# TURING MACHINES, COMPUTERS, AND ARTIFICIAL INTELLIGENCE

*PETER R. KREBS*

A thesis submitted in part fulfillment
of the requirements for the degree of
Master of Arts

University of New South Wales
November 2002

## Abstract

This work investigates some of the issues and consequences for the field of artificial intelligence and cognitive science, which are related to the perceived limits of computation with current digital equipment. The Church-Turing thesis and the specific properties of Turing machines are examined and some of the philosophical 'in principle' objections, such as the application of Gödel's incompleteness theorem, are discussed. It is argued that the misinterpretation of the Church-Turing thesis has led to unfounded assumptions about the limitations of computing machines in general. Modern digital computers, which are based on the von Neuman architecture, can typically be programmed so that they interact effectively with the real word. It is argued that digital computing machines are supersets of Turing machines, if they are, for example, programmed to interact with the real world. Moreover, computing is not restricted to the domain of discrete state machines. Analog computers and real or simulated neural nets exhibit properties that may not be accommodated in a definition of computing, which is based on Turing machines. Consequently, some of the philosophical 'in principle' objections to artificial intelligence may not apply in reference to engineering efforts in artificial intelligence.

# Table of contents

# Introduction

The concept of the Turing machine is of great importance for computer science, especially in the context of computational theory. The Turing machine and the Church-Turing thesis are often used as essential criteria for a definition of computation itself. Artificial intelligence (AI) as a field of inquiry within computer science is not only a theoretical enterprise, but also an engineering discipline. Some of the philosophical arguments against AI seem to rise from misunderstandings in the theoretical foundations and from misinterpretations of the properties of Turing machines in particular. Lucas (Lucas 1964, Lucas 1970), Searle (Searle 1980, Searle 1990a) and Dreyfus (Dreyfus & Dreyfus 1986, Dreyfus 1992) have argued against the possibility of engineering intelligent systems for different reasons. Lucas's arguments in particular are based on the properties of Turing machines. In this thesis I argue that modern electronic computers are not subject to some of the perceived limitations of Turing machines. Moreover, an artificial intelligence based on modern computing machines is not necessarily constraint by the properties Turing of machines.

The thesis begins with a short historical account of the origins and developments in the field of artificial intelligence. In chapter two I will discuss the origins of the mathematico-logical definition of computation. Some of the problems with this narrow definition of computation, which have been suggested in the literature (Sloman 2002, Copeland & Sylvan 1999), are examined. It will be shown that the Church-Turing thesis is frequently misinterpreted and I will argue against some of the objections to an artificial intelligence, where these objections are based on misinterpretations (e.g. Kurzweil 1990). In chapter three I examine the essential properties of Turing machines, and I argue that modern computers are super-sets of Turing machines. I will show that some philosophical objections, like Lucas's Gödelian argument against mechanism (Lucas 1964, Lucas 1970), do not necessarily apply to all computing machines. In chapter four and chapter five I outline the main features of classical artificial intelligence and connectionism in relation to computation. Some of the similarities and differences between the serial and the parallel approaches to artificial intelligence in terms of Turing machines are discussed in chapter six. A summary of the main arguments is presented in the conclusion.

# Chapter 1

# Historical Background.

Questions about human thought and cognition have been asked since antiquity. Plato (ca. 428-348 B.C.) and Aristotle (ca. 384-322 B.C.) considered the relationships between the body and an immortal soul or mind. Galen (A.D. 131 – 201) investigated the human anatomy [1] and proposed a theory, which had natural and animal spirits flowing through a system of hollow nerves (Singer 1959). However, the scientists and thinkers of the Renaissance were arguably the first to connect the activities of the human brain and the human mind from a philosophical viewpoint. René Descartes (1596-1650) maintained a separation of the mind and the physical body, and his philosophical investigations in relation to the mind-body problem were perhaps the most clearly articulated. They have maintained their relevance until today as a prime reference for Dualism. Thomas Hobbes (1588-1679), a contemporary of Descartes, did not offer any detailed theory of mind in mechanical or physical terms, but we can see in Hobbes's *Leviathan* of 1651, what must be recognized as *the* pre-cursor of the symbol-processing hypothesis.

When a man *Reasoneth*, hee does nothing but conceive a summe totall, from *Addition* of parcels; or conceive a Remainder, from *Subtraction* of one summe from another; … For as Arithmeticians teach to adde and subtract in *numbers*; so the Geometricians teach the same in *lines, figures, …* The Logicians teach the same in *Consequences of words;* adding together *two Names*, to make an *Affirmation* … and Lawyers, *Lawes,* and *facts*, to find what is *right* and *wrong* in the actions of private men. In summe, in what matter soever there is place for *addition* and *subtraction*, there is also place for *Reason*; and where these have no place, there *Reason* has nothing at all to do. … For REASON, in the this sense is nothing but *Reckoning* (that is, Adding and Subtracting) of the Consequences of generall names agreed upon, for the *marking* and *signifying* our thoughts; … (Hobbes 1914, 18)

[1] Galen studied and experimented mostly with animals. Dissections of human bodies were typically not performed during his time, however he carried out detailed studies on apes. His work remained unchallenged until the great anatomists, such as Andreas Vesalius, in the 16th century.

Hobbes describes human cognition as the manipulation of symbols, words or “generall names” with the aid of logical operators, which themselves are based on primitive functions such as “*addition* and *subtraction*”. This concept is part of the current debate in artificial intelligence and in the philosophy of mind in general, albeit in a more formalized form. The possibility of mechanizing and automating the execution of such primitive functions with the aid of computing machinery led to the physical symbol-processing hypothesis and the beginnings of AI in the 1940s. However, a possible connection between automated computing devices, thought, and creative behaviour was entertained earlier. During the first half of the nineteenth century Lady Ada Lovelace, who is regarded as the first computer programmer, worked with Charles Babbage on the *Analytical Engine*. This mechanical device was never completed for organizational and financial reasons. Fortunately, many of the plans and descriptions and even some components of Babbage’s engines have survived and their design and computing power has since been investigated and is well understood [2]. Alan Turing remarked that

> although Babbage had all the essential ideas, his machine was not at the time a very attractive prospect. The speed which would have been available would be definitely faster than a human computer … (Turing 1950, 439)

The machine was designed to be a mathematician’s tool and the idea of intelligence being “produced” by the analytical engine had not been anticipated. Nevertheless, Lovelace speculated on the computational powers of the machine and she envisaged that the machine might been able to play chess and compose music (Kurzweil 1990, 167).

The research into the possibilities for the creation of intelligent systems began with the conception of the theoretical and technological foundations of modern computing machines itself. The combination of an emerging computational theory, breakthroughs in the engineering of digital computers, and a

[2] The late Allan Bromley worked and published on Babbage’s machines for many years. A good introduction into Babbage’s engines by Allan Bromley and early computing machines in general can be found in Aspray (1990).

mechanistic theory of mind led to a computational theory of mind. The computational theory of mind has a strong philosophical foundation through the work of Putnam on functionalism (Putnam 1990) and the representational theory of mind by Fodor. Pylyshyn (Pylyshyn 1984) has presented a widely accepted form of the computational theory of mind, which is accepted by many and forms part of the philosophical and ideological foundation for the discipline of cognitive science. Some aspects of the current computing paradigm seem to influence the theory of mind nevertheless. Computing theory, which forms the basis for computer science, is sometimes applied directly to cognitive science. It is this naïve form of computationalism, which Dreyfus has criticized (Dreyfus 1992). There are more moderate positions where only certain attributes of *formal* computing are deemed to be of relevance in the discussion about intelligence, artificial and otherwise. The application of Gödel's incompleteness theorem by Lucas (Lucas 1964, Lucas 1970) and Penrose (Penrose 1990) is one example, where certain aspects and consequences of a mechanistic philosophy of mind are used as arguments against an artificial intelligence (Lucas's argument is explored in chapter three).

The computational theory of mind is obviously not a clearly defined or homogenous field of inquiry. Harnish points out that the computational theory of mind is essentially a special case of a representational theory of mind which goes back to the empiricists John Locke and David Hume (Harnish 2002, p 105). Locke (1632-1704) argued against innate ideas and knowledge and he held the view that all our knowledge comes from experiences, namely sensations and perceptions. Hume (1711-1776) spoke of ideas impressed on the mind as faint images and explained some phenomena of cognitive processes such as vagueness and memory in his theory (Russel 1946).

The field of artificial intelligence, as we know it, started with the work of McCulloch, Pitts, Minsky, Newel**l**, Simon and McCarthy, amongst others, as far back as the 1940s [3]. Newell and Simon (Newell & Simon 1997, Newell 1990) proposed the physical symbol processing hypothesis, which forms the theoretical basis for “classic” AI, while McCulloch, Pitts and Rosenblatt (Rosenblatt 1958) are credited with building the first computational models of neurons, which were based on knowledge of basic physiology and functionality of the brain. Both concepts were greatly influenced by formal propositional logic and, of course, by Turing’s computational theories (Russell & Norvig 1995, 16). The first major conference, a workshop in Dartmouth in 1956, had Newell, Simon, Minsky and McCarthy attending. Russel and Norvig comment that

> for the next 20 years, the field would be dominated by these people and their students and colleagues at MIT, CMU, Stanford, and IBM. Perhaps the most lasting thing to come out of the workshop was an agreement to adopt McCarthy’s new name for the field: artificial intelligence. (Russell & Norvig 1995, 17)

Since the beginning of modern computing two schools of thought have dominated the field of AI. On the one hand there is the *connectionist* approach to AI, which is loosely based on structures and primitive elements adapted from biological entities, namely neurons. Currently, networks of neuron-like structures are modeled by simulating their behaviour and the interactions between them on computers. It is hoped that looking at the structures in the brain, and by modeling these structures in computer systems will provide new insights into the working of our minds and will provide a basis for the engineering of intelligent systems. On the other hand, there is an approach to AI, which aims to build systems that behave intelligently with little reference to brain architecture. These symbol-manipulating systems use generally classic processing methodologies and have been referred to as *serial*, or *digital* processing systems. Neither of these terms in my opinion adequately describes the fundamental architecture of these systems nor do they adequately describe the fundamental differences from the connectionist

[3] The list of names that should be compiled just to give credit to the major players would be very long indeed. The people I have named here have contributed to the field of artificial intelligence in general and some of their work is directly pertinent to this thesis and will return these particular contributions later in this work.

approach. I will use John Haugeland's term *good old fashioned AI* (GOFAI) to refer to such systems (Haugeland 1997). GOFAI includes among the many developments, expert systems, which comprise substantial amounts of knowledge in the form of propositions stored in databases and inference engines to deduce new knowledge. Both approaches to AI have co-existed since the inception and development of modern computing machinery. Since the 1940s the emphasis has shifted several times from one approach to the other, depending on the emergence of promising results or failures respectively. After Minsky and Papert published their work *Perceptrons* (Minsky & Papert 1969), which exposed rigorously the limitations of single layer neural nets, research into neural nets was almost abandoned for some twenty years and during this time most efforts were directed into GOFAI techniques. Since the 1980s, research has again been dominated by work in neural nets and efforts into GOFAI have declined, with Lenat's CYC [4] project probably the most notable exception.

The GOFAI versus connectionism issue is only one of the ongoing debates in AI. Even the aims and goals of AI are not clearly stated and suggestions for definitions are many. The construction of intelligent machines and the exploration of the workings of our own intelligence seem to be the aims on which most practitioners agree (Schank 1990). AI is also an engineering enterprise and an empirical endeavour. The engineering branch is concerned with the design and construction of computer systems that behave intelligently, or exhibit at least some intelligent behaviour. The question of what can be accepted as an "intelligent" system is a difficult one indeed. Weizenbaum's ELIZA (1966) was a program written to demonstrate that it is possible to create something behaving as seemingly "intelligent" using relatively simple techniques. ELIZA takes input from a user and responds like a psychoanalyst, seemingly "dealing" with the user's problem. Sample dialogues with the program can be found in Kurzweil (Kurzweil 1990) and

[4] Lenat's CYC is a large-scale expert system project that started in the 1980s. Although the program is still active in 2002, no real progress has been made. The idea behind CYC is to establish a database containing much of all the knowledge there is, including most of the contextual rules.

Hofstadter (Hofstadter 1980) among others. Weizenbaum himself remarked about the apparent success of ELIZA that

> [he] had not realized that extremely short exposures to a relatively simple computer program could induce powerful delusional thinking in quite normal people. … This reaction to ELIZA showed me more vividly than anything I had seen hitherto the enormously exaggerated attributions an even well-educated audience is capable of making, even strives to make, to a technology it does not understand. (Weizenbaum 1976, 7)

ELIZA showed that intelligence is difficult to define and difficult to detect - some people are more easily misled than others. Turing suggested measuring intelligence in terms of performance, i.e. how "intelligently" a system behaves (Turing 1950). Whether the so-called *Turing Test* is a suitable means to decide whether systems are intelligent or not, is still part of the debate in AI. Expectations about the performance of intelligent systems change over time. Terry Winograd's *SHRDLU* (1971) was for a long time considered by some people to be a good example of an early success in AI. Haugeland, for example, wrote in 1986 that

> The best known and most impressive block-world program is Terry Winograd's (1971) simulated robot SHRDLU, …[it] can carry on surprisingly fluent conversations … Moreover, SHRDLU is not all talk … he will tirelessly comply, right before our wondering eyes. (Haugeland 1986, 186)

Today, Winograd's *SHRDLU* is universally classed as being rather trivial in terms of exhibiting any "intelligent" behaviour. It has been said about AI that the goal posts are shifted whenever something from the list of "but you can't do this – items", *has* been done.

# Chapter 2

# What is Computation?

In this chapter I will establish the origins of the mathematico-logical definition of computation. In the second part I will outline the connection between the Church-Turing thesis and the concept of computing. The term *computation* is used throughout the literature and its meaning changes largely depending on the context in which the term is used. Computer science people generally mean by computation the execution of a program, unless they refer to computation in terms of computational theory or complexity theory. The latter may also be the concept of computation that mathematicians might refer to. Lay people seem to associate computation with number crunching and rocket science, while psychologists and philosophers seem to include a lot more. A few thinkers even attribute computational powers to walls and rocks; for others computing is synonymous with game playing. Even the players in the field of cognitive science attach to the term computing much more than there should be. Harnad points out that

> the fathers of modern computational theory (Church, Turing, Gödel, Post, von Neumann) were mathematicians and logicians. They did not mistake themselves for psychologists. It required several more decades for their successors to begin confusing computation with cognition … (Harnad 1995, 379)

While Harnad considers the relationship between computing and cognition might be one of confusion, Pylyshyn argues that "cognition *is* a type of computation" (Pylyshyn 1984, xiii).

During the second half of the nineteenth century several mathematicians and logicians worked on the foundations of mathematics. The underlying question for most of this work was about the consistency of mathematics itself. Boyer refers to this period as the nineteenth-century age of rigor (Boyer 1986, 611). The mathematician David Hilbert raised a list of seventeen unsolved problems in mathematics in 1900 and later in a more specified form in 1928. Among these questions, two were – and still are - of particular interest to computation and arguably related to theories of cognition. Hilbert asked the question whether it is possible to produce a proof that a sequence of steps or transformations, which are based on a set of axioms, can never lead to a contradictory result. The second problem was concerning the question of whether there is a definite method to show that some mathematical procedure will yield a result. Essentially, the question was about the existence of a general algorithm to determine whether a particular problem has a solution. In response to the first problem, the work of enormous proportions by Russell and Whitehead, the *Principia Mathematica* (1910-1913) was intended to prove that arithmetic and all of pure mathematics could be derived from a small set of axioms. Russell and Whitehead wanted to show that mathematics is essentially indistinguishable from logic (Boyer 1986, 611). However, by 1931 Gödel concluded that the system of arithmetic, which Russell and Whitehead had proposed, was not complete. While Gödel did not show that an axiomatic system leads necessarily to contradictory statements, he was able to conclusively prove that any such system contains propositions that are undecidable within the system. Gödel's *Incompleteness Theorem* does not state that arithmetic is proved to be inconsistent, but that arithmetic is certainly not consistent *and* complete (Hodges 1992, 93).

During the search for a solution to Hilbert's question about the *decidability* of mathematics, Alan Turing devised a theoretical computing device with strongly mechanistic, or machine-like, principles of operation. With the help of this device, now commonly referred to as a Turing Machine, Turing was able to show that there are numbers that cannot be computed by means of an algorithm or effective procedure [5]. Using Cantor's diagonalization argument and Gödel's result on self-referential statements (Gödel's Incompleteness Theorem), Turing could demonstrate that algorithmically unsolvable problems

[5] The term *computable* does not refer to whether a number has an infinite number of digits. The decimal expansion of the square root of two has an infinite number of digits, however there is a definite and finite description (algorithm) how to calculate each and every one of the

do in fact exist. He was able to show that there cannot be a definite method to check whether an algorithmic solution for a particular problem exists. Turing came to this conclusion because he could prove that there is no logical calculating machine that can determine whether another will ever come to a halt – i.e. complete a calculation – or will in fact continue forever (the *Halting Problem*). The overall result for mathematics and computational theory is that there is no algorithm or effective procedure to determine whether there is an algorithmic procedure for some problem. There is a list of postulates and theorems dealing with computability, which are closely related to this insight. In essence the Church thesis, Turing thesis and Church-Turing thesis are stating the same core result. There are of course, significant differences between these, and for certain arguments the details and the associated implications become relevant.

Not everyone accepts a purely mathematico-logical definition of computation in terms of Turing machines as sufficient or adequate to be usable in cognitive science or artificial intelligence. Sloman (Sloman 1996) asks whether analog computing and forms of neural computation may have to be included in a workable definition of computation (see also Haugeland 1981, Haugeland 1998). I will return to the question of equivalence of Turing machines and neural nets in chapter six. There are other ways to describe and define concepts of computation, and I will outline some alternatives. Because I argue in this thesis that the concept of computation based on Turing machines is too narrow, some of the alternative views must be considered.

Chalmers offers a computational concept that includes Turing machines, neural nets, cellular automata [6] and Pascal programs (Chalmers 2001). However his definition has fewer basic assumptions and components than Turing machines. Chalmers accepts that every physical system is an implementation of a simple finite state automaton and that every system

digits. Moreover, any arbitrary number of digits can be determined, in principle, in finite time using finite amounts of tape (memory) by a specific Turing machine.

implements computation. This claim is essentially a dangerous one, because it would follow that a rock is really a single state automaton and therefore is also a "computer". The question here is about how trivial computation can be, and can we still refer to it as computation. A single state automaton, which by virtue of its simplicity cannot accept input or produce output, is trivial indeed.

Harnad suggests that semantic interpretability is a necessary criterion of computational systems (Harnad 1995). He holds that a formal definition of computation in terms of Turing machines does not rely on the semantics of symbols that are processed. Real and useful computation, however, has to make systematic sense.

> It is easy to pick a bunch of arbitrary symbols and to formulate arbitrary yet systematic syntactic rules for manipulating them, but this does not guarantee that there will be any way to interpret it all so as to make sense. (Harnad 1995, 381)

I think that the definition of computation must include even more than the coherent and "sense-making" semantic interpretability suggested by Harnad. Symbol manipulation can only be regarded as computation, if these symbols *are* semantically interpreted. Any computation with the aid of a machine, an electronic calculator or a personal computer, is a very complex process at the binary level. Because of that, semantic interpretation at this level for most users of the machine is neither possible nor necessary. The engineer, however, is able to determine that the bit pattern at some memory location corresponds to a digit, if the output of a certain flip-flop is zero and some other conditions are met. The semantic interpretation of the inputs and outputs by the human using the machine are required, that is, there must be some intentionality and intent in relation to computation. Pressing buttons on an electronic calculator is no different to pressing buttons on a remote control unit for a television, even if for some reason the sequence of button presses on the calculator was syntactically correct and produced a result.

[6] Chalmers defines a *cellular automaton* as a superset of a finite state automaton, which acts on vectors of symbols as input rather than single symbols. They are Turing machine equivalent.

Analogously, sliding beads on an abacus does not constitute calculation, unless the beads are "place-holders" for numbers, i.e. the beads must have semantic interpretations attached to them. Values on the abacus are encoded in the patterns of beads [7]. The manipulations of the beads, or symbols in general, have to follow the formal rules to maintain these semantics. Computation is an intentional, i.e. purposeful, process in which semantically interpreted symbols are manipulated. Semantic interpretation as a requirement for computation provides a method to eliminate some instances of trivial and pathological forms of computation. McDermott (McDermott 2001) includes the thermostat and the solar system in a list of examples what he considers to be computers. Moreover, he suggest that

> the planets compute their positions and lots of other functions … The Earth, the Sun, and Jupiter compute only approximately, because the other planets add errors to result, and, more fundamentally, because there are inherent limits to measurements of position and alignment. If Jupiter moved 1 centimeter, would it still be aligned? (McDermott 2001, 174)

I disagree with McDermott's notion of a *computing* universe. The solar system is not computing anything - nor is the solar system measuring any positions as inputs. McDermott's solar system supposedly computes a function using "distances $r_1$ and $r_2$, and velocities $v_1$ and $v_2$" (McDermott 2001, 175). We do not have any evidence that the solar system is not computing epicycles rather than using Kepler's formula. Dreyfus is also critical of a computing solar system and he comments on the "computation by planets".

> Consider the planets. They are not solving differential equations as they swing around the sun. They are not *following* any rules at all; but their behavior is nonetheless lawful, and to understand their behavior we find a formalism – in this case differential equations – which expresses their behavior *according to* a rule. (Dreyfus 1992, 189)

The questions about trivial or incidental computation are much more serious than that. By "incidental", I mean the type of computation that happens without any purpose. Searle's "wall implementing word-star" (Searle 1990b) and

[7] Two beads side by side on the same wire can represent the numbers two or six on some

Putnam's "rock implementing every finite state automaton" (Putnam 1988) deal with the problem of computational functionalism. Searle concludes that physical systems are not intrinsically computational systems. He notes that

Computational states are not *discovered within* the physics, they are *assigned* to the physics. … There is no way you could discover that something is intrinsically a digital computer because the characterization of it as a digital computer is always relative to an observer who assigns a syntactical interpretation to the purely physical features of the system. (Searle 1990b)

Searle argues against the view that physical systems are "engines", neither semantic engines nor syntactic engines as, for example, Haugeland does. (Haugeland 1986).

McDermott rejects Searle's arguments against a functional view of computing and offers a more concrete definition (McDermott 2001). For McDermott, a computer is purely a syntactic engine. The concept of computing includes a much wider range of systems, and it is therefore much broader than a concept based on digital computers. He suggests as a definition, that a computer is a physical system having certain states. The states of such systems may not be necessarily discrete and can also be partial, that is, such a system can be in several states at once (McDermott 2001, 169). Another important feature of such a computing system is that its outputs are a *function* of its inputs. Interestingly, the notion of function, as McDermott proposes, could be taken from a textbook in mathematics:

A computer computes a function from a domain to a range. Consider a simple amplifier that take an input voltage *v* in the domain [-1, 1] and puts out an output voltage of value 2*v* in the range [-2, 2]. Intuitively, it computes the function 2*v* … (McDermott 2001, 173).

McDermott's admission of states that may be non-discrete, i.e. they can be continuous or may be partial, has some important consequences for his concept of computation. Firstly, there is a possible contradiction in that

types of abaci. The meaning (value) of beads depends on their position on the abacus.

systems cannot have states that are not discrete. The term *state*, I would think, presupposes that there is an amount of stability at a certain level in a system when it is considered to be in a state. Stability does not mean that the entire system is without motion. A washing machine can be in several states, switched off, washing, spin-drying, and so on. At the level of these descriptive states the machine can only be in one state at any time, with transitions in between - the machine cannot wash and spin-drying concurrently. However, at a lower level, the machine can be dynamic and may exhibit no stability at all – e.g. the machine's motor is turning and the timer is winding down and so on. The motor of the washing machine itself can be in many states: not turning, turning slowly forwards, turning fast, and so on. There is a large number of ways defining *states* for objects, or attributing *states* to objects. Looking at the turning motor, it becomes meaningless to define *states* to the various degrees of turn, say, without discretizing the process of turning, because the turning of the motor is a continuous process. An engineer may well say that something or other should happen once the motor turns through 180 degrees, but then the process is no longer a continuous one. The introduction of a fixed marker or a fixed event – 180 degrees – allows for the introduction of higher level states: not yet at 180 degrees, at 180 degrees and triggering some event, past 180 degrees, and so on.

Secondly, if a system is allowed to be in states, there have to be guards that such states are not mutually exclusive. McDermott says that a system "might be in the state of being at 30 degrees Celsius and also in the state of being colored blue" (McDermott 2001, 169). The problem here is that McDermott identifies states at a semantic level and not at syntactic level. I would argue that his concept of *state* is much more a concept of *properties*. Consider again the simple amplifier, which implement the function 2*v*, which is continuous. The function definition, however, narrows the possible inputs for this function to numbers. *Blueness* cannot and must not be allowed in the domain of a function 2*v*. Accepting that a computer computes a function in terms of mapping from a domain to a range implies that such a computation does occur at a syntactic level and must remain semantically interpretable during the process.

The story so far establishes the background for a definition of computation that is largely in aid of solving a particular mathematical problem. In the following chapters, I will investigate some aspects of the Church-Turing thesis, which is regarded as one of the most important theorems for computer science, artificial intelligence and of course for a computational theory of mind.

## The Church – Turing thesis

Throughout the literature there are many definitions and descriptions of the Church-Turing thesis and much of what Turing said seems to be either ignored or re-interpreted as Copeland and Sylvan suggest (Copeland & Sylvan 1999). There are many opinions and inferences about the implications of the Church-Turing thesis regarding possible limitations of computing machines and especially what it may mean to AI and cognitive science. Copeland and Sylvan list several definitions, which they find either not correct or "biased in favour of a subset of digital and digitally emulable procedures" (Copeland & Sylvan 1999). Before investigating whether this view can be justified, I will attempt to state the Church-Turing thesis according to Turing. The use of Turing's work for a definition of the Church-Turing thesis, rather than Church's, seems to be the preferred option in the literature. Church established independently from Turing the connection between the computability of functions and the ability to express computable functions in a particular mathematical form, his λ-calculus. Church's results show that any computable (i.e. algorithmic) function can be transformed into an expression in the λ-calculus[8]. Since then, it has been shown that Church's results are equivalent to Turing's description that any algorithmic function can be computed by a Turing machine [9]. Hence the name *Church-Turing* thesis (Sipser 1997).

Turing refers to computing machines in terms of procedures executed by humans in his 1937 paper *On computable numbers, with an application to the Entscheidungsproblem*:

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions… (Turing 1937, 117)

[8] Penrose gives a concise introduction to Church's λ-calculus and a sketch of Church's proof in *The Emperors New Mind* (Penrose 1990, 86-92).

[9] In this work I refer to "Turing" machines even in a historical context. This should not be construed as an anachronism. Turing called his machines *LCMs* and *theoretical machines* - he did not refer to them as *Turing* machines.

Turing relates two terms, which need clarification regarding their meaning before the age of digital electronic computers and their meaning now. In Turing's time, a *computer* was a person who solves mathematical problems using numbers and little more than pen and paper. Turing does not refer to machines when he uses the term *computer*. The idea of a *machine* includes also his theoretical "logical calculating machine", i.e. what we now call the Turing machine. Turing gives a detailed description of what his understanding and definitions of *machines* are (see Turing 1948). The "Logical Computing Machine" – i.e. Turing machine – and the "Practical Computing Machine" – i.e. electronic digital computer – are the machines of concern as far as the Church-Turing thesis is concerned. I will argue, that in fact only the Turing machine and the special kind of Turing machine, known as a Universal Turing machine, are directly related to the Church-Turing thesis.

Other than the quotation above, we find several references in Turing's work, which can help us to determine what the Church-Turing thesis entails:

It is found in practice that LCMS[10] can do anything that could be described as 'rule of thumb' or 'purely mechanical'. This is sufficiently well established that it is agreed amongst logicians that 'calculable by means of an LCM' is the correct accurate rendering of such phrases. (Turing 1948, 111)

and

It is possible to produce the effect of a computing machine by writing down a set of rules of procedure and asking a man to carry them out. Such a combination of a man with written instructions will be called a 'Paper Machine'. A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine. (Turing 1948, 113)

Copeland and Sylvan argue that Turing's answer to Hilbert's question "concerns the limits only of what a human being can compute, and carries no implication concerning the limits of machine computation" (Copeland & Sylvan

[10] Logical Computing Machines

1999). Copeland and Sylvan offer this as a definition of the Church-Turing thesis proper:

Any procedure that can be carried out by an idealised human clerk working mechanically with paper and pencil can also be carried out by a Turing machine. (Copeland & Sylvan 1999)

This is the Church-Turing thesis: An idealised machine can also follow any procedure that can be followed mechanically by a human. “Any procedure” can only mean the application of an algorithm for computation with numbers. After all, the 1937 paper was specifically written in response to the problems from Hilbert’s program. Turing referred here to a “disciplined” human, which clearly indicates to me that the Church-Turing thesis is concerning the possibility of automating “mindless” human computations. It is the mechanical, automatic principle of algorithms that matters.

The Church-Turing thesis says nothing about the relationship between Turing machines and digital computers. Although Turing remarks elsewhere that Turing machines can be implemented or emulated on digital computers. He states that

in practice, given any job which could have been done on an LCM one can also do it on one of these digital computers. (Turing 1948, 112)

This remark does not imply that a digital computer cannot do other things besides doing what a LCM, i.e. Turing machine, can do. In terms of computing ability it can be said that, whatever LCMs can compute is a subset of the things digital machines can compute. The converse, that what can be done on “one of these digital computers” can also be done on a Turing machines, does not follow. However, the assumption that electronic computers are somehow equivalent to Turing machines is widely held. Copeland and Sylvan give several examples of what are, in their opinion, renditions of the “so-called” Church-Turing thesis (Copeland & Sylvan 1999):

Since all computers operate entirely on algorithms, the … limits of Turing machines … also describe the theoretical limits of all computers. (McArthur quoted in Copeland & Sylvan 1999)

And

[any] problem for which we can find an algorithm that can be programmed in some computer language, *any* language, running on some computer, *any* computer, even one that has not been built yet but *can* be built, and even one that will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine. (Harel quoted in Copeland & Sylvan 1999)

Let me add two more examples. Putnam writes

It should be remarked that Turing machines are able in principle to do anything that any computing machine (of whichever kind) can do. (Putnam 1975, 366)

At the end of this sentence we find a footnote in the original text in which Putnam reinforces his claim:

This statement is a form of *Church's thesis.* (Putnam 1975, 366)

Ray Kurzweil, prolific writer and AI "guru", makes the following claim about Turing machines and the Church-Turing thesis:

The Turing machine has persisted as our primary theoretical model of computation because of its combination of simplicity and power … As for its power, Turing was able to show that this extremely simple machine can compute anything that any machine can compute, no matter how complex. If a problem cannot be solved by a Turing machine, then it cannot be solved by any machine (and according to the Church-Turing thesis, not by a human being either). (Kurzweil 1990,112)

Kurzweil's assumptions about what the Church-Turing thesis states do not stop there. He reiterates that, according to Turing,

if a problem is presented to a Turing machine is not solvable by one, then it is not solvable by human thought. (Kurzweil 1990, 177) and that

in the strongest formulation, the Church-Turing thesis addresses issues of determinism and free will. Free will, which we can consider to be purposeful activity that is neither determined nor random, would appear to contradict the Church-Turing thesis. Nonetheless, the truth of the thesis is ultimately a matter of personal belief … (Kurzweil 1990, 117)

I disagree with Kurzweil on the "computer-human-Turing machine equivalence" hypothesis. Specifically, Kurzweil's attempt to connect "free will" with the Church-Turing thesis must also be rejected, because Turing made it quite clear what the relationship between "free will" and the Church –Turing thesis is:

A man provided with paper, pencil, and rubber, and *subject to strict discipline*, is in effect a universal machine. (Turing 1948, 113, italics added)

"Subject to strict discipline" in this sentence can be translated into an even stronger form: *Do not think – just blindly follow the algorithm*. Free will and any restrictions on what humans can do, think, or believe are not part of the Church-Turing thesis. I argue that free will is anathema to the Church-Turing thesis, because free will in relation to the *proper* Church-Turing thesis would suggest a possibility of deviation from the algorithm that the human computer is executing. Anything other than strictly following that algorithm is against Turing's rules and regulations. Human beings must not have free will, when they want to emulate a Turing machine. Kurzweil's "free will" claim relies on his assumption that the Church-Turing thesis says something about the limits of human intelligence. However, there is no evidence that it does. Kurzweil's statement:

If a problem cannot be solved by a Turing machine, then it cannot be solved by any machine (and according to the Church-Turing thesis, not by a human being either). (Kurzweil 1990,112)

has no basis.

In its "proper" form, the Church-Turing thesis describes a relationship between the execution of an algorithm by a human and the possibility of the execution of that algorithm on a Turing machine. The Church-Turing thesis only establishes that some mathematical problems, which have solutions that can be found by the application of suitable algorithms, can be implemented on Turing machines. We may accept as an extension to Turing's own claim that Turing machines can in practice be implemented on digital machines. This establishes clearly the relationship between algorithm and physical computing machinery, namely von Neumann machines. The Church-Turing thesis makes no claims about what a digital machine can do *besides* emulating Turing machines. Copeland and Sylvan comment that

This thesis (the Church-Turing thesis properly so called) … carries no implication concerning the limits of machine computation. Yet the myth has somehow arisen that in his paper of 1936 Turing discussed, and established important results concerning, the theoretical limits of what can be computed by machine. (Copeland & Sylvan 1999)

The Church-Turing thesis seems to have undergone a process of re-definition over the last fifty years or so. Moreover, the Church-Turing thesis has been cited to give credence to countless claims about what computers can or cannot do. The interpretation of the Church-Turing thesis has even been stretched to make conjectures about the limits of human intellectual abilities.

The mathematical concept of computing in terms of Turing machines is the only well-defined concept of computation (Sloman 1996). The mathematical concept is about formal structures, which do not require physical representation. The essential part of Gödel's proof is the possibility to express an entire computational system as a series of numbers. In fact, a computational system can be expressed as a single, probably large, Gödel number. "Thus a number can satisfy the formal conditions for being computation" (Sloman 1996, 180). Sloman points out that computation, which is definable as a sequence of numbers, is unsuitable to "build useful engines or explain human or animal behavior" (Sloman 1996, 180). The question is whether computation has causal powers. Sloman believes that

an abstract instance of computation (e.g. a huge Gödel number) cannot make anything happen. This shows that being computation in the formal sense is not a *sufficient* condition for being an intelligent *behaving* system, even though the formal theory provides a useful conceptual framework for categorizing some behaving systems. Sloman 1996, 181)

If the abstract notion of computation is not sufficient for a system to behave intelligently, then what else is necessary? Sloman suggests that the solution is the combination of computational ideas and some machine, which has the causal powers. The argument here is that Turing machines by themselves, although they may give us, by definition, a clear understanding of what computation is, are in fact inconsequential for artificial intelligence. According to Sloman, a formal definition of computation in terms Turing machines is

inadequate for the purpose of identifying the central feature of intelligence, because satisfying it is neither sufficient nor necessary to be intelligent.
(a) It is not sufficient because the mere fact that something is implemented on or is equivalent to a Turing machine does not make it intelligent, does not give it beliefs, desires, percepts, etc. Moreover, a computation in this sense can be a purely static, formal structure, which does nothing.
(b) Turing-machine power is not necessary for such aspects of intelligence as the ability to perceive, act, learn,…, because there is no evidence that animals that have these abilities also have Turing equivalent computational abilities…(Sloman 1996, 190)

AI is concerned with intelligent systems and only physical machines can have causal powers. In machines with causal powers that are relevant to artificial intelligence, formal computation (i.e. Turing machines) may or may not be involved in such processes. Sloman argues, correctly I think, that *some* computation (a program) must be controlling *some* physical system (a computer) that has causal powers to interact with the world. The formal aspect of computation is of little concern here.

We can agree that a machine, i.e. computer, can interact with world through screen, keyboard, speakers and sensors of all kinds. This interaction between program, machine and world needs to be discussed further. The program or "computational idea" must be in control of the machine, in order to have effects on the physical machine. We would expect that the programs have some causal power to alter the physical states of machines. We can view the program to be equivalent to the algorithm it actually implements. However, this program, i.e. algorithm, can be described as a single Gödelian number like any other formal computation. This Gödelian number has certainly no causal power in relation to some physical machine: a number can not have causal powers over a physical system. However, the steps of some computation, some algorithm, can be transcribed into a physical form that is machine-readable. A program sitting in memory, ready for execution, is a series of patterns, which are the computational ideas in an encoded form. These patterns are physically realized as the states of a series of flip-flops, as charges in a series of tiny capacitors or as the holes in a pack of punched cards. The program or algorithm in this form is a collection of physical symbols.

The machine has causal powers to change its own states – the machine is behaving automatically. The changes of the states in the machine occur according to the various patterns, which are in fact the program. Sloman claims that the combination of bit patterns (the formal computational structure) and the design, which provides the necessary causal links between bit patterns and physical state changes in the machine, "provides a foundation for meaning" (Sloman 1996, 192). He says that at this level (executing machine code) the machine actually

*understands*, in a limited fashion, instructions and addresses composed of bit patterns. While obeying instructions, it manipulates other bit-patterns that it need not to understand at all, although it can compare them, copy them, change their components, etc. This limited, primitive understanding provides a basis on which to implement more complex and indirect semantic capabilities, including much of AI. (Sloman 1996, 192)

Essentially, Sloman grants the machine a degree of autonomy and understanding at the lowest level of operation. Not everyone agrees.

Kearns claims that it is necessary to act intentionally to follow a procedure, "*in order* to achieve a purpose, *trying* to get things *right*" (Kearns 1997, 280). He also argues that the Church-Turing thesis has two distinct forms. Kearns refers to the Church-Turing thesis "proper" as *Turing's procedural thesis* and uses the term *Turing's mechanical thesis* to express the view that there exists a mechanical device that would produce the same outputs as the algorithm, if it was executed by a human. He says that

for every effective mark manipulation procedure, we can, in principle, *build a physical device* which, by causal processes, produce the outputs that we would get (in principle again) by carrying out the procedure. (Kearns 1997, 279, italics added)

From here it can be argued, as Kearns does, that machines are built as an aid to our own actions. Kearns views Turing machines as "*either* … a mark manipulation procedure for someone to carry out or … as a device which operates independently of us" (Kearns 1997, 274). In doing that Kearns provides an elegant solution for the connection of "*intentionality*, in the *on purpose* sense" (Ibid.) and pure procedure. He suggests, that a person who follows an algorithm is acting "purposively" and as a person

must understand the rules, and intend to implement them. She *intentionally* tries to get the right result. (Kearns 1997, 275)

Kearns offers a second interpretation of Turing machines as "spatiotemporal engines", in which he asks the question, whether a machine computes at all. He notes that "no one actually builds Turing machines, because they would be too tiresome to work with" (Kearns 1997, 275) and then he raises a most interesting point when he claims that

in a physically realized Turing machine, physical tokens of marks will be causally effective in determining just how the device "maneuvers" and what strings the device produces The physical Turing machine does not follow rules, for causal processes are neither purposive nor intentional. (Kearns 1997, 275)

Kearns essentially argues here that Turing machines do not compute. They merely should be seen as extensions of our enterprise. Machines do not carry out any procedures, because they do not act intentionally. That fact that these machines produce the correct answers is the consequence of the correctness of the procedures. If computers do not carry out procedures, then computation would only be possible in a human-machine system, or for a human without the aid of any machinery, of course. The idea that computers by themselves are not performing any computation and should be viewed only as mere physical objects has also been suggested by Searle and Putnam (McDermott 2001). Clark (Clark 2001) holds the view that computation is a functional concept, i.e. the computation is only computation if it used for that purpose. As far as the computational properties of machines or physical devices are concerned, he maintains that computers are the result of a "deliberate imposition of a mapping, via some process of intelligent design" (Clark 2001,18). Churchland and Sejnowski (Churchland & Sejnowski 1992) also propose this functional point of view and say that:

We count something as a computer because, and only when, its inputs and outputs can *usefully* and systematically be interpreted as representing the ordered pairs of some function that interests us. Thus there are two components to this criterion: (1) the objective matter of what function(s) describe the behavior of the system, and (2) *the subjective and practical matter of whether we care what the function is*. (Churchland & Sejnowski 1992, 65)

Sloman (Sloman 1996, 185) illustrates this same idea with the example of a rock with a "table-like structure". Unless someone uses the structure as a table, it remains a rock. A slide-rule for example remains a collection of uninteresting pieces of wood or plastic with scratches on them, until someone uses them to calculate. Then, the slide-rule becomes a calculating tool. But can a slide rule be called a computer? Yes and no.

Yes, because a slide-rule is a physical artifact to help humans calculate things. It can be argued that the marks on the slide-rule are representations or symbols for points on number lines, i.e. representations or symbols for *numbers*, and the manipulation of segments on the number lines (defined by the marks) is under some sort of algorithmic procedure. There are in fact clearly specified algorithms for various operations on slide-rules describing what number to set on what scale and where to read of results after a chain of operations. It would be more difficult to argue that a human without a slide-rule could execute the same operations as effective procedures. The Church-Turing thesis requires that a human can do the calculation in the first instance by following some procedure. It is conceivable that one could have a human being doing calculation with numbers using geometrical methods. Addition, multiplication and the like can be done with a pair of compasses, straight edge and a suitably accurate representation of number lines of various scales.
On the contrary, the slide rule does not exhibit the necessary properties to make it equivalent to a logical calculating machine. For example, a slide rule has no discrete states and is not mechanical in the sense that it is not able to do things automatically. A slide rule is certainly not a computer in the digital machine sense, yet it is clearly an aid for a human to calculate. Abaci, slide rules and other calculating *tools* are usually operated by applying algorithms and these operations can, *in principle* I believe, be executed by a human with pen and paper. It seems that a definition of computation has to be broad enough to include a wide range of computational (calculating) processes.

It is difficult to formulate a definition of computation that includes the formal aspects of Turing machines, which are important for the field of computer science, if we want to make this definition broad enough so that other suggested forms of computation can be accommodated. It has been argued that the Church-Turing thesis in its “proper” form places no restrictions on what artificial intelligence can do. The Church-Turing thesis is closely linked to the concept of the Turing machines, and the Turing machines and their properties are the subject of the next chapter.