actually yellow. But system

F_V must include arithmetic. Therefore, it must be incomplete, and there is no guarantee that the a truth value can be computed for S_Y.

Gödel's incompleteness opens the possibility that all the formal knowledge about a topic is not enough to prove everything about that topic. As always, a point of view from TMs is more material and more insightful. Undecidability of Turing halting means than some behaviors of machines cannot be predicted, they need to occur with physical extension. In turn, if we modeled the mind as a computer, this would mean that we would not always be able to predict what is *true* about proesses. Since we are talking about a mind, this is consistent with the idea of experience as *being there*. But it does neither prove nor disprove that there is something beyond the physical. It only shows how neither Jackson's proposal nor Dennett's refutation can settle the issue.

We have just scratched a metaphysical issue. We know that computation requires a physical extension (i.e., hardware) to run. But we can also speak of an unextended design that correlates with the organization of that physical extension (i.e, software). Could the mind be the software aspect to the brain–body–environment? By virtue of Turing halting, software is not enough to predict everything about the processes it determines on the machine. So if the mind is modeled as a computer, and the actual becoming of its processes is part of its essence, then it cannot occur as an unextended aspect to the machine. It must have an extension.

12. A FEW LIMITATIONS OF THE COMPUTATIONAL THEORY OF MIND

Based on results reviewed in previous chapters, here we summarize a few claims we can now make regarding limitations of the computational theory of mind (CTM). Remember that this view holds that, in some sense, the mind is a computer. We will take this asseveration as equivalent to the following one: the mind implements an instance of a Turing machine model. With this new wording the claim is neither stronger nor weaker than the original. It is not weaker because the original wording implies the new one: the fact that something is an actual computer implies that there is a Turing machine model for it. We base this on the assumption that the Church–Turing thesis is reasonable and has solid support (see the end of section 2.2.3). Also, the new wording is not stronger because, in the opposite direction, it implies the original one: if something can be modeled as a Turing machine, then it is a computer.

It may seem like the previous analysis is incomplete in the sense that the Turing machine has a particular architecture (see section 2.2.3, and figure 2.7b). In contrast the notion of computer allows diverse architectures. Therefore, it may seem like a wording that speaks of a Turing machine corresponds to a stronger claim. But since the power of the Turing model of computation is maximal in practice (as follows from Church–Turing thesis), this is not an issue for our purposes. It is not because of two reasons.

First, as metatheoretical frameworks, *computational mind* and *Turing machine mind* allow theories with the exact same potential predictive power, which follows from the fact that both correspond to the exact same class of computational power. In other words, a behavior that cannot be exhibited in one framework, neither can in the other.

Second, it is because of its remarkable power that the Turing machine can model the architecture of any computer that can actually be implemented. If this were not the case, then we would have a computer with more power than a Turing machine, which would violate the Church–Turing thesis. In consequence, there are two architectures to consider apart from the organization of the computational process that is being modeled: the serial structure of Turing machines, and the architecture that underlies the *programming* of the particular Turing machine model instance.

Remember that a Turing machine can, for instance, implement an artificial neural network, which corresponds to a parallel architecture (see end of section 10.4.0.2). Under reasonable assumptions, neural networks can also implement Turing machines (Hyötyniemi, 1996), specially if we couple them to an environment that provides potentially unlimited information storage space for any practical purposes. The fact that a Turing machine can model any implementable computer architecture means that it does not enforce a particular architecture on the modeled object. This is why researchers of neural networks can derive their theoretical predictions from neural network simulations that run on their digital computers (e.g., Sejnowski & Rosenberg, 1987; Deng et al., 2009).

12.1. Some basic limitations

The following are the most direct limitations that we can derive from computability theory.

(i) *A process that has a formal definition may not have a mechanical implementation.* It follows from Turing halting undecidability and its analysis in section 4.3.3, as visualized in figure 4.1, that a process can have a formal definition but be impossible to realize.

(ii) *In general, there is no way to know if a given mechanical process will ever reach a particular state we may be interested in.* If a systematic method for solving this problem existed, this would allow us to solve the Turing halting problem. Turing proved that such a thing is not possible (chapter 4).

(iii) *In general, there is no way to know if two formal descriptions of mechanical processes predict the same behavior.* If we had a systematic method that solved this problem, we could solve the Turing machine equivalence problem in particular. But as a consequence of Turing halting undecidability, this problem cannot have a solution (section 4.3.2).

These limitations may have important consequences for cognitive neuroscience. Even if we had a complete description of causal relationships in the nervous system, full body, and environment of a subject; there is no general method to predict that a particular state will be reached. Consequently, there is a high chance that a full theory of behavior is not attainable from the study of nervous systems. Additionally, there is no general method to determine if a nervous system, or any subset of it, matches a function we can formally define. This may limit the analytic understanding of the nervous system and its functions.

12.2. Underdetermination of theoretical models by evidence

The fact that several different models can predict the same data is not a new issue. Descartes (1986, first published in 1641) doubted everything he believed, under the supposition that there could be a Evil Demon devoted to present him false illusions of the world. His experiences would be the same whether they came from a true external world or the deceptions of the demon. Therefore evidence was not enough to determine if a true external world or an Evil Demon thesis was true. The same issue has been discussed as a widely applicable problem in science (Duhem, 1991; Quine, Churchland, & Føllesdal, 2013).

In an optimistic and naive manner, we could expect the power of Turing machines to rescue us from underdetermination, on the grounds that different models for the same evidence could be computationally equivalent in the sense that they generate the same predictions. In other words, it could be hoped that those models are not different in a relevant way. Unfortunately, this is not the case.

The fundamental limitation is that, however large, any record we can make of observations is finite. And a computer can model any given piece of finite information. Furthermore, it can do so in an infinite amount of relevantly different ways. In order to visualize this, consider a single piece of past observations. Consider too a finite random sequence of events that could be observed in the future. Now append the random sequence to past observations. We can program a computer to model the result. Since we can append random events in infinitely many ways, and we can program computers for each resulting sequence. We have just created an infinite amount of theories that are all equally right about observations we currently possess, but make different predictions about the future.

However, we can hope for theories that partially capture regularities in current observations, in order to produce sensible predictions. Once again, artificial neural networks offer a concrete example. They can be inductively trained to recognize members of a class[1] (Chauvin & Rumelhart, 1995). They do not need to be previously presented with all members of the class, of which there could be infinitely many. After the presentation of (and repeated training with) a few instances, the network often acquires the capability of successfully recognizing members that where not presented before. But this has a few conditions. When a network has too many artificial neurons, and therefore two many connections, its number of parameters is big enough to "learn examples by heart". This makes it very easy for the network to model training instances, but does not force it to capture the underlying similarities that make those part of a class. As a consequence, the network fails in front of new examples. On the opposite end, when a network has too few artificial neurons, it lacks the complexity needed to capture the essential features of the class. But when the number of units is just right, and the amount of training is just enough, artificial neural networks generalize surprisingly well. Nevertheless, the fact remains that any set of finite observations can be modeled by a computer, actually several different ones that, depending on our luck, could make slightly or wildly different predictions.

12.3. Metaphysical orthogonality of the computational theory of the mind

That the human mind is a computer is a metaphysical claim. If we accept it, it follows that metaphysical properties of computers are also properties of the human mind. But the same properties will apply to other objects like a digital gaming console, which is not a human mind. What metaphysical properties are specific of the human mind? The CTM cannot be used to decide this. Since several different computational models can be consistent with any observations we may have, it will always be the case that the same

[1]An actual solution of a problem by using artificial neural networks would involve several classes and training with several instances of each class. This paragraph focuses on one class for simplicity.

evidence will allow models with different properties, whether metaphysically relevant or not.

Consider physicalism or epiphenomenalism, for instance, as they would be treated in the context of the CTM. One holds that the mind supervenes on the physical. The other that mental events are non-physical but caused by physical events. As much as, strictly speaking, the two theories do make different metaphysical claims, we could sensibly say that those are not really predictions but assumptions. Should any other metaphysical claims be theoretically derived from the sum of metaphysical assumptions and computational models in such a theory, those could be considered predictions, strictly speaking. But not in a sense that allows us to tell theories apart, because they would be the just the consequence of metaphysical assumptions.

We could propose, just as an example, that the nature of the mind is such that a necessary condition for it is to process information in a massively parallel way as opposed to a serial Turing machine. And predictions about what can be observed would follow, possibly including metaphysically relevant consequences. But since (massively parallel) artificial neural networks can model anything, the claim could not be falsified. In order to propose a falsifiable metaphysical claim within the computational theory of mind, we would need to restrict acceptable models to a computational power class strictly weaker than Turing machines. In principle this does not seem like a problem. A computer that does not harness its full potential is still a computer. But human minds can simulate Turing machines. And a weaker model (than Turing machines) cannot. By reductio ad absurdum, within the computational theory of mind, we cannot propose a falsifiable metaphysical claim that is specifically about the mind or about any other object of which we can only make a finite amount of observations.

We will have to limit ourselves to non-specifically mental metaphysical properties. We can say that if the mind *is* a computer, then it needs to be so in a physically extended way. And the same is true of any computer, whether it is a mind or not. This is a metatheoretical consequence of Turing halting. Since computational theories are incomplete, the only place in which a machine can actually possess *all* the attributes that its design determine, is in its physical extension, with time for allowing the execution of the machine's processes, and space that allows the determined patterns to actually take place.

13. CONCLUSIONS: THE CONSEQUENCES OF THEORETICALLY MODELING THE MIND AS A COMPUTER

The typical image of a computer characterizes it as a dumb device that gets its intelligence but also its errors from a programmer. Perfectly logical, fast, and rigid. Incapable of learning or adapting; any deviation from the expected would cause it to explode. Since software can be copied, several computers can behave the same way. They do not seem too diverse. If such an obnoxiously logical device was used to attempt the creation of an artificial mind, it would probably be all reason but no heart, brilliant at maths but its jokes would be bad. It could probably understand quantum mechanics, but not a flirtatious look.

Our revision presents a different image. Computers[1] can learn and evolve on their own, make generalizations, deal with the uncertain, be diverse, etc. In a certain sense, it is problematic that computers are *that* powerful. They can model any finite set of observations, which means they can simulate any piece of observable behavior we can sensibly record. Consider that there is no way for us, as observers of a reality, to make infinite records of a behavior in a finite amount of time.

Then, in which sense is this problematic? This would prima facie seem like a solution for building successful models of behavior. A hint can be found in the example of a teacher training a pupil to learn the rules for a number series. Even if the pupil has correctly written down several numbers of the series, how many more numbers are needed so that we know the pupil has mastered the right rules (Wittgenstein, 1953, remark 145)? In the case of "training" a computational model, the fact that it has successfully modeled a finite amount of observations does not guarantee that it will continue to do so when new observations come in the future. Since a computer can model *any* finite sequence of observations, in particular it can model an infinite amount of sequences that begin *correctly* and then go on in any of infinitely possible *wrong* ways. See section 12.2 for a more detailed discussion.

[1]Because of the arguments exposed in chapter 2, our use of the word *computer* in this chapter corresponds to a notion of a Turing machine. More properly, to any computer machine that can be modeled as a Turing machine.

Excessive modeling power is also a problem when we use theoretical models with the purpose of learning something about the nature of a studied object. Because a computer can model any finite piece of observable human behavior, there is a naive way in which human mind and computers could be related[2]: since the model works, then the nature of the mind could be that it behaves like a computer. But a computer can also model the behavior of a thermostat or the chemical reactions that occur in a ripening apple. Since a computer can model finite behavior of about anything, how does that say anything specific about the nature of any modeled thing? A more in depth analysis of this issue can be found in section 12.3.

A psychological model can be implicitly computational. Take the *bystander effect* for instance: people are less likely to help a victim when others are present, probably because of a lowered sense of responsibility (Darley & Latane, 1968). A simple logic for this phenomenon is shown in algorithm 2.

Algorithm 2 Bystander effect. N is the number of bystanders, including the modeled person. Note that the model does only consider groups of 1, 2, or 4 people. The RANDOM procedure gives a result that tends to vary a lot, being true with a frequency that corresponds to its argument given as a percentage, and false in the rest of cases

```
1: procedure REACTION($N$)
2:     if $N == 1$ then
3:         if RANDOM($85\%$) then
4:             Report the issue with a delay of about 1 minute.
5:     else if $N = 2$ then
6:         if RANDOM($62\%$) then
7:             Report the issue with a delay of about 1.5 minutes.
8:     else if $N = 4$ then
9:         if RANDOM($31\%$) then
10:            Report the issue with a delay of about 2.5 minutes.
```

Since computers can model any finite amount of data, in particular they can be made as good as any finite description of a finite amount of data. In the terminology of computer

[2]This does not mean that any mind–computer relationship proposal is naive.

science, a psychological model of observable facts should correspond to a *non-empty partial recursive function*. *Non-empty* means that it has to provide a prediction for at least one case. *Partial function* means that it is allowed not to make predictions for some cases. *Recursive function* means that when it does, there must be an effective mechanical procedure (i.e., a Turing machine) that calculates those predictions unambiguously. Otherwise, we cannot fully agree on how the theory relates to observations, past or predicted. The set of predictions made by the model is, generally speaking, *recursively enumerable*. This means that there is a Turing machine that can generate all the potentially infinite predictions of the model, one after another. But if we start with a possible outcome, and ask if it is among the predictions of the model, there is no guarantee that this can be formally answered. This is a consequence of the undecidability of Turing halting. We could wait forever for the machine to produce the prediction, without knowing if it ever will.

To any model that generates behavioral or physiological (or in any way measurable) predictions from a finite amount of data in a *systematic* way, there is a corresponding Turing machine. Here, systematic means that predictions are not any arbitrary sequence of events, but are related to and derived from observations by some method. Just like there is a gap that separates a model and the modeled, there is a gap that separates *questions about the model* and the *model itself*.

At this point it is natural to question that the bystander effect model, or any other model of comparable simplicity is a model of the mind. At most, it is a model of a phenomenon that occurs to people with minds, which is what makes it psychologically relevant. Nevertheless, we could argue that any other model that targets psychological phenomena at a measurable level, being comprehensive and explanatory enough for whatever the purposes of the modeler are, will have the property of generating systematic predictions from a finite amount of data. In conclusion, if such a thing as a systematic model of the mind based on a finite amount of observations exists, it is computational, even if it is implicitly so.

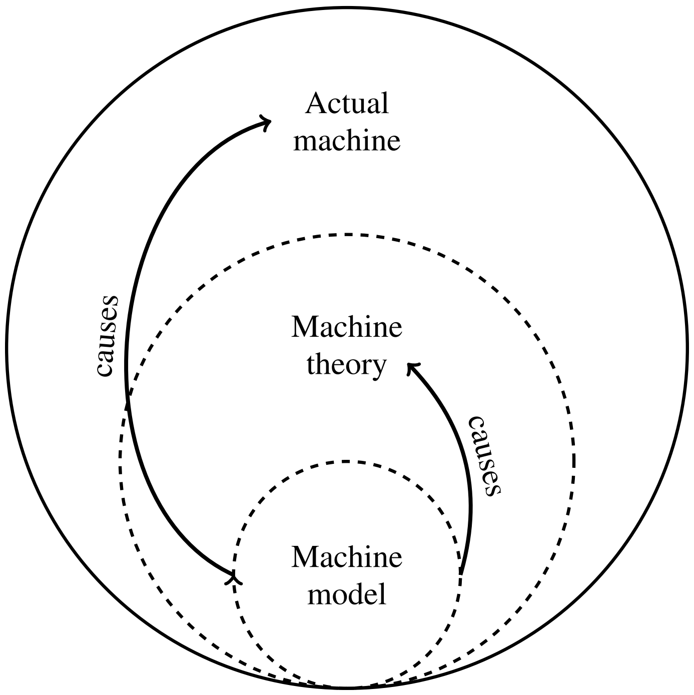
In order to understand the consequences of modeling the mind as a computer, we need to understand a deep epistemological issue regarding computational models. A consequence of Turing halting undecidability is that there is a surprising and inescapable gap between the actual behavior of a machine and what can be predicted about it through modeling. It is surprising because the same model that fully determines the operation of the machine has limits regarding the predictions it can make about that same operation. How can this be possible? In order for a prediction to be useful, it has to be generated in a finite amount of time. But there are claims about formal processes that requiere infinite time in order to be falsified. Turing halting undecidability proof forces that, in many cases, there is not a finite-time alternative method that can provide us a correct answer.

All we can do is run the machine and see what happens. To that end, it is required that the machine has a physical extension. Or at least that an agent with the capability of simulating the machine, be human or machine, has a physical extension. But even if we run the machine we may never get to know if it will ever reach the state we are interested in. Because if the state has not occurred yet, there is no guarantee that it will occur, nor that it will not.

Figure 13.1 displays the situation. The theory does not fill the modeled object. No surprise there. But note that what the theory cannot answer corresponds to facts caused by the model of the machine, which is a subset of the theory. We are not asking questions like what the color of the machine is or how much it weights. We are asking, for example, if a particular state is among those that the model determines to take place at some time (given a particular input to the machine).

The computational theory of mind holds that, in some sense, the mind is a computer. Turing halting undecidability dictates that the predictive power of a theory that adheres to that claim is necessarily incomplete respect to its object of study (as depicted in figure 13.1). As a consequence, the same is true of its explanatory power: more phenomena will occur

FIGURE 13.1. Relationships between a model of a machine, what we can answer about it (theory), and the actual machine. Dashed circles inside the solid circle reflect to which extent the model and the theory can speak about the machine.



than those that the theory can explain. This conclusion can be extended to any psychological theory that only makes systematic predictions strictly based on finite observations. And *finite* observations are about the only kind we can make.

Computational models are not only incomplete in the sense we just mentioned. They are also very limited as theoretical tools when discussing the nature of modeled objects in a specifically psychological manner, as discussed in section 12.3. Therefore, for a metatheoretical framework to accommodate psychological theorization in a proper and comprehensive way, it must allow more than modeling observations. Formalization is useful, but it is not enough.

Modeling is highly important. It allows us to make predictions and to explain. At the same time it helps us understand. But it is fundamentally incomplete. A program seeking to base psychology exclusively in formal methods and measurable data can definitely produce advances, but it would be too restrictive for the whole of psychology. The consequence of theoretically modeling the mind as a computer is that we need to go beyond computational models.

References

Aho, A. V., & Ullman, J. D. (1977). *Principles of compiler design*. Addison-Wesley.

Anderson, M. L. (2003). Embodied cognition: A field guide. *Artificial intelligence*, *149*(1), 91–130.

Astrom, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation. *Journal of mathematical analysis and applications*, *10*(1), 174–205.

Atrash, A., & Pineau, J. (2010). A bayesian method for learning POMDP observation parameters for robot interaction management systems. In *The POMDP practitioners workshop.* ICAPS 2010, Toronto, Canada.

Baker, C., Saxe, R., & Tenenbaum, J. (2011). Bayesian theory of mind: Modeling joint belief-desire attribution. In *Proceedings of the Cognitive Science Society* (Vol. 33).

Ball, W. (2010). *A short account of the history of mathematics*. Dover Publications.

Bays, T. (2004). On Floyd and Putnam on Wittgenstein on Gödel. *The Journal of Philosophy*, *101*(4), 197–210.

Bennett, C. H. (1973). Logical reversibility of computation. *IBM journal of Research and Development*, *17*(6), 525–532.

Bennett, C. H., Bernstein, E., Brassard, G., & Vazirani, U. (1997). Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, *26*(5), 1510–1523.

Berggren, J. L., Borwein, J. M., & Borwein, P. (2004). *Pi: a source book*. Springer.

Bernstein, E., & Vazirani, U. (1997). Quantum complexity theory. *SIAM Journal on Computing*, *26*(5), 1411–1473.

Block, N. (1995). The mind as the software of the brain. *New york*, *3*, 377–425.

Boring, E. G. (1933). *The physical dimensions of consciousness.* The Century Co.

Cantor, G. (1874). Ueber eine eigenschaft des inbegriffs aller reellen algebraischen zahlen. *Journal für die reine und angewandte Mathematik*, *77*, 258–262.

Cantor, G. (1891). Uber eine elementare frage der mannigfaltigkeitslehre. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, *1*(1), 75–78.

Carnap, R., Frank, P., & Schlick, M. (1934). *Logische syntax der sprache*. J. Springer.

Cassandra, A. R., & Kaelbling, L. P. (2016). Learning policies for partially observable environments: Scaling up. In *Machine learning proceedings 1995: Proceedings of the twelfth international conference on machine learning, tahoe city, california, july 9-12, 1995.*

Chater, N., & Oaksford, M. (2008). *The probabilistic mind: Prospects for bayesian cognitive science*. OUP Oxford.

Chauvin, Y., & Rumelhart, D. E. (1995). *Backpropagation: theory, architectures, and applications*. Psychology Press.

Ching, W.-K., Huang, X., Ng, M. K., & Siu, T.-K. (2013). Introduction. In *Markov chains* (pp. 1–46). Springer.

Chomsky, N. (1957). *Syntactic structures.* Mouton.

Chomsky, N., & Miller, G. A. (1963). Introduction to the formal analysis of natural languages. In R. D. Luce, R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology* (Vol. 2, pp. 269–322). New York: Wiley.

Church, A. (1936). An unsolvable problem of elementary number theory. *American journal of mathematics*, *58*(2), 345–363.

Churchland, P. M. (1988). Perceptual plasticity and theoretical neutrality: A reply to Jerry Fodor. *Philosophy of Science*, *55*(2), 167–187.

Clark, A. (1997). Being there: Putting mind, world, and body back together. *Cambridge, MA*.

Copeland, B. J. (2002). Accelerating turing machines. *Minds and Machines*, *12*(2), 281–300.

Cosmides, L., & Tooby, J. (1992). Cognitive adaptations for social exchange. *The adapted mind: Evolutionary psychology and the generation of culture*, *163*, 163–228.

Cox, R. T. (1946). Probability, frequency and reasonable expectation. *American journal of physics*, *14*(1), 1–13.

Cox, R. T. (1961). *Algebra of probable inference*. JHU Press.

Darley, J. M., & Latane, B. (1968). Bystander intervention in emergencies: diffusion of responsibility. *Journal of personality and social psychology*, *8*(4), 377.

Davidson, D. (2001). *Essays on actions and events: Philosophical essays*. Clarendon Press.

Dawson, J. (2006). Shaken foundations or groundbreaking realignment? A centennial assessment of Kurt Gödel's impact on logic, mathematics, and computer science. In *Logic in computer science, 2006 21st Annual IEEE Symposium on* (pp. 339–341).

De Finetti, B. (2017). *Theory of probability: A critical introductory treatment* (Vol. 6). Wiley.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer vision and pattern recognition, 2009 IEEE Conference on* (pp. 248–255).

Dennett, D. C. (1993). *Consciousness explained.* Penguin UK.

Descartes, R. (1986). *Meditations on first philosophy with selections from the objections and replies (J. Cottingham, Trans.).* Cambridge: Cambridge University Press.

Duhem, P. (1991). *The aim and structure of physical theory*. Princeton University Press.

Euclid, & Williamson, J. (1788). *The Elements of Euclid*. Printed at the Clarendon Press.

Feigl, H. (1958). The 'mental'and the 'physical'. *Minnesota studies in the philosophy of science*, *2*(2), 370–497.

Floyd, J., & Putnam, H. (2000). A note on Wittgenstein's "notorious paragraph" about the Gödel theorem. *The Journal of philosophy*, *97*(11), 624–632.

Fodor, J. A. (1975). *The language of thought* (Vol. 5). Harvard University Press.

Fodor, J. A. (1983). *The modularity of mind: An essay on faculty psychology*. MIT press.

Formica, G. (2011). Almost von Neumann, definitely Gödel: The second incompleteness theorem's early story. *Logic and Philosophy of Science*, *9*(5), 151-158.

Frege, G. (1893). *Grundgesetze der Arithmetik*. H. Pohle.

Gallistel, C., & King, A. (2011). *Memory and the computational brain: Why cognitive science will transform neuroscience*. Wiley.

Gandy, R. (1980). Church's thesis and principles for mechanisms. *Studies in Logic and the Foundations of Mathematics*, *101*, 123–148.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für mathematik und physik*, *38*(1), 173–198.

Heisenberg, W. (1927). Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik A Hadrons and Nuclei*, *43*(3), 172–198.

Heisenberg, W. (1949). *The physical principles of the quantum theory*. Courier Corporation.

Hintikka, J. (2000). *On wittgenstein*. Wadsworth/Thomson Learning.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society* (Vol. 1, p. 12).

Hodges, A. (2005). Can quantum computing solve classically unsolvable problems? *arXiv:quant-ph/0512248v1*.

Hofstadter, D. (1980). *Gödel, escher, bach: An eternal golden braid*. Penguin Books.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). *Introduction to automata theory, languages, and computation*. Pearson.

Huxley, T. H. (1882). On the hypothesis that animals are automata, and its history. In *Meeting of the British Association for the Advancement of Science, 1874, Belfast.*

Hyötyniemi, H. (1996). Turing machines are recurrent neural networks. *Proceedings of STeP*, *96*.

Jackson, F. (1982). Epiphenomenal qualia. *The Philosophical Quarterly*, *32*(127), 127–136.

Jackson, F. (1986). What Mary didn't know. *The Journal of Philosophy*, *83*(5), 291–295.

Kleene, S. C. (1938). On notation for ordinal numbers. *The Journal of Symbolic Logic*, *3*(4), 150–155.

Kleene, S. C. (1943). Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, *53*(1), 41–73.

Kleene, S. C. (1952). *Introduction to metamathematics*. Wolters-Noordhoff.

Kleene, S. C. (1967). *Mathematical logic*. Wiley.

Lakoff, G., & Johnson, M. (1980). *Metaphors we live by*. University of Chicago Press.

Lewis, D. (1980). Mad pain and martian pain. *Readings in the Philosophy of Psychology*, *1*, 216–222.

Lewis, D. K. (1966). An argument for the identity theory. *The Journal of Philosophy*, *63*(1), 17–25.

Littman, M. L. (2009). A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, *53*(3), 119–125.

Lucas, J. R. (1961). Minds, machines and gödel. *Philosophy*, *36*(137), 112–127.

Ludlow, P., Nagasawa, Y., & Stoljar, D. (2004). *There's something about Mary: essays on phenomenal consciousness and Frank Jackson's knowledge argument*. MIT Press.

Mahon, B. Z., & Caramazza, A. (2008). A critical look at the embodied cognition hypothesis and a new proposal for grounding conceptual content. *Journal of physiology-Paris*, *102*(1), 59–70.

Markov, A. A. (1906). Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, *15*(135-156), 18.

Marr, D. (1983). *Vision: A computational investigation into the human representation and processing of visual information*. Henry Holt and Company.

McCauley, R. N., & Henrich, J. (2006). Susceptibility to the Müller-Lyer illusion, theory-neutral observation, and the diachronic penetrability of the visual input system. *Philosophical Psychology*, *19*(1), 79–101.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Moser, P., & Trout, J. (2002). *Contemporary materialism: A reader*. Taylor & Francis.

Murawski, R. (1998). Undefinability of truth. The problem of priority: Tarski vs Gödel. *History and Philosophy of Logic*, *19*(3), 153–160.

Newell, A., Shaw, J. C., & Simon, H. A. (1959). Report on a general problem-solving program. In *IFIP congress* (pp. 256–264).

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, *19*(3), 113–126.

Null, L., Lobur, J., et al. (2014). *The essentials of computer organization and architecture*. Jones & Bartlett Publishers.

Ord, T. (2002). *Hypercomputation: computing more than the turing machine.*

Peano, G. (1889). *Arithmetices principia: nova methodo*. Fratres Bocca.

Pekonen, O., Franzèn, T., & Mar, G. (2007). Godel's theorem: an incomplete guide to its use and abuse. *The Mathematical Intelligencer*, *29*(2), 66–70.

Penrose, R. (1994). *Shadows of the mind: A search for the missing science of consciousness*. Oxford University Press.

Penrose, R. (1999). *The emperor's new mind: concerning computers, minds, and the laws of physics*. Oxford University Press.

Piaget, J. (1970). *Structuralism.* ERIC.

Pinker, S. (1997). How the mind works. 1997. *NY: Norton*.

Pinker, S. (2005). So how does the mind work? *Mind & Language*, *20*(1), 1–24.

Place, U. T. (1970). Is consciousness a brain process? In *The mind-brain identity theory* (pp. 42–51). Springer.

Post, E. L. (1936). Finite combinatory processes—formulation. *The Journal of Symbolic Logic*, *1*(03), 103–105.

Post, E. L. (1944). Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, *50*, 284-316.

Prinz, J. (2006). Is the mind really modular. *Contemporary debates in cognitive science, ed. RJ Stainton*, 22–36.

Putnam, H. (1967). Psychological predicates. *Art, mind, and religion*, *1*, 37–48.

Putnam, H. (1988). Representation and reality. *Cambridge, Mass.: A Bradford Book*.

Quine, W. V. O., Churchland, P. S., & Føllesdal, D. (2013). *Word and object*. MIT press.

Rang, B., & Thomas, W. (1981). Zermelo's discovery of the "Russell Paradox". *Historia Mathematica*, *8*(1), 15–22.

Rumelhart, D. E., McClelland, J. L., & PDP Research Group, C. (Eds.). (1986). *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1: Foundations*. Cambridge, MA, USA: MIT Press.

Russell, B. (1918). The philosophy of logical atomism. *The Monist*, 495–527.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and brain sciences*, *3*(03), 417–424.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Complex systems*, *1*(1), 145–168.

Shannon, C. E. (1948). A mathematical theory of communication, part I, part II. *Bell Syst. Tech. J.*, *27*, 623–656.

Shapiro, L. (2007). The embodied cognition research programme. *Philosophy compass*, *2*(2), 338–346.

Smart, J. J. (1959). Sensations and brain processes. *The Philosophical Review*, *68*(2), 141–156.

Spelke, E. (1994). Initial knowledge: Six suggestions. *Cognition*, *50*(1), 431–445.

Stromswold, K. (1999). Cognitive and neural aspects of language acquisition. In *M. Gazzaniga (ed.), The cognitive neurosciences.*

Tarski, A. (1936). Der wahrheitsbegriff in den formalisierten sprachen. *Studia Philosophica*, *1*, 261–405.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Journal of Mathematics*, *58*, 345–363.

Turing, A. M. (1937). Computability and λ-definability. *The Journal of Symbolic Logic*, *2*(04), 153–163.

Turing, A. M. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, *2*(1), 161–228.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460.

Tye, M. (1986). The subjective qualities of experience. *Mind*, *95*(377), 1–17.

Varela, F., Rosch, E., & Thompson, E. (1992). *The embodied mind: Cognitive science and human experience*. MIT Press.

Von Neumann, J. (1951). The general and logical theory of automata. *Cerebral mechanisms in behavior*, *1*(41), 1–2.

Von Neumann, J. (1993). First draft of a report on the EDVAC. *IEEE Annals of the History of Computing*, *15*(4), 27–75.

Wagner, P. (2009). *Carnap's logical syntax of language*. Springer.

Weyl, H. (1921). Über die neue Grundlagenkrise der Mathematik. *Mathematische zeitschrift*, *10*(1), 39–79.

Wittgenstein, L. (1953). *Philosophical investigations*. Oxford: Basil Blackwell.

Wittgenstein, L., & Russell, B. (1921). Logisch-philosophische abhandlung.

Wittgenstein, L., von Wright, G. H., Rhees, R., Anscombe, G. E. M., & Anscombe, G. E. M. (1978). *Remarks on the foundations of mathematics*. MIT press Cambridge, MA.