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, Newell, 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 $2v$ in the range [-2, 2]. Intuitively, it computes the function $2v$ … (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 $2v$, 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 $2v$. 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.

Chapter 3

Turing Machines.

In order to discuss the perceived theoretical limits of electronic computers in general, the concepts and theoretical limits of Turing machines need to be explored first. The properties of Turing machines and the notion of an algorithm are obviously closely related through the Church-Turing thesis. My intention is to show what these properties of Turing machines are and that some of these properties are not applicable to digital computers. A Turing machine is an entirely theoretical entity, which can be looked upon as the mathematical foundation for the development for almost all digital computers, as we know them today. The vast majority of computing devices today are based on a common architecture and are commonly referred to as von Neumann machines. Von Neumann machines are register based machines and the symbols - data and program code - are accessed in memory by using the addresses of their storage location. They are essentially serial or sequential machines, in that one command is executed after the other. Each command in itself is a short sequence of discrete events, known as the fetch-execute-store cycle. Modern hardware is much more sophisticated and some of the operating “principles” of von Neumann machines are really not always applicable in a strict sense. For example, modern processors use pipelining techniques where the execution of one instruction is partially overlapping the execution of the previous and the next instruction. At the hardware level, we can find many semi-autonomous devices that interact with the main processor by means of interrupts. These devices can access the main memory and the machine effectively does some parallel processing.

While it is clear that Turing machines do not exists as real machines we can assume a well defined architecture for these theoretical entities nevertheless. Turing described the architecture and the workings of this elementary “machine” in detail (Turing 1937). There are some similarities and some

distinct differences in operation [11], when we compare von Neumann machines with Turing machines. Firstly, programs in von Neumann machines are typically stored in memory, analogous to a Universal Turing machine, and access to memory can be random (direct) and relative, whereas in a Turing machine only relative, i.e. only sequential, access is possible. Von Neumann machines are much more efficient in operation, because the architecture allows for program structures such as sub-routines and the like, even in situations where they merely emulate a particular Turing machine [12]. Von Neumann machines are also easier to program due to the higher level programming languages and the appropriate compilers. Modern programming languages allow for computers to be programmed in a somewhat human "comprehendible" language.

Turing machines are described in many textbooks as if they were real machines and often their mechanical features and the description of their operation is given in detail. While this approach may help to become familiar with the concept, it may also distract from the purely conceptual nature of the Turing machine. Even Turing described his "logical calculating machine" using the paper tape, on which symbols are written, changed or erased as an analogy for memory. However, a true Turing machine is not a machine made of physical things, like a read-write head, a controller box and a very long tape as working and storage space, but a mathematical and entirely abstract thing. While certain properties of Turing machines can never be realized in physical systems, i.e. electronic computers, most practical problems for a Turing machine can generally be accommodated nevertheless. Alan Turing remarked in a lecture to the London Mathematical Society regarding the relationship between Turing machine and digital computers that

[11] The term "operation" in relation to Turing machines relates only to thought experiments or approximations with pen and paper.
[12] Harnish claims that that von Neumann machines are "designed, in part, to overcome the liabilities of Turing machines" (Harnish 2002,133).

machines such as the ACE[13] may be regarded as practical versions of this same type of [universal Turing] machine. There is at least a very close analogy. Digital computing machines have all the central mechanism or control and some very extensive memory. The memory does not have to be infinite, but it certainly needs to be large. (Turing 1947).

Given the technological advances that have been made in the fifty years since, the "analogy" between Turing machines and digital computers has become even closer, in the sense that a modern computer could emulate even very complex Turing machines.

[13] The automatic computing engine, or ACE, had been proposed in 1945. This proposal is technically very detailed and contains functional descriptions and circuit diagrams of all major components. The paper is reprinted in D.C. Ince (ed), *Mechanical Intelligence, Collected Works of A. M. Turing*, 1992, Elsevier.

A definition for Turing machines.

In order to examine the limitations of the mathematico-logical concept of computation, it necessary to examine the properties of Turing machines. After establishing briefly what these properties are, I will show that some philosophical issues, like Lucas's argument against mechanism (Lucas 1964, Lucas 1970), have little relevance for an artificial intelligence. The Turing machine in the context of computer science and theory of computation can be described in mathematical terms. Sipser (Sipser 1997, 128) gives a formal definition of a Turing machine as a

> 7-tuple, (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}) where Q, Σ, Γ are all finite sets and
>
> 1. Q is the set of states,
> 2. Σ is the input alphabet not containing the special *blank* symbol '_',
> 3. Γ is the tape alphabet, where $\{\_\} \in \Gamma$ and $\Sigma \subseteq \Gamma$,
> 4. δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is the transition function,
> 5. $q_0 \in Q$ is the start state,
> 6. $q_{accept} \in Q$ is the accept state, and
> 7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$

This definition and other equivalent forms[14] show the link between logic and symbol processing. The rules of the computational process are embedded in the entries on the machine table d. This table comprises typically many entries, which explicitly determine the transition from one state of the machine to the next. The input alphabet S can contain any symbols whatsoever, as long as they are discrete i.e. they can be uniquely identified. The first *blank* denotes the end of the input string and can therefore not be used inside the input string[15]. These are essentially arbitrary but semantically interpretable symbols, which are processed (read, written or erased), according to the specific machine states at the time. Moreover, the machine states, the tape

[14] Wells describes a Turing machine as a quadruple (K, S, d, s) omitting the *accept* state and *reject* state of Sisper's model. Wells also incorporates the tape alphabet (G) into S. (Wells 1996, 35)

[15] This is a matter of convention. Any symbol can be used as a marker to denote the end of the input, as long as it is reflected in the machine table of that particular machine.

alphabet and the transition functions, are all expressed in terms of elements of *finite* sets. A Turing machine accepts no additional information for the execution once the machine is in the start state q_0. It follows that the computational process on a Turing machine and all computation on equivalent "real" machinery, when programmed to implement a Turing machine, are strictly deterministic[16], discrete, closed and traceable.

The properties of Turing machines have been the cause of long and sometimes heated debate in relation to AI. It is claimed that they are *formal systems*, that they are *discrete state* machines, and that they are *deterministic* and *closed* systems. Additionally, Turing machines are *finite* in their set-up, although they may run for an infinite amount of time. The input string of a Turing machine, specialized or universal, must also be finite. This is also true for the tape or memory itself, at least for machines that will eventually halt. Any machine that halts does so after a finite amount of time and can only use finite memory. For "useful" machines there is no need for infinite tape. By term "useful machines" I mean machines that halt and implement algorithms. For this type, it is required that there is always *sufficient* memory available to the machine – in the sense that there can always be more accessed, if necessary.

All of these properties are the result of the machines' very design. Or, more accurately, these properties make up the machine. It is for these properties, that Turing machines may not be suitable to produce intelligent systems or may not be able to model the human mind, as Lucas (Lucas 1964, Lucas 1970), Dreyfus (Dreyfus 1992), Searle (Searle 1980), and others have argued. I will deal with these properties in turn and I will outline some of the philosophical issues and consequences arising from them.

[16] Turing machines do exists as a *non-deterministic* variant. However these non-deterministic machine all have a deterministic equivalent machine. The non-determinism of Turing machines is restricted to several possible state transitions from a finite set of choices. Moreover all of the possible choices are predetermined when execution of the algorithm begins.

Turing machines as formal systems.

J. R. Lucas argued against Mechanism and against the idea of artificial intelligence in his famous paper *Minds, Machines and Gödel* (Lucas 1964). Lucas's argument against mechanism is built on the claims that Gödel's theorem applies to all formal systems and therefore to all levels of computing machinery as well, whereas minds themselves are not constraint by Gödel's theorem. He writes that

> Gödel's theorem must apply to cybernetical machines, because it is of the essence of being a machine, that it should be a concrete instantiation of a formal system. It follows that given any machine which is consistent and capable of doing simple arithmetic, there is a formula which it is incapable of producing as being true –i.e., the formula is unprovable-in-the-system – but which we can see to be true. It follows that no machine can be a complete or adequate model of the mind, that minds are essentially different from machines (Lucas 1964, p 44)

Although Gödel's argument is a mathematical one, the application of his theorem to other formal systems is generally accepted. In his 1961 paper, Lucas gives this one line précis of Gödel's theorem, which nevertheless can be regarded as an outline of the core result:

> Gödel's theorem states that in a consistent system which is strong enough to produce simple arithmetic there are formulae which cannot be proved-in-the-system, but which we can see to be true (Lucas 1964, 43)

It is necessary to present Lucas's objection in the context of his understanding of what a machine is. He emphasizes that a machine's behaviour is

> completely determined by the way it is made and the incoming 'stimuli': there is no possibility of its acting on its own. (Lucas 1964, 45)

Lucas makes essentially two claims in his paper. The strong claim is that *Mechanism* is false, which entails that no form of computing machinery or any machinery can ever be used to successfully implement something equivalent

to a human mind. The weaker claim is that it is impossible to implement a human mind or to successfully model a human mind using a Turing machine. The following argument demonstrates that Lucas's argument against Mechanism does not hold. A number of mechanical devices like analog computers and non-synchronous parallel computers are not equivalent to Turing machines (Sloman 1996). Sloman argues, correctly I believe, that human cognition involves more than Turing machine computation. Sloman lists among others non-synchronous parallel processes, continuous (analog) processes and chemical processes that are all involved in the human brain (Sloman 1996, 181). It is an acceptable claim that non-Turing machine computation is not subject to Gödel's theorem. Benacerraf agrees that Lucas's claim against mechanism is insufficient, if mechanism entails "non-Turing machine" computing.

> It is an open question whether certain things which do not satisfy Turing's specifications might also count as machines (for the purpose of Mechanism). If so, then to prove that it is impossible to "explain the Mind" as a Turing machine (whatever that might involve) would not suffice to establish Lucas's thesis – which is that it is impossible "to explain the mind as a *machine".* (Benacerraf 1967, 13)

Sloman has successfully argued this "open" question and we must accept that machines can compute functions, which are not computable by Turing machines. Non-Turing machine computation can be done on *real* computing machines, but Lucas's concept of a machine explicitly demands that the machine be deterministic. Lucas wants to simply exclude all devices that do not fit into his concept in order to protect his argument. In *The Freedom of the Will*, Lucas writes

> We should say briefly that any system which was not floored by the Gödel question was *eo ipso* not a Turing machine, *i.e.* not a computer within the meaning of the act. (Lucas 1970, 138)

Lucas's strong claim, that *mechanism* is false, is certainly not sustainable. There have been several successful refutations of Lucas's arguments on the

grounds that the argument is logically flawed, or invalid. (see Slezak 1982, Whiteley 1962).

Lucas seems not always clear about who can find Gödel sentences – sentences that cannot be shown true within a formal system – in which system. The question is not about minds over machines, but about one formal system against another. Gödel showed that the formal system of arithmetic contains propositions that cannot be shown to be true within arithmetic. Gödel's incompleteness theorem implies that Gödel himself, if he were a formal system, will contain such propositions as well. Lucas could "in principle" be able to find them in Gödel and vice versa. The propositions in Gödel's incompleteness theorem are *about* arithmetical propositions and are not arithmetical propositions. This leads to a very interesting condition, when we compare two "copies" of a formal system against each other. Each copy would contain Gödel sentences, which can only be specified by the other copy.

There are two more points in Lucas's argument, which give rise to some concerns. Lucas's argument rests on the assumption that the mind can always apply some method to a Turing machine to show that this particular Turing machine contains an unprovable statement. He claims that a mind can "see" that such statements are true, while the machine, because of Gödel's theorem, cannot prove them to be true. The important point here is to clearly determine what kind of statements can be made *within* formal systems and what type of statements can be made *about* formal systems. Lucas's mind is outside the formal system and observes that it is possible to introduce a statement into this system, which cannot be proven with the rules and axioms of that system. Lucas claims that in essence, Gödel sentences are contradictions akin to "a partial analogue of the Liar paradox" (Lucas 1970, 129), which a machine could not resolve. Whiteley sums up the challenge posed by Lucas:

the trap in which the machine is caught is set by devising a formula which says of itself that it cannot be proved in the system governing the intellectual operations of the machine. This has the result that the machine cannot prove the formula without self-contradiction. (Whiteley 1962, 61)

However, Whiteley recognizes that Lucas has reduced Gödel's theorem to an inability of the machine to resolve self-contradicting statements. I think that this is an over-simplification and misinterpretation of Gödel's theorem, but it is Lucas's interpretation after all and Whiteley uses this interpretation for his own counter argument. Whiteley claims that Lucas cannot resolve the statement "This formula cannot be consistently asserted by Lucas" himself without contradicting himself. Lucas has been placed into the same position into which Lucas wants to put the machine: Lucas is inside his own formal system and the rest of us are on the outside. Anyone reading this sentence can *see* the truth of this statement, while Lucas is unable to do so. Lucas can be trapped in the same way that Lucas wants to trap the machines. Whiteley points out that human minds can deal with contradictions of this kind easily in that we can make a statement *about* the contradiction. He suggest that Lucas can

escape from the trap by stating what the formula states without using the formula: e.g. by saying 'I cannot consistently assert Whiteley's formula' (Whiteley 1962, 61)

It is this type of action that Lucas wants to deny to the machine. Whiteley claims that a machine can be programmed, in principle, to deal with contradictions of this sort. The machine can use all the axioms and rules in an attempt to prove a formula and indicate a positive result, or the machine may indicate that it cannot prove the formula because of some non-resolvable condition, a circularity, which might cause the machine to deadlock. The machine can make some statement *about* the formula such as "I cannot prove the formula: 'The machine cannot consistently assert this formula'".

Lucas's arguments are not convincing and have been refuted for a variety of reasons. While Turing machines remain in principle still open to a Gödelian attack, Lucas fails to recognise that many forms of computation with physical

systems are not necessarily formal systems i.e. they are not “machines within the act”.

Turing machines as deterministic and closed systems.

Under the heading of organized machines, Turing describes in *Intelligent Machinery* (1948) computing devices that are “apparently partially random” and “partially random” (Turing 1948, 113). The partially random machine is one that allows

> several alternative operations to be applied at some points, … the alternatives to be chosen by a random process. (Turing 1948, 113)

The deterministic character of the Turing machine is not compromised by this proposition. Turing specifically asks that a random choice be made from “several alternative operations”, which demands that at any one time there is only a *finite* number of choices. It can be shown that for every non-deterministic Turing machine there exists an equivalent deterministic Turing machine (see Sipser 1997, 138). All Turing machines are therefore deterministic. The perceived determinism of machines in general is called upon in many arguments against mechanism and AI, because it seems to offer an easy and ready-made argument against “free” will. In support of his main argument against mechanism, Lucas assumes that all computing machines are absolutely deterministic. He claims that the behaviour of machines

> is completely determined by the way it is made and the incoming ‘stimuli’: there is no possibility of its acting on its own: given a certain form of construction and a certain input of information, then it must act in a certain specific way. (Lucas 1964, 45)

He claims further that this inherent determinism limits the machine not only to what it *must* do, but also, more importantly for Lucas, what the machine *can*

do: Machines, unlike minds, are restricted to purely mechanical acts. Lucas denies machines any form of decision making on the grounds of their inherent determinism. He also rules out the introduction of randomization and statistical tools to simulate decision making by machines. To avoid that a machine reaches a state where it might become inconsistent, Lucas claims that

> clearly in a machine a randomizing device could not be introduced to choose any alternative whatsoever: it can only be permitted to choose between a number of allowable alternatives. (Lucas 1964, 45)

It is not clear whether Lucas rejects that a "randomizing device" can possibly be built or that "randomizing devices" are not allowed so as not to endanger his argument. I assume that Lucas holds the view that random numbers, or events, cannot be produced within a closed and deterministic system. This view is consistent with Turing's as far as Turing machines are concerned, but Lucas extends this position beyond Turing machines and attributes deterministic behavior to machines in general. Having "established" that all machines are deterministic, Lucas claims in *The Freedom of the Will*, that the "physical determinist" will insist on his ability to describe a human in finite terms, because there are only a finite number of beliefs and only a finite number of possible inferences. Lucas argues that such a mechanistic model of a mind must result in describing human reasoning as a "proof-sequence of formulae". From here, Lucas recalls Gödel's theorem again to make his claim against physical determinism:

> We now construct a Gödelian formula in this logistical calculus, say *L*, which cannot itself be *proved-in-the-logistical-calculus-L*. Therefore the particular human being who is, according to the physical determinist, represented by the logistic calculus L, cannot produce such a formula as being true. But he *can* see that it is true: any rational being could follow Gödel's argument … Therefore a human being cannot be represented by a logistic calculus … (Lucas 1970, 133)

On the surface, Lucas seems to argue in a logical and rigorous way. George (George 1962) offers an interesting reply by arguing that only deductive systems are fully deterministic, while inductive or probabilistic systems are not. Moreover, George claims that deductive machines, to which Lucas's argument applies, are of no cybernetic interest, and he suggest that

in cybernetics we are not dealing with machines that are wholly specified in advance. They are self-programming or self-organising and their subsequent behaviour will depend upon the environment in which they operate … It is therefore clear that no limit of the kind implied by Gödel's theorem can be placed on possible machines, where by machines we mean anything that can be *effectively constructed.* (George 1962, 62)

Sloman argues also against the validity of applying of Gödel's theorem to general computing machinery and says that

philosophical debates about Gödel's incompleteness theorem proving that there are limits to what a particular computing system can do, are irrelevant to the problem of what sorts of intelligent mechanism can be designed: for all these theorems are relevant only to 'closed' systems. i.e. systems without means of communication with teacher, etc. (Sloman 1998, 104)

Recall that it is a design feature of Turing machines that at the start of the execution *all* of the procedure, or program, and *all* data are provided. Turing specifically explains that Turing machines do not accept input, once they started to execute:

The types of machines that we have considered so far are mainly ones that *are allowed to continue in their own way for indefinite periods without interference from outside*. The universal machines were an exception to this, in that from time to one might change the description of the machine which is being imitated. (Turing 1948, 115, italics added)

The fact that the universal Turing machine is an exception relates purely to the fact that such a machine may start up with a different program. Program changes are coded and implemented *before* the machine starts. For a universal Turing machine the distinction between program and input-data is not that clear. The core program, which is the universal Turing machine itself, remains fixed, although there are many ways to implement such a core program. It is possible to change the core program as well as the particular

program before each task. The core program of the Universal Turing machine and the program to be executed by the Universal machine with the accompanying input data, transform the universal machine into a specific Turing machine for that task. Because there are no changes allowed from the start of the universal machine onwards, the entire process from the loading and execution of the specific task is utterly pre-determined. This is, of course, rarely true of "real" computers with "real" programs, which typically request some user-input during execution. Moreover, programs written for applications in AI are often influenced by external environmental factors through the use of sensors. "Real" systems are usually open and are therefore not deterministic. Any interaction with the real world where values of variables may influence the sequence of execution or may change data makes the system non-deterministic. Lucas only claims that any system, which

> … is sufficiently determinate to support physical determinism, and sufficiently careful to avoid inconsistency, … provides enough logistic structure for Gödel's argument to apply. (Lucas 1970, 134)

Why would an intelligent system, human or otherwise, have to be "sufficiently determinate" to support physical determinism at all? Arguably, all physical systems may be deterministic at some low level in a Laplacian sense. However, Lucas's argument is explicitly targeting decision making by

> an ordinary computer together with a few randomizing devices, selecting from a range of alternatives sufficiently circumscribed to avoid inconstancy. (Lucas 1970, 135)

Consider a system, engineered around a program that can accept and run chunks of executable code. A second system, which is provided with a variety of sensors, measures a series of environmental factors and outputs mathematical functions obtained through regression analysis. These functions are coded, compiled and transferred to the first system. There, these code fragments become imbedded in the yet to be executed code. Such a system cannot be regarded as deterministic, with randomizing devices and lists of choices, as Lucas would like it. Lucas's objections are not applicable to anything but the Church Turing thesis in the "proper" sense and Turing machines – the objections are only

concerned with theoretical entities. Lucas makes a startling admission when he considers the prospect of a non-deterministic machine actually being engineered:

> Perhaps one day there might be produced a computer which was so complicated that it ceased to be predictable, even in principle, and started doing things on its own account, or, to use a very revealing phrase, it have a mind of its own. It would begin to have a mind of its own when it was no longer entirely predictable and entirely docile, but was capable of doing things which we recognised as intelligent and not just mistakes or random shots, but which we had not programmed into it. But then it would cease to be a computer with the meaning of the act, no matter how it was constructed. We would say, rather, that we had created a mind …(Lucas 1970, 137)

The argument here is about whether von Neumann machines are Turing machines "in the meaning of the act" or not. Lucas writes

> … any description of human beings and human activity which is entirely rule-governed and admits of only regularity explanations is sufficiently rule-governed to constitute a formal logistic calculus, and is open to Gödel-type arguments. (Lucas 1970, 165)

When I first read this passage I highlighted these words and placed two words in the margin next to it: *Yes!* and *So?*.

Yes, because Lucas is right in his argument that Turing machines are always deterministic and hence candidates for the application of the Gödelian argument. *So?*, because von Neumann machines are not closed systems. They can be programmed to act on "what goes on in the world", resulting in machines that are not "entirely predictable" and machine that are "entirely docile". However they do *not* cease to be computers.

Lucas is very much aware that real computers can do a lot more than Turing machines; otherwise he would not have to exclude them explicitly from his arguments. These arguments rest on assumptions about the limits of Turing machines, namely their deterministic behaviour. I have argued that computing systems, which interact with the environment and are acted on by the environment, are not necessarily Turing machine equivalent, because the property of determinism cannot be assumed.

Turing machines as discrete systems.

Any Turing machine is a discrete state machine and the vast majority of computing machinery employed in AI is based on the von Neumann computer architecture, so that almost all modern digital computers have the "discreteness" property from their very design. There are many processes in the physical world, which we consider continuous. It can be argued that this may not be true for real physical things, because at quantum level, so the argument goes, everything happens in small discrete jumps. The physical world may exhibit universally a very fine granularity. However part of modeling the physical world is achieved through mathematics and in this domain continuous functions *do* exist. Turing machines, however, cannot model systems that are either continuous or dense[17].

Generally, continuous processes can be approximated in digital computers to any desired accuracy. The only constraints are limited storage capacity and time taken for computation. For engineering purposes approximations are usually "good enough" and for artificial intelligence, as far as computer interfaces to the real world are concerned, this seems also true[18].

If a computer model is implemented on a computing machine, which uses some analog processing components, then the entire system can no longer be considered a Turing machine. Modern computers and the applications (programs) running on these machines are interrupted by operating systems, use real time data and are subject to many other influences. Turing anticipated this and explored some of the possibilities for "unorganised" machines. He used the term "interference" to denote external influences acting on running machines.

There is the extreme form in which parts of the machine are removed and replaced by others. This may be described as 'screwdriver interference'. At the other end of the scale is 'paper interference',

[17] *Dense* is describing the situation where some entity is not continuous, but consists of infinitely many discrete parts. The set of rational numbers is a good example. This set of numbers is obviously not continuous – it is a set of discrete numbers -, but between any two rational number is another one.

[18] As far as the human senses are concerned, electronic or suitable mechanical machinery is typically much more sensitive.

which consists in the mere communication of information to the machine, which alters its behaviour. (Turing 1948, 115)

It seems clear that Turing differentiates quite clearly between "Logical Calculating Machines", digital electronic computing machines, and computing machines in general. In *Computing Machinery and Intelligence* (Turing 1950) he reminds us of the conceptual machine automating an essential human activity, which can be executed with pen and paper and then he contrasts this with a more challenging computing device:

An interesting *variant* on the idea of a digital computer is a 'digital computer with a random element' … Sometimes such a machine is described as having free will (though I would not use this phrase myself). (Turing 1950, 438)

This "variant" is not to be considered a Turing machine and it is also beyond reach for a Gödelian attack.

Turing machine as finite systems.

Turing machines are composed of finite sets of several entities. Recall Sisper's definition of a Turing machine as a "7-tuple, $(Q, S, G, d, q_0, q_{accept}, q_{reject})$ where Q, S, G are all finite sets" (Sipser 1997, 128). It is not intuitive that a Turing machine is a finite machine, which may under certain conditions produce *non-finite* output, or may execute indefinitely, or both. The important concept is that at the beginning of the execution, at state q_0, the machine is completely defined, with a finite number of entries in the machine table (d), and the finite set of entries on the machine's input tape. This follows from the "proper" Church-Turing thesis: the input must be finite. If this input was not finite, then a human computer could not perform this operation either. Turing demanded that the rule-of-thumb, i.e. algorithm, has a finite number of steps:

The "computable" numbers may be described as the real numbers whose expressions as a decimal are *calculable by finite means*. (Turing 1937,116, italics added)

The necessity for an algorithm to be finite is part of its definition - an effective procedure or an algorithm must terminate to be considered an effective procedure. Haugeland introduces the concept of a formal system by comparing such systems to types of games we play. He says

> A *formal system* is like a game in which tokens are manipulated according to rules, in order to see what configuration can be achieved. In fact, many familiar games – among them chess, checkers, Chinese checkers, go and tic-tac-toe – simply *are* formal systems. But other games – such as marbles, tiddlywinks, billiards, and baseball – aren't formal at all (in the sense we care about). What's the difference? All formal systems have three essential features (not shared by other games): they are "token manipulation" games; they are digital; and they are "finitely playable". (Haugeland 1986, 48)

Turing machines are formal systems and Haugeland rightly points out that formal systems must be "finitely playable". For a process to be algorithmic, any interaction with the world, i.e. accepting input from a human via the keyboard say, must only allow choices from a *finite list*. During a game with a chess program the human player can select only from the list of possible and valid moves at any particular state of the game. At the level of the chess program, we can argue that it is Turing machine computing, i.e. strictly algorithmic. There are many tasks, which can be performed on a real computer, that allow input that is not taken from a list of possible inputs. For example, I can start a program that is in fact an interpreter of some higher computer language, like Prolog or Python. Other than the restrictions due to the particular syntax of the language, I can write and execute any program I can imagine. I argue that playing with, i.e. programming in, Prolog or Python is not "finitely playable". From here we must concede that computers, when they are running a program that allows creating and execution of *ad hoc* tasks – other programs in fact –, are not formal systems. If a computer under certain conditions is not a formal system, then this computer is also not a Turing machine under these conditions. A computer in this scenario will also violate the rule that a Turing machine must eventually halt. An ideal "real" computer will never halt, as long as the operating system is stable, the hardware holds up, and power is supplied. The problem of determining what is Turing

machine equivalent or not, seems to be depending on the level of inspection. If we disregard the actual level of implementation of a chess program running on a personal computer, we can argue that the chess program is algorithmic and therefore Turing machine computable. We can imagine that a human with pen and paper could “in principle” follow the chess algorithm as well, although in real terms this is an intractable problem. The human could certainly do it for a game of Tic-tac-toe. However, the human could not do it for a game of “operating system” - not even in principle. The operating system program is not an effective procedure, but much more a stimulus – response program. While there are algorithmic routines embedded, the execution of many of these routines is essentially depending on outside events. Most critical is that operating systems under normal conditions do not *halt*. The necessity for an algorithm to end at some point in time is an essential part of its effectiveness. The proper Church-Turing thesis stipulates that whatever a human can compute by following a procedure can also be done by a Turing machine. Being able to do something implies being able to finish the task.

Some numbers are clearly computable through the application of an algorithm, although their decimal expansion has an infinite number of digits. The trick here is that a finite procedure or algorithm can be used to determine any *arbitrary* number of digits. It is important to note that a Turing machine cannot produce all digits, although the Turing machine can be programmed to produce *n* digits and then *halt*. To produce all digits of a non-terminating number, the machine would have to run forever. This poses the problem that it is not possible to accept a Turing machine’s output “so far” as a solution. The algorithm, that the human would use has typically a rule like: calculate the square root of two to ten significant digits, or repeat the procedure until the error is less than 10^{-4}. Real programs on real computers have such *do…until* constructs, or the like, as well. The point here is that any useful procedure must eventually stop. Once the procedure has terminated we can obtain the produced output. The result of a Turing machine computation is only valid when the machine *has* halted. After all, the marks on the tape may only represent some intermediate result. Analogously, it makes little sense to

examine the registers or memory of a machine to look for a result while it is still running.

Turing machines in operation are *automatic* formal systems. A computer is automatically manipulating symbols according to certain rules of the system (Haugeland 1981, 38). Haugeland rightly includes the automatic state transitions in the list of what constitutes computation. All 'useful' computation involves some mapping of inputs to some outputs according to some rule. Even for trivial functions, like $y = x$, a simple computing machine must still automatically read the input *x* and write it to the output *y*. The machine must even for the most trivial forms of computation change state automatically, even to do nothing useful at all. The shortest Turing machine program needs at least two states: a start-state and a stop-state (halt). The transition between these two states follows according this single rule in the machine table, but the machine does so *automatically*. This particular process of calculation, which I would like to call a *null-algorithm*, is the smallest program possible. Note that a null-algorithm is distinct from a no-operation instruction (NOP), which is implemented in many computer systems. A NOP is a designed function to waste a certain number of clock-cycles by going through the fetch-execute-cycle, while a null-algorithm does not have any statement at all. Both programming elements cannot form a program by themselves – they must be followed by a proper program-terminating command. The computation of a null-algorithm in a higher program language like C must execute at least one single statement to end, abort or exit. A program without *any* executable statements does not exist [19].

It has been argued that the concept of logical-mathematical computation, i.e. in a Turing machine sense, is too narrow to escape attack from a Gödelian attack. However, real and practical computation with inputs and outputs and possible interfaces to the real world, as it is performed during many applications on digital

[19] It is possible to write programs in some higher languages that do not contain any executable statements in the source code. In C, for example, *main() {}* is a program that does not do anything at all. However the closing *}* is an implicit *end* statement, for which the compiler will generate some executable code.

computers, is beyond the constraints of Turing machines. Open computer systems do not have the necessary properties that any Turing machine must have. I have argued that real computing machines with real programs are not deterministic. Therefore real computers can do much more than Turing machines can do. In fact, real computers may have everything necessary "to generate general intelligent action". [20]

[20] A phrase used by Newell and Simon (Newell & Simon 1997).

Chapter 4

Physical symbol processing systems – GOFAI.

So far I have argued that “real” and practical computation extends beyond the logical and mathematical concept, which is based on the computational power of Turing machines. Turing machines can only solve algorithmic problems; in fact, the purpose of their inception was to *define* algorithmic computability. Turing machines and physical instantiations of these, i.e. Turing machines emulated on the von Neumann architecture, are symbol manipulating systems. Such systems modify symbols or bit patterns according to rules that are specified in a machine table, in the case of Turing machines, or in programs, in the case of modern computing machines based on the von Neumann architecture. Moreover, von Neumann machines are supersets of Turing machines in terms of computing power, because of their openness and input-output functionality, as I have claimed.

One aspect of AI research is based on the hypothesis that physical symbol systems like modern computers are indeed suitable for the task of engineering intelligent systems. Many have argued against this assumption and claim that Turing machine computation or computation in general is insufficient (Dreyfus 1992, Searle 1980). Alan Newell and Herbert Simon are considered the founders of the physical symbol systems hypothesis. Much of what has been achieved in GOFAI and AI in general has its origins in the work of Newell and Simon. Their theories are based on a series of assumptions and definitions, which have been subject to further interpretation and refinement over many years. Indeed, Newell and Simon revised and restated certain aspects of their own earlier work. Newell gives a definition for a physical symbol system as follows:

a physical symbol system consists of a set of entities, called symbols, which are the physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus a symbol structure is composed of a number of instances (or tokens) of symbols related in some physical way (such as one token being next to another). At any instant in time the system will contain a collection of symbol structures. Besides these structures, the system also

contains a collection of processes that operate on expressions to produce other expression: processes of creation, modification, reproduction, and destruction. A symbol system is a machine that produces through time an evolving collection of symbolic structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves. (Newell & Simon 1997, 86)

The description of a physical symbol system must be seen against the claim of what such a system is supposedly able to do. The following passage states the core assumptions about the relationship between intelligence and physical symbol systems. In support of their hypothesis, Newell and Simon make several fundamental claims, namely that

a physical symbol system has the necessary and sufficient means for general intelligent action. … By "necessary" we mean that any system that exhibits general intelligence will prove upon analysis to be a physical symbol system. By "sufficient" we mean that any physical symbol system of sufficient size can be organized further to exhibit general intelligence. By "general intelligent action" we wish to indicate the same scope of intelligence as we see in human action: that in any real situation, behavior appropriate to the ends of the system and adaptive to the demands of the environment can occur, within some limits of speed and complexity. (Newell & Simon 1997, 87)

Newell and Simon also establish a probable connection between physical symbol processors, i.e. computing machines, and human minds. They claim that there is at least a strong similarity between the two:

At this point I wish to be explicit that humans are symbol systems that are at least modest approximations of knowledge systems. They might be other kinds of systems as well, but at least they are symbols systems. (Newell 1990, 113)

These definitions and claims about physical systems, the possibility of engineering intelligent behavior, and the assumption that humans are "at least" symbol systems, have been the subject of various arguments and objections. I will present some of the arguments for and against the basic assumptions of the physical symbol processing hypothesis against the background of computation.

The basic assumptions.

Hubert Dreyfus, probably the most prominent critic of GOFAI discusses some of the "assumptions underlying the persistent optimism" Dreyfus 1992, 153), and he claims under the heading *the biological Assumption* that

> the view that the brain as a general-purpose symbol-manipulating device operates like a digital computer is an empirical hypothesis which has had its day. (Dreyfus 1992, 162)

Many of Dreyfus's arguments against the physical system hypothesis stem from the application of phenomenology to artificial intelligence. Human beings in the process of decision making, learning or problem solving are always immersed in the world. The claim is that human beings experience the real world only through the phenomena that we receive from the real world. Dreyfus (Dreyfus 1992) points out that the phenomena are more than a personal interpretation or representation of the world. He says

> When we are home in the world, the meaningful objects embedded in their context of references among which we live are not a model of the world stored in our mind or brain; *they are the world itself*. (Dreyfus 1992, 266)

Dreyfus includes personal experiences, personal plans, and memories in this world. If memories and future plans are part of the world, as we perceive it, then our reasoning will also be influenced by this private perception. Human thinking and reasoning is dependent on the current personal context. The semantics of symbols in this perceived world are also determined within some particular context. In human cognition, symbols do not have a fixed meaning and the context that determines the meaning of a symbol cannot completely be described in a symbol system itself. Dreyfus believes that certain knowledge, which an intelligent system must be able to work with, cannot be made available to a computer in any appropriate form. He says that

The ontological assumption that everything essential to intelligent behavior must in principle be understandable in terms of a set of determinate independent elements allows AI researchers to overlook this problem. (Dreyfus 1992, 206)

Dreyfus's objection is that AI workers seem to accept that all necessary information can be "symbolized" and stored as propositions and sets of rules. He claims that the meaning of such elements will be dependent on the context of their use, and the resolution of ambiguity by rules and sub-rules will lead to an infinity of rules (Dreyfus 1992).

The claim that a computer may not be the appropriate tool for an inquiry into intelligence has been presented in even stronger forms. Fetzer (Fetzer 1998) argues that at least some human cognitive processes are not algorithmic. He illustrates that some thought processes do not fit into a computational paradigm. *Dreams and daydreams*, *memory*, and *perception* are examples, according to Fetzer, that show that at least some cognitive processes are not algorithmic and therefore the computational theory of mind should be rejected. He says in his conclusion that

Even if some of our thought processes are computational, most of them are not, which makes *our best theory* either trivial or false (Fetzer 1998).

Fetzer's argument relies on he assumptions, analogous to Dreyfus's, that non-algorithmic processes are instrumental in achieving intelligent action or intelligent behaviour. There is good evidence that analog and other non-digital processes are in fact going on in the brain and we can safely assume that the brain is "responsible" for intelligent behaviour. We must accept that the achievements, within the field of AI in the quest for systems that exhibit intelligent behaviour, have generally been disappointing. Nevertheless, there has been no conclusive argument to show that the brain is necessarily the *only* suitable system to ultimately exhibit "intelligence". There is a further difficulty with this type of argument, because the strengths of all arguments against a computational theory of mind and a computational theory of AI seem to be directly related to the understanding of what the *premises* of the arguments are. For example, if the

term computation is taken to be synonymous with Turing machine computation then Pylyshyn's claim "cognition *is* a type of computation" (Pylyshyn 1984, xiii), could possibly be rejected. However, if computation entails anything that maps some input to some output, then Pylyshyn's assumption is very probable indeed. The confusion in the arguments is even greater, because the *conclusions* are also open to interpretation. The plausibility of a computational theory of mind is judged on what type of computational processes are assumed to take place in the brain. There are other approaches on offer, which may be helpful in resolving these issues.

Rapaport suggests that computationalism, i.e. a computational theory of the mind, should be based on the premise that "cognition is comput*able*, not that it is computa*tion*", as Pylyshyn suggests (Rapaport 1998, original italics). In doing so, Rapaport offers an elegant solution to the question of what the relationship between brains, minds and AI might be. On one hand, Dreyfus and others have legitimately claimed that brains are not digital computers and that brains compute in ways we do not (yet) understand. On the other hand, the field of AI can achieve its goals, because computers *can* compute "digitally" or otherwise what the brain does somehow. A "computa*ble* theory of mind" caters for alternative possibilities for intelligent systems based on different, yet related, principles.

Obviously the notion of computation has to include more than what Turing machines offer in terms of computing power (in the sense of computability not performance) for either approach. Rapaport remarks that

> The kind of thing that can be computable is a function, i.e. set of ordered pairs – 'input-output' pairs, to use computer jargon - such that no two pairs have the same first elements but different second elements. Roughly, a *function* is computable if and only if there is an 'algorithm' that computes it, i.e. an algorithm that takes as input the first elements of the function's ordered pairs, manipulates them (in certain constrained ways), and returns the appropriate second elements. To say that it is an *algorithm* that does this is to say that there is an explicitly given, 'effective procedure' for converting into the output. (Rapaport 1998)

Rapaport remains committed to computation in the Turing machine sense and effectively closes the doors on the opportunities he opened earlier. He argues that for activities, which are usually considered not to be algorithmic – e.g. cooking a dinner – can nevertheless be expressed as an algorithm in the Turing machine sense. He concedes that this would have to be a "very complex" algorithm, but his point is that the cooking of the dinner is a *result* of an algorithm (Rapaport 1998). It follows that cooking a dinner is computable. The difficulty with this argument is that there would have to be a very complex algorithm for every possible dinner, if cooking as an activity is treated this way. Rapaport's reductionism for achieving Turing computability introduces a great deal of complexity and probably even intractability for more elaborate "dinners". Higher level computation in a less constrained form that can operate on concepts rather than discrete interpretable symbols seems a more promising approach. Rapaport's notion of a function as an 'ordered pair' is in my mind also a much too narrow and too mathematical definition. A more acceptable and practical definition of a function has to include alternative ways in which the mapping of the input to output may occur. Simple look-up tables, algorithms and inductive reasoning may all be employed in human brains to produce outputs for a range of given inputs. Attempts to model, simulate or replicate such processes should not be restricted by eliminating possible methodologies on the grounds that "the brain does not work like this" or "computer can only do this or that". The computable versus computation proposition is interesting because AI as an engineering discipline can continue engineering things that seem to be doing *what* brains do, without having to worry about that some methods may not reflect *how* brains are doing it. GOFAI, unlike connectionism, is not directly analysing and utilizing brain structures in the development of intelligent systems. The activities within GOFAI *need not* be legitimized or justified by comparison with observable biological entities.

The assumptions concerning *intelligence* are as important as the assumptions made about the relation between symbol systems and human minds. Newell (Newell 1990) defines intelligence as the capacity or ability to use every bit of available information or knowledge in the process of achieving the goal. Together

with this definition, Newell offers an explanation to elucidate these assumptions. He points out that

> if a system uses *all* of he knowledge that it has, it must be perfectly intelligent. (Newell 1990, 90)

This claim seems to me ignoring the issues surrounding the frame problem, which is concerned with the need to separate important information or knowledge, which is *relevant* for a particular situation or problem, from the total body of knowledge a system may have. Dennett offers an account of the frame problem in his essay *Cognitive Wheels* (reprinted in Dennett 1998) and he concludes:

> What is needed is a system that genuinely *ignores* most of what it knows, and operates with well-chosen portion of its knowledge at any moment. Well-chosen, but not chosen by exhaustive consideration. (Dennett 1998, 197)

It seems that the ability *not* to use all of the available information must also be judged as evidence for intelligence, which contradicts Newell's claim. The frame problem is also negating a second point of clarification, in which Newell claims that intelligence is limited to the knowledge which is available to the system at a particular point in time. If part of the knowledge is inaccessible during evaluation for reasons that are within the system, then the failure to use this knowledge can be construed as exhibiting lack of intelligence. (Newell 1990, 90). However Dennett clearly shows that the solution to the frame problem is to purposefully make a lot of knowledge inaccessible to the system.

The definitions offered by Newell and Simon seem much more complex and in need of refinement to make them workable in the context of cognitive science and AI. In fact, Newell and Simon modify and qualify the previous requirements to the extent that they introduce the *heuristic search hypothesis*.

> The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search – that is, by generating and progressively modifying symbol structures until it produces a solution structure. (Newell & Simon 1997, 96)

The frame problem illustrates that a symbol system should *not* make use of all the available information in a goal-seeking activity. The remaining difficulty is how to determine what information should be left out. To make exhaustive searches in large search spaces tractable, a lot of information has to be eliminated, before a search for a particular solution can even begin. Dennett points out that relevant information must be "well-chosen, but not chosen by exhaustive consideration" (Dennett 1998, 197). Dennett illustrates in his well-known "robot with a bomb" example that elimination of unnecessary information by exhaustive search is ineffective. Brute force searches cannot solve the frame problem. Newell suggest that exhaustive searches can be refined and made workable by the introduction of heuristics. The issue here is that some heuristics are used to get to "good enough" solutions for the problem at hand. Heuristics are often shortcuts based on previous experience, or have to be introduced because of incomplete information for an informed search. Often, like in chess playing systems, a complete search of the solution space is intractable for reasons of storage (memory) or time constraints. Often heuristics used in search algorithms are based on human experiences, not on experiences of the symbol system. Deep Blue, for example, plays a good game of chess, because the heuristics within the system mirror the good chess games played by masters over many years, not because Deep Blue makes use of all the information (knowledge) it has. A chess machine could have *all* the possible information, albeit only in principle, *all* possible valid moves, which would be the condition that Newell required for a truly intelligent symbol system in his first claim. However no symbol system, including humans, can do that given time and memory constraints and it would follow that humans are not intelligent either. They cannot play chess properly, because as symbol systems the humans must make use of all the available information to be intelligent. The difficulty here is to reconcile what a system *can* do with what a system *should* do. A computer would win every chess game if the computer would play chess like a Turing machine with a full game tree. A game of chess is essentially a trivial game and the rules can be coded as an algorithm for a Turing machine. The difficulty lies in the size of the solution space that has to be stored and searched. The inability to manage that space, which represents

a gigantic “move” look-up table, forces us to introduce heuristics in to the chess playing programs.

Elsewhere, in view of the problems with brute force searches, Newell and Simon present yet another and very different view on what intelligence entails:

> The task of intelligence, then, is to avert the ever-present threat of the exponential explosion of search. (Newell & Simon 1997, 102)

Newell and Simon present definitions for intelligence and the detection of intelligent behaviour. While it seems obvious that different principles must be applied for different problems at hand, it becomes clear that there is no single uniform approach. Some technical issues turned out to be much more complex than the earlier assumptions. Hoffmann points out that Newell and Simon’s hypothesis is rather general and that symbol systems appear to be equivalent to the notion of the Turing machine (Hoffmann 1998, 178). The physical symbol hypothesis is also dependent on the computational framework through the connection between the syntax and the semantics of the symbols.

Semantics.

Many arguments against the physical symbol-processing hypothesis and against AI as such have been presented over the years. Lucas’s Gödelian argument and Searle’s Chinese room (Searle 1980) are arguably the most important philosophical objections to AI, at least if we were to judge by the number of contributions to the debate and the amount of literature on the arguments. Lucas’s and Searle’s arguments are of the *in principle* type in that, according to their claims, the endeavors of AI must fail, because some of the assumptions at the very foundations are flawed. I have considered Lucas’s arguments already, and it should be clear that his arguments are directed against an artificial intelligence that is based on symbol processing, rather than being directed against the connectionist approach, although the actual argument is based on the

formalism of such systems. Searle's objections are very different from Lucas's Gödelian argument and I will provide a sketch of the Chinese room argument.

Imagine a room with a human being inside, who has a library of books with instructions to modify symbols according to fixed rules: whenever there is certain symbol, replace it with another symbol, add another symbol, or erase a symbol. The human in the room does whatever the rules say and the person receives pieces of paper with symbols written on them through a window. After writing down other symbols, strictly according to his rulebooks, the results of this transformation – another piece of paper – are pushed out of the window. What the person inside the room does not know is that the symbols are in fact Chinese characters and that the result is interpreted as answers in Chinese to questions given in Chinese. Yet the person inside the room does not know or speak any Chinese; the symbols have no meaning whatsoever to the person inside the room.

The claim of Searle's argument is that syntax alone is insufficient to attribute any meaning to the symbols in symbol processing system, because purely syntactical system lacks intentionality – hence (strong) AI is impossible. Searle claims that strong AI cannot be realized on physical symbol processing machines, because the meaning of symbols remains externally interpreted. This argument, the many counter arguments, and Searle's refutations of the counter arguments have filled many volumes, however Searle's Chinese room is still debated [21].

The problem of semantics is essentially that of connecting the real world with the representation that are used in computation. Winograd and Flores explain that

> the problem is that representation is in the mind of the beholder. There is nothing in the design of the machine or operation of the program that depends in any way on the fact that the symbol structures are viewed as representing anything at all (Winograd & Flores 1986, 86)

[21] There have been many serious refutations of Searle's argument. While the argument is compelling on the surface, it has been shown that it is flawed because the conclusions are based on the wrong assumptions (see Copeland 1993, Haugeland 1998, Dennet 1998)

The claim that meaning is always external or, "there is nothing …in the program" as Winograd and Flores put it, is probably too strong. I argue that the syntax in a formal interpretable system restricts the semantics of its representations. In other words, representations are limited to what their meaning *can* be through the logical connections within such a system. When representations, symbols or values in variables are used as premises in logical inferences and some of these variables are dependent on external processes, then limits do also apply. For example it makes no sense semantically and syntactically to multiply strings by real numbers. An operation such as *name* * *Pi* (where *name* = 'Fred' and *Pi* = 3.14) makes no sense. The variables are semantically and syntactically of different kinds [22]. Any formal system will constrain the kind of data and the operations that can be done with certain kinds. It is possible to change one data type to another, but doing so has usually some 'side-effects': the variable *Pi* of the type *number* can be changed into variable *Pi* of the type *string*: (the number 3.14 becomes the string '3.14'). Now we can no longer multiply *Pi* by a number, because *Pi* is not a number. *Pi* is a variable of the type string that can hold "values" like 'Fred', 'Banana' or '3.14'. The names of variables are certainly arbitrary in a system, but the kinds of variables and the possible operation that can be performed are not.

Pylyshyn uses the term *semantic function* to describe the semantic content of a symbol within a certain context (Pylyshyn 1984). The concept of semantic functions can be explained in the context of a positional number system, like our decimal system. The symbol '5', say, is not necessarily always "meaning" five. The symbol '5' in '1560' means really five-hundred, while the symbol '5' in '12345' means five. The semantic interpretation of symbols according to their place-value is an example of a semantic function. The meaning of '5', as five or five hundred, is not intrinsic in the symbol, but is dependent on the wider context where and how the symbol is used. Any transformation rules for symbols in a physical symbol processing system have to maintain the semantics of such symbols. A computer program asked to perform a calculation may produce symbols, which

[22] I use the term *kind*, although *type* in the computer science sense would describe it better. However, *type* (and *token*) is used in the discussion of symbols and denotes something

were not included in the original problem: '? 6 + 5' will produce '11' as output. The answer '11' uses the symbol '1' twice, yet each '1' has different semantic value, namely 'one' and 'ten'. The point here is that the meaning of symbols is not *entirely* external to the computational system. Interpretable symbol processing by machines entails that certain semantic properties must also be maintained (see Haugeland 1981). In order to do that, rules about semantics must exist besides syntactical rules and the sets of rules are interdependent. Pylyshyn says that

> we need the syntactic or the symbolic level because we must preserve certain interpretations over mental operations, just as we must preserve the numerical interpretation by SF[23]…This we can do only if we have a semantic function whose definition has access to the generative structure of the symbolic expression.(Pylyshyn 1984, 62)

The interpretation of symbols, or assigning semantics to a symbol, is not an entirely arbitrary action. Because of the interdependence of syntactical rules and semantic functions, the interpretation of some symbols (1 as ten, 1 as one) is governed by the result of syntactical manipulation. The string of digits as the output of some numerical calculation by a computer must be interpreted primarily as a number - I do not think there is any other choice. Once this output is accepted as a number, there are many possible meta-interpretations, depending on the context of the calculation. This number may be re-interpreted afterwards as a temperature, a grey level of a pixel, a frequency, and so on. Winograd and Flores claim "that representation is in the mind of the beholder" is not necessarily true at the input-output level of a physical symbol processing system.

The future of GOFAI.

The development of intelligent system has not produced the results that one would have expected or hoped for, given the huge increase in performance of modern computing machinery. The processing power in terms of speed has increased in the order of several magnitudes and memory, especially with the

---

different.

[23] semantic function

use of virtual memory [24], is for practical purposes unlimited. Winograd's *SHRDLU* (1971) has been cited as one of the great breakthroughs in the history of intelligent machines (Harnish 2002, Kurzweil 1990). Since then the mood has changed. Progress is slow - in fact even some of the GOFAI pioneers and researchers have their doubts about the future of some aspects of AI. Haugeland writes

> Despite its initial plausibility and promise, however, GOFAI has been in some ways disappointing. Expanding and organizing a system's store of explicit knowledge seems at best partially to solve the problem of common sense. This is why the Turing test will not soon be passed. (Haugeland 1997, 21)

Dreyfus's concerns are much deeper and he claims that GOFAI has failed altogether. He says

> indeed, what John Haugeland has called Good Old-Fashioned AI (GOFAI) is a paradigm case of what philosophers of science call a degenerating research program[25] (Dreyfus 1992, ix).

Dreyfus attacks GOFAI from several angles, one of which is the perceived need for "common sense" for intelligent behaviour to occur. Even the largest database of propositional knowledge is, according to Dreyfus, insufficient to resolve ambiguities. Usually, the resolution of ambiguities does not present any difficulty to humans. Implementing a large database with many rules and data items or truths will need many meta-rules on how to apply the rules whenever there are changing circumstances and contexts. It has been claimed that a schema of rules and meta-rules will lead to an infinite regress of rules about rules.

Lenat, the driving force behind the *Cyc* project, seems to ignore this argument. While many believe that GOFAI is no longer considered an area of serious

[24] Virtual memory refers to the practice of storing the content of data in fast (expensive) random access memory (RAM) onto slower (cheaper) magnetic storage, whenever there is no need for immediate access. The operating system swaps this data back into RAM when the running processes need to access it later. The small time penalty for swapping memory is usually accepted, given the financial savings.

[25] The term "degenerative research program" goes back to Imre Lakatos (*Falsification and the Methodology of Scientific Research Programs,* in Lakatos and Musgrave, *Criticism and the Growth of Knowledge*, 1974, Cambridge UP)

research, Lenat and the *Cyc*-team have taken the “brute-force” road to intelligent systems. *Cyc*, so they claim, will have the intellectual capacity of a nine year old. This will be achieved by creating an expert system containing the largest collection of “knowledge” and rules of inference ever. Any large system will be constrained by storage capacity, as it grows to house ever more data and associated rules. The underlying support structures i.e. database management and other operating system components will eventually suffer speed degradation, which will impact negatively on the performance of an intelligent system in turn. Much of this degradation could possibly be offset by a sufficient amount of “brute force” - alternatively, parallel processing might offer a solution. Others maintain that the symbol processing methodologies (GOFAI) cannot be abandoned. Hoffmann (Hoffmann 1998) comments that symbols are necessary at some level of explanation of a cognitive system. Whether these symbols are used at the lowest level of implementation or at higher levels of abstraction, is a matter of the complexity of the system in question. Nevertheless symbols are required at some level of explanation in a non-symbol approach.

> …in more complex systems, the use of symbols for describing abstractions of the functionality at the lowest level is inevitable. …any description of a sufficiently complex system needs layers of abstraction. Thus, even if a non-symbolic approach uses tokens at its base level which cannot be reasonably interpreted, there still needs to be a more abstract level of description. (Hoffmann 1998, 257)

Winograd (Winograd 1990) challenges the idea that connectionism may be fundamentally different and points out that representations are still at the basis of connectionist systems.

> Although the new connectionism may breathe new life into cognitive modeling research, it suffers an uneasy balance between symbolic and physiological description. Its spirit harks back to the cybernetic concern with real biological systems, but the detailed models typically assume a simplistic representational base much closer to traditional artificial intelligence (Winograd 1990, 184).

Symbol systems will continue to be useful in their own right as a methodology for building systems and will also be necessary in aid of other methodologies, to explain and describe functional levels.

Chapter 5
Connectionist Systems.

From very beginnings of artificial intelligence, systems based on models of human physiology have been part of that research. The earliest computational models of neurons are those of McCulloch and Pitts in 1949. The proposed model had been shown to be able to compute any computable function, provided a sufficient number of the neurons were connected in the appropriate way (Haykin 1999, 38). Donald Hebb's work *The Organization of Behavior* (1949) had a profound impact in that it described how the synaptic strength of neural connections is depended on the history of their activation. This insight forms the basis of how artificial neural nets store and modify "knowledge". When Rosenblatt introduced the *perceptron* as a neural model, which included an effective learning algorithm, the research and development of neural networks became the main field of inquiry within AI. The hopes and funding for the inquiry into artificial neural networks sharply declined after the publication of the book *Perceptrons* in 1969. In their work, Minsky and Papert showed rigorously that single layer nets of perceptrons have severe limitations on what they can compute (Minsky & Papert, 1969). The book contained a small section in which Minsky and Papert explained that the limitations that they have outlined may not be applicable to multi-layered neural networks. Nevertheless, the "monograph … did not encourage anyone to work on perceptrons, or agencies to support work on them" (Haykin 1999). In the last two decades, neural networks have again become a major research topic after the initial work of Rumelhart and others on back-propagation learning (Rumelhart et al. 1986).

The basic assumptions of connectionism.

The connectionist approach to AI and cognitive modeling assumes that the structure of the brain and the components within are essential for the realization of observed phenomena, i.e. intelligence. In other words, it is necessary or at least "helpful" to base any theoretical and practical work on what has been uncovered by neuro-science. The task at hand for AI is then to build good models of neurons, or primary elements (PE), and build structures with these models to represent brain structures. Most of the neuron and structural models are realized in software only, but the functionality of the individual artificial neuron is more or less based on the physiology of a human neuron. A human neural cell changes the electro-chemical signal on the output (the axon) depending on the state of its inputs (synaptic connections on the dendrites). The synaptic strength of individual connections between biological neurons is represented in artificial neurons by multiplying the signal of connections by some factor, the weight. These weights are the essential part in neural networks with respect to learning and representation of knowledge. Simple models of artificial neurons and their intended functionality can be described in mathematical terms:

$$O_n = g(i_n) \text{ ,where } i_n = \sum i_j w_j$$

The output of the neuron o_n is a function $g()$ of the sum of the neuron's weighted inputs ($i_j w_j$). The step function, or Heaviside function, has typically been used as the activation function $g()$ for earlier neural models:

$$g(i_n) = \{ 0 \text{ if } i_n < 0; 1 \text{ if } i_n => 0\}.$$

If the sum of the weighted inputs i_n is less than zero, the output will be zero; when the sum is zero or greater, the output will be one. Because of the definition and implementation of this particular activation function, the neural

modal exhibits a digital “all or nothing” behavior. In recent models this function has been replaced by the sigmoid function:

$$g(i_n) = 1 / (1 + e^{-xc})$$

This most important property of this function is that it is continuous and differentiable, which is a necessary property for certain learning algorithms. The constant *c* allows for the adjustment of the “step”- characteristic of the function. For large values of *c* the graph of the transition functions between zero and one becomes more steeply.

The different models of neurons that can been described in mathematical terms can also be converted into programs and data structures in a variety of ways. The implementation of such mathematical models in terms of the actual data and physical structures on computer systems (arrays, vectors or the like) is only of minor importance. For the purpose of this work I will assume that the functional specifications in mathematical terms are not compromised in their particular instantiation on a computer. Programmers will use different data structures to implement neural models, because a particular computer language or computer system may offer particular features resulting in an effective and elegant solution in terms of memory efficiency or speedy execution, or both. We can therefore ignore some of the issues that relate to the implementation of artificial neurons for the time being and list some of the mathematical and computational implications.

Rosenblatt’s *perceptrons* were designed as a two-layer network. A two-layer neural net has only one layer of weighted inputs between the input layer and the output layer. Irrespective of the number of neurons and the number of connections between individual input units and output units, this type of network cannot solve some very basic functions.

Each artificial neuron or primary element (PE) is typically implemented as a two-step computation. Firstly there is an input function, which calculates the

weighted sum of all activated inputs of the PE. The second step is to activate the PE's output according to some activation function. Both of these functions are linear in the case of the *perceptron*. The input function is simply the sum of products, the output function is a step-function – the output is either 0 or 1, depending on some predetermined threshold value. This particular model of a neuron, the *perceptron*, can only classify outputs if these outputs are linearly separable. It can be shown easily that a two-layer neural network cannot solve the Boolean *xor*-function (see Haykin 1999, Hoffmann 1998, Russell & Norvig 1995). A much more in-depth treatment of the limits of linearly separable functions can be found in Minsky and Papert (Minsky & Papert 1969). It is important to note that Rosenblatt was able to demonstrate a general algorithm for perceptrons that will learn any separable function with sufficient training examples (Russell & Norvig 1995, 575).

Multi-layered neural nets containing one or more additional layers of neurons, generally referred to as *hidden units*, are much less limited. Although multi-layered networks have been described earlier, the introduction of the error-back-propagation method to change the weights of connections between neurons gave the technology a fresh start during the 1980s. The back-propagation method, or "back-prop" algorithm, analyzes the contribution of error for each neuron to the total error and adjust the weights according to this contribution. Russell and Norvig sum up the properties of the back-propagation algorithm:

> the trick is to assess the blame for an error and divide it among the contributing weights (Russell & Norvig 1995, 579)

The adjustment of weights is performed on all units starting from the output units 'backwards' to the hidden units. The back-propagation algorithm is a form of gradient descent and relies on some additional assumptions and properties, which go well beyond the simple perceptron. Because a gradient descent algorithm is based on partial derivatives of the activation function, it is necessary that the error function itself is a differentiable function. To be

differentiable, this function must be continuous. Obviously, the activation function (*Heaviside* or step function) that is used in a perceptron is unsuitable. The function most commonly chosen as a replacement is the logistic function, which I have described earlier. Unlike the *perceptrons*, neural models using non-linear components, behave functionally in different ways. The behavior of the individual neuron effects the behavior of the entire network.

> For some ranges of inputs … these units exhibit an all or nothing response (i.e., they output 0.0 or 1.0). This sort of response lets the units act in a categorical, rule-like manner. For other ranges of inputs, however, … the nodes are very sensitive and have a more graded response. In such cases, the nodes are able to make subtle distinctions and even categorize along dimensions which may be continuous in nature. *The nonlinear response of such units lies at the heart of much of the behavior which makes networks interesting*. (Elman & Bates et al. 1996, 53, original italics)

Elman et al. observe that in many cases the behaviour of a neural net seems not to be "programmed in" (Elman & Bates et al. 1996, 56). Neural nets are no longer completely predictable once they are even moderately complex, either for the number of neurons or for the number of connections. Many aspects in their design, like choosing the number of hidden nodes, are not well understood and often the tuning of a neural net is essentially done empirically by trial and error. This non-predictability of neural nets or the inability to fully model their behavior has no theoretical basis. Their behaviour can be described *in principle* at all levels, as long as they are closed – as long as there are no external factors influencing the net's behavior. The unpredictable behaviour stems from our inability to describe the net in proper mathematical terms and that the problem of prediction becomes intractable. Neural nets are dynamical systems, whose future behavior depends largely on initial conditions. The aspect that neural nets may contain units, which are activated by probabilistic methods, is complicating the issue further. Unlike the program structures in GOFAI applications, the workings of neural nets are extremely difficult if not impossible to trace, because of the parallelism of the structures. Moreover, there are no "semantically interpretable" symbols anywhere, no

variables to be checked for truth conditions and the like, except at the implementation level of neural net itself.

If there are no symbols, can we still refer to the processes in a neural network as computation? It is possible to build up a real neural network by implementing the functionality of every simulated neuron in a network with a single integrated circuit, an operation amplifier (OpAmp) of some kind [26]. All the connections between them and the associated weights can be implemented using resistors. The transformation of the input patterns to the output pattern, or the mapping of the inputs on the outputs, would be identical to that of the simulated network. In the real OpAmp network this mapping would be carried out according to Ohm's and Kirchhoff' laws. There are no discrete steps nor are there any symbols: there is no Turing machine like computing anywhere. Yet, the network still "computes" some function *f* that maps inputs net_{in} onto some outputs net_{out}

$$net_{out} = f\,(\,net_{in}\,)$$

The notion of computation must include more than what a definition in terms of Turing machine can offer. There is no doubt that the OpAmp network can be simulated as an artificial neural net – after all, that is what the OpAmp net replaced. The function that is performed by the OpAmp net is also computable: we can build up an exhaustive list of all input/output mappings, as long as our inputs and outputs are discrete. A perhaps very long list of statements like

if net_{i1} = 1 and net_{i2} = 4 … and net_{in} = 34 then net_{out} = 124
if net_{i1} = 2 and net_{i2} = 3 … and net_{in} = 12 then net_{out} = 125

could describe the network in an algorithmic form. A human being with paper and pencil can, at least in principle, evaluate such a list of instructions in a mechanical fashion. The interesting point here is that there are several ways

[26] This type of standardized component is readily available. They are linear over a wide range and they are relatively simple to use.

to implement some function *f*. Moreover, the type of implementation and the level of implementation are essentially determining whether we count the resulting process as *computation*. The OpAmp network and the simulated network at its functional level are behaving according to Kirchhoff's law and by doing so, the networks compute some function *f*. Three distinct modes of computation emerge, which are only dependent on their physical implementation. Theoretically the real networks can compute this function *f* in the domain of all real numbers. For the simulated network the real numbers can be approximated to an arbitrary number of significant digits. The Turing machine computable version of the function, i.e. the "exhaustive" list, must be finite so that the procedure can be executed in finite time. It follows that the domain of the computable function *f* must be restricted to a finite set of discrete numbers for effective computations. Turing machines (and humans) can compute *any* number for that function, but not all numbers of that function.

Knowledge and learning.

Many aspects of neural nets are conceptually very different from architectures used in GOFAI. Knowledge representation and learning raise some particular issues in the bottom-up approach. While knowledge in classical (GOFAI) systems is represented through semantically interpretable symbols, the very absence of symbols in neural nets leads to the question how and where *knowledge* is stored in neural networks (see Hoffmann 1998, Smolensky 1990, Boden 1991)

In expert systems, arguably the most advanced form of classical AI, knowledge is introduced to the system by declaring certain values to variables (facts) and rules of inference about those facts. In *Prolog*, which is typical for an entire class of languages and systems often used in classical AI, facts[27]

[27] *Facts* are true within the *Prolog* system by definition or declaration. They may or may not true in the real world. The term *fact* is used as a technical term here.

are expressed as relations. A statement like *female(pam)* or *male(bob)* establishes the fact that something named *pam* is an instance of something called *female* or the fact that something named *bob* is an instance of something called *male*. Similarly, a relation in the form *parent_of(tom,pam)* tells the system that something named *tom* is something called *parent_of* something called *pam*. *Prolog* can recall facts in response to a question. If asked a question *?- female(pam)*, *Prolog* would respond *yes*, because something named *pam* is in fact an instance of something called *female*. The system would respond to the question *?- parent_of(X, pam)* with *X=tom*, because it can instantiate the variable *X* with *tom* from the fact that *tom* is a *parent_of* something *pam*. If the fact that *mother_of(pam,bob)* is given to the system, then it will be a truism that *bob* is the *mother_of pam*. The names of facts and rules such as 'mother' or 'parent_of' are selected for human understanding only, and the meaning of facts and the associated rules are completely arbitrary. The unary relation *male(X)* might also be called *xtxtvb(X)*, as long as we keep the system consistent. Therefore, if Bob is represented as *mnb_2,* the *xtxtvb((mnb_2)* becomes a truism when given to the system. Rules can contain relations of other rules and the *Prolog* system can resolve complex relationships. The definition of *mother_of* should be improved to also stipulate that *mother_of* implies the *female* attribute: *mother_of(X, Y) :- parent_of(X, Y), female(X)*, which translates to the notion that '*X* is the mother of *Y* ' is defined as *X* being a parent of *Y* and that *X* is also female. Someone's grandmother is a mother of one of the person's parents. In *Prolog,* this would look something like *grandmother_of(X, Y) :- mother_of(X ,Z), parent_of(Z, X)*. A *Prolog* system may contain thousands of facts and rather complex relations or rules and such a system will deduce from its facts and rules the answer to questions in the form: is it true that Pam is grandmother of Bob? Or question in the form: who is a parent of Pam? Knowledge within such systems is represented as symbols and their meaning is defined by definition: the human concept 'mother' can be represented in such systems as *mother(…)*, *female_parent_of*(…), or *q24uie(…)*. In *Prolog* facts are expressed as membership of sets, i.e. *mother(pam)* can be interpreted that *pam* $\hat{I}$ *mothers*. It is important to note that the relationship

between *pam* and *mother* is always maintained and that inferences can be drawn from the fact that *pam* is a *mother* and that one of the *mothers* is an entity named *pam*.

A schema for a representation of knowledge must also include a system to interpret and extract that knowledge. Simply finding stored facts is of course trivial, at least, if the number of facts is not too large. However, the system must be able to deduce new facts from the already stored knowledge and a set of rules of inference. A system of relationships between members of sets can be described by using combinations of logical the operators *and*, *or*, and *not*. Earlier we saw that a mother could be defined as someone's female parent: *mother_of(X, Y) :- parent_of(X, Y), female(X)*. The relationships *parent_of()* and *female()* must both be satisfied. From here we can also deduce that if someone is a parent and if that someone is female then this someone is also a mother. This statement can be written formally as a formula in the first order predicate calculus:

$$\forall x \forall y \;\; \text{parent\_of}(x,y) \wedge \text{female}(x) \rightarrow \text{mother}(y)$$

The number of elementary rules needed to establish even the most complex inferences is relatively small. Copi (Copi 1979) lists nine rules of inference and ten rules of replacement. A set of six rules is needed to resolve propositions containing quantifiers such as $\forall x$ (for all x) or $\exists x$ (there exists at least one x). A large collection of facts that is operated on with classical rules of inference and quantification rules does not always produce the outcomes we expect. Human reasoning is not necessarily adhering to the same kind of rules; depending on the context we might implicitly apply additional rules. Utterances like "fish don't fly, birds do" are usually perfectly acceptable. However, we do know that some fish fly and we do know that some birds cannot fly. Generalizations may be valid, i.e. acceptable or "reasonable" within a certain context, but they may not be valid universally. One approach to resolve this issue in AI systems could be through the introduction of probability as a measure of "certainty". Decision-making and logical

inferences, which include elements of probability, are not uncommon in AI systems. The likelihood of events occurring independently or depending on some other events can be mathematically described and can be modeled in programs. Bayesian networks[28], probabilistic networks and causal networks represent a common approach of implementing such systems[29]. The use of probabilistic information during decision making is intuitively acceptable, because, as it seems, humans do very similar things. Unlike the mathematically correct interpretations of probabilities in computers systems, humans get calculations involving probabilities notoriously wrong. For example, many people are terrified during air travel, but do not even consider the much higher probability of getting hurt or killed during the trip to the airport by car.

In neural networks representations for 'facts' and associated rules for their transformation or manipulation do not exist in any comparable form. A neural network performs a transformation of the input stimulus, a pattern of sorts, into an output pattern. The output pattern might be as simple as 'yes' or 'no' or the network may be able to produce hundreds of characters or sounds. Essentially, the network transforms one pattern into another pattern by minimizing the error during the transformation.

During the transformation of the input patterns the contents are not preserved. The transformation process of a neural network is strictly irreversible. The feedback process cannot be used to re-generate a previous input from an output. A network, for example, which is trained to recognize numbers from images, cannot reproduce the bit patterns of the recognised numbers. A neural network can however create 'truths' by recognizing patterns and categorizing them. What used to be a fuzzy 5-ish shape at the input nodes is transformed into a '5' at the output by a probabilistic process.

[28] The term *network* does not refer to neural nets. The analogy is used to describe the interdependency of variables (nodes) and the probabilities attached to the neighbouring nodes.

[29] For an introduction to the theory of probability and an introduction into probabilistic reasoning see Russel and Norvig (Russell & Norvig 1995).

To say that pattern recognition in simulated neural networks is merely non-linear regression, using straightforward statistical methods with out memory or intelligence involved, is certainly an oversimplification. Knowledge is contained or distributed within the network structure as a whole rather than stored as 'data' for an inference engine, as it is the case for many classical AI systems. In neural networks, knowledge is "hidden" in the connections and their associated weights within the entire structure. These networks are, to a point, monolithic in contrast to their architecture. This monolithic property impacts on the storage capacity or knowledge capacity of a network. In GOFAI systems the capacity is practically unlimited, given that there is enough physical storage, without the need for any changes to the logic or without any impact on the information that is already stored. Van Gelder points out that for parallel distributed processing systems, addition of new information results in the modification of old knowledge.

> Depending on the particular scheme in question, it is typically the case that storing more items results in gradual worsening of performance across all or most items rather than an inability to store any particular one. This is the much vaunted *graceful degradation* of distributed systems and clearly follows from the nature of superposed representation (or rather, one important aspect of it). (Van Gelder 1991,48)

There are functional layers within the architecture, - a layer of input nodes to which the inputs to the outside world are connected, hidden layers and finally an output layer -, however there are no recognizable forms of data or rules to be found within. This is quite different from an expert system where rules and facts are separable from each other and from the supporting structure i.e. the program.

Smolensky offers an explanation for how the representation of knowledge could be understood and he suggests that

> the semantically interpretable entities are *patterns of activation* over large number of units in the system, whereas the entities manipulated by formal rules are in the individual activations of cells in the network. (Smolensky 1997, 239)

Smolensky introduces the concept of *subsymbolic* computation to establish a link between individual neurons and a semantically interpretable level of abstraction for interpretable symbols. Statistical inference rules and connection-strength manipulate these subsymbolic structures. (Smolensky 1997).

There is a fundamental difference between memory access in a von Neumann machine and memory access in a neural net. Access to a storage location in a von Neumann machine is a primitive operation: a value of a variable is accessed through a read operation of a memory location associated with the variable. This operation is entirely different for neural nets. The memory of a neural net cannot be accessed by means of location or by name. In traditional systems, it is possible to get information by looking at a memory location by its address or by its name[30]. In neural nets this not possible. Neural nets have to have their memory "jogged" in order to retrieve information.

When a memory is retrieved, it is "addressed" by its *contents*: a fragment of a previously-instantiated activation pattern is put into one part of he network (by another part of the network), and the connections fill out the remainder of that previously-present pattern. This is much a more involved process than a simple "memory fetch". (Smolensky 1997, 241)

This statement is problematic for several reasons. It is one of the connectionist fundamental assumptions that "knowledge" is fully distributed in the connections throughout the net. It is not only claimed that knowledge is fully distributed, but it is also claimed that memory within networks is associative.

A neural network system looks for closest matches, much as our brains locate memory contents by matching an example an input stimulus than by any addressing scheme. For example, if you hum a few bars of a tune and it is one we have heard before, we may be able to 'name that tune'. (McCord & Illingworth 1991, 61).

This assumption is unlike the principle of holograms for which it is assumed that *all* information is at any one point. For neural nets it is only assumed that *all* the information is essentially *everywhere*. A small bit of neural net only contains a small bit of information, which explains the effect of graceful degradation. Taking a few neurons out of a network -real or artificial- has usually no perceptible effect on the overall performance. However, unlike in a hologram, a small portion of a neural net is insufficient to have access to, or re-build, the total information. A hologram allows the reconstruction of the entire image from a fragment, albeit in degraded form.

[30] Strictly speaking this is not true. Reference by name is essentially a service function by the operating system or the implementation level of the system. Typically there is a (hidden) look-up table to resolve the name / address resolution for the user.

Smolensky seems to subscribe to the idea that a fragment of an activation pattern has enough information to re-build the rest of the network. He argues for the subsymbolic paradigm where the symbols of GOFAI are replaced with "fine-grained" subsymbols (Smolensky 1990). He explains that

> subsymbols are not operated upon by "symbol manipulation": they participate in numerical – not symbolic – computation. Operations that in the symbolic paradigm consist of a single discrete operation (e.g. a memory fetch) are often achieved in the subsymbolic paradigm as the result of a large number of much finer-grained (numerical) operations. (Smolensky 1990, 307)

The assumption that the operations at the subsymbolic level are "large number of much finer grained operations" does indeed apply to neural models, which are implemented on a digital computer. In computer models, numbers, i.e. the product of weight and some value representing the output potential of some other firing neuron, represents a neuron's inputs. This product is represented in some arbitrary machine format as an approximate for some real valued variable. The implementation of the artificial neuron introduces therefore a low-level fine granularity to the model. The mathematical, theoretical model lacks this granularity, because the input i_n can be a real number. The input i_n and the sum S must also be a real value, because we want the activation function to be continuous. The real values of the sum S form the domain of the differentiable activation function.

This granularity at the implementation level of artificial neurons may not apply to biological neurons; in fact there is good evidence that analog processes, which have no discreetness by definition, form at least part of a neurons functionality (Churchland & Sejnowski 1992, Bear & Connors & Paradiso 1996). Discrete entities are not explicitly part of the idealized mathematical model, which does usually not take timing issues into consideration. What may be needed to make the subsymbolic hypothesis applicable for the biological neuron, the mathematical model, and the realization on a computer, is a theory for a calculus for infinitesimal subsymbols. For now, the subsymbol hypothesis does not explain anything outside the realm of computer models. The notion that information is distributed statistically and *smoothly* throughout the network could then be applied in all contexts. The low-level granularity, which must be accepted in computer

modeling and computer simulation, is of no concern for the functionality of the net: it is possible run the model to any arbitrary level of approximation. It is possible, at least theoretically, to run a model at levels, where perturbations due to noise would already influence the real biological neuron.

Whether information is distributed smoothly or as a finite set of subsymbols does not change the ways in which this information can be accessed. To re-create an exact replication of a particular *activation pattern* does require the re-creation of the exact input pattern. Activation patterns are *one-to-one* functions of the inputs. It is of course possible to create an identical output pattern by stimulation of the network with an approximate input pattern. After all, that is exactly what we want to achieve with a classifier. A network, which has learned to separate apples from pears, should ideally output "apple" for all kinds of apples, and it should ideally output "pear" for all sorts of pears. However, the network can not list all apples and pears it has seen so far [31]. I would describe neural networks as information "one ways", where the exact input information is lost in order to retain a more conceptual form of the information. Output patterns are *many-to-many* mappings – a better description for classifiers would be *many-to-some*, or *many-to-few*.

McDermott argues that the notion of a symbol may also include a particular state of an analog system (McDermott 2001, 187). We can assume that "real" nets never settle into a state of rest and I would consider a brain in a truly stable state no longer a brain "in the meaning of the act". Artificial neural nets settle in different stable states depending on their input patterns. This is part of their design. Especially during the learning process, neural nets execute some learning algorithm every time the network has settled following exposure to some input pattern. Artificial neural nets in "production" are still clocked and their operation is therefore in discrete steps and also traceable. The outputs of an artificial neural net may be viewed as a sequence of discrete values and these outputs may be used as inputs for other computations. McDermott claims that "if anything like this

[31] There has been some work where partial feed-back of output patterns is used to create contextual relationships in networks for the recognition of serial events (Elman & Bates, et al. 1996)

is happening, then we are seeing the presence of discrete symbol tokens" (McDermott 2001, 188). Of course, it is possible to take the stable state of a neural net as the input of some other program. A neural net that has been trained as a classifier for character recognition could be connected to a program that may just count the number of times the network recognized individual characters. The assignment of the neural network's "value" at some time t_n into a variable of the counting system takes a quasi-analog and transforms it into a discrete entity. I use the term quasi-analog to indicate that the input-output relation is usually not a smooth one, because a classifier network does stabilize due to the clocked implementation and the learning process of the network itself. Also, the selection of the activation-function at the neural level is contributing to a certain degree of granularity. Recall that all-or-nothing response of a neuron is only compromised for a very small interval of inputs and that the use of the logistic function primarily satisfies the need for the error-function to be differentiable. Taking the output of a neural network as a discrete value does not introduce discrete "symbols" into the net, as McDermott claims. Arguably, the *entire* network could be taken as a single semantically interpretable symbol in the context of a larger system.

Given that there are no symbols within a neural net and that all knowledge seems to be distributed within, the question of how and what a network learns, arises. Dreyfus explains that

> in situations of supervised learning, it is really the person who decides which cases are good examples who is furnishing the intelligence. What the network learns is merely how to capture this intelligence in terms of connection strengths. (Dreyfus 1992, xxxix)

Dreyfus explicitly mentions that these concerns relate to supervised learning. His claim is also applicable to unsupervised learning, or reinforcement learning, where some feedback in form of overall cost or gain is used to re-adjust the connection strengths. This feedback is not the product of the neural net's intelligent, but it is an introduced stimulus to which the neural responds, according to the learning algorithm employed. The learning algorithm, the associated response mechanism and the stimuli are all the results of human reasoning and engineering. While this position emphasizes human superiority over machine, the issue is less clear cut. It is possible to construct a scenario where the "which

cases are good" - decision could be made by some process other than human reasoning. Reinforcement learning is already a step in that direction. The question is, at what higher level goals have to be specified, before the algorithmic "it is only doing what we tell it to do" stance, must replaced by "we have no idea how the network got there" attitude. Setting a goal for a system does not imply that the system has as an immediate consequence nothing to contribute towards that goal. The possibility of some contribution by the system towards the goal is not disputed, not even by Dreyfus, who at least accepts that a neural net can "learn" how to capture someone else's intelligence in connection strengths (Dreyfus 1992, xxxix).

Chapter 6

Serial vs Parallel Computation

The basic assumptions about classic physical symbols systems (GOFAI) and connectionist systems (neural nets) are vastly different. However, it has been argued that these differences do not exist at the computational level. A series of arguments has been based on claims that parallel and sequential processing is computationally equivalent (Penrose 1990, Dennett 1991). One of the difficulties with such a claim is that much depends on the level of description. At its functional level, a neural network is generally assumed to be implementing a parallel architecture. At the level of implementation this functionality may be realized on a serial or a parallel architecture. Neural networks are usually simulated with programs that are designed for a von Neumann machine, such as a personal computer (PC). A program, which is simulating a neural net and is running on a PC, is executed instruction after instruction i.e. in serial fashion [32]. However, a neural network can also be implemented on a different architecture. One could use separate microprocessors for each "neuron" and have these neurons interacting synchronously with each other. Such a model would represent an implementation of a truly parallel-distributed system. So, a neural net can be a parallel system *and* a serial system. The classification depends on the level of description. This example illustrates that the levels of description are of importance in the contexts of computation and artificial intelligence. Moreover, some philosophical considerations are also dependent on the level of description.

Some of the issues surrounding the distinction between serial and parallel processing can be illustrated by looking once more at the Turing machine. A variant form of the Turing machine is non-deterministic and the execution of some instructions does occur concurrently (in parallel). The idea behind a non-deterministic Turing machine is that the machine may assume, at certain points during the execution, one of several possible machine states. As with finite state

[32] On modern PCs this is often no longer the case. Some PCs have more than one processor and many components are effectively autonomous devices, so that a certain amount of parallelism can be assumed.

automata (FSA), the execution of a non-deterministic Turing machine can be traced by following *all* possible steps *in parallel* from the point where branching into multiple paths is possible. From here on, these paths are assumed to be executing independently of each other. A *halt* instruction, for example, could be executed in any of the possible paths, and if encountered, will halt the machine. However, it can be shown that for any non-deterministic Turing machine there exists an equivalent deterministic machine (see Sipser 1997, 138) [33]. This fact can be misconstrued as to indicate that non-determinism is Turing machine computable. Moreover, it may lead to the assumption that a single Turing machine could be found that is equivalent to an instance of some truly parallel computing. It comes as no surprise that the followers of the "Turing machine computation" paradigm will argue that there is no difference between serial and parallel computation. For example, Penrose says about parallel machines that

> Using more than one Turing device in *parallel action* - which is an idea, that has become fashionable in recent years, in connection with attempts to model human brains more closely – does *not* in principle gain anything (though there may an improved speed of action under certain circumstances). Having two separate devices which do not directly communicate with one another achieves no more than having two which *do* communicate; and *if* they communicate, they, in effect, are just a single device! (Penrose 1990, 63)

and that

> there is no difference in principle between a parallel and a serial computer. Both are in effect Turing machines. The only difference can only be in the efficiency, or speed, of the calculation as a whole. (Penrose 1990, 514)

Penrose's claim, that "communicating" computing devices are "just a single device", is only defendable in the context of deterministic machines. Moreover, Penrose allows for a "Turing device" to communicate, which cannot happen in terms of the definition of a Turing machine. It is a requirement for a Turing machine that the program (in the case of a Universal Turing machine) and the data are fully specified when the machine starts executing. Parallel processing is not possible on a Turing machine beyond the apparent parallelism in non-deterministic machines. Therefore, I believe, it is impossible to explain the

[33] Analogously, there is an equivalent deterministic FSA for any non-deterministic FSA.

difference between serial and parallel computing, or equivalence for that matter, unless we go beyond the level of Turing machines.

I have argued earlier in this document that computers are, in terms of their functionality, supersets of Turing machines. The question now is, whether there are grounds for claims that computing on "real" serial and parallel computers is equivalent. Boucher argues that parallel computation is fundamentally different form serial processing (Boucher 1997). He notes that networks, for example, cannot be represented on a single machine because,

> The input-output behavior of the whole network does not simply depend on the programs its computers are running, but also the respective speeds of the various computers involved. No single sequential computer, therefore, is equivalent to a network, which is a different entity entirely. (Boucher 1997)

A similar view is also put forward by Sloman (Sloman 1996), who says that for unsynchronized variable speed machines Turing equivalence cannot be shown. He argues that a system comprising unsynchronized parallel computers do not have

> well-defined global states and well-defined state transitions, as a Turing machine does (Sloman 1996, 181)

The systems described in these examples have in common that they describe autonomous machines (processes) that are linked together to form larger systems, and that the communication between the machines (processes) is *asynchronous*. In these circumstances, a single equivalent Turing machine cannot represent the entire system, because the interleaving of the actions of the individual processes is impossible. Boucher (Boucher 1997) and Sloman (Sloman 1996) argue correctly that unsynchronized parallel processes are not Turing equivalent, because unsynchronized parallelism is essentially non-deterministic. A simulated network made up of neurons, which are implemented on individual free-running serial processors [34], is non-deterministic because the execution time of the programs varies between the individual neurons. The summation function for

[34] Such a scheme might be useful when modeling the changes to the threshold due to the firing rate in individual neurons. Each neuron would have to have its own rate of decay and could not be subject to an overall (network wide) clocking mechanism.

these neurons' inputs and their weights is also dependent on the firing neurons' timing.

This scenario presents yet another example in support of the fact that two or more processes can communicate with each other in ways that prohibit replacement by one computational entity (or one Turing machine as Penrose claims). The most significant point is that the processes communicate asynchronously. In a truly parallel distributed neural network, a neuron communicates with its "neighbors", whenever it changes the output according to its activation function, or whenever one of its inputs changes due to some other neuron changing the output. *Whenever* is a function of a neuron's summation function, activation function and some non-deterministic component that is dependent on the timing of other neurons due to their own execution speed. Without a global (network-wide) clocking function, i.e. some synchronization, the network is in effect non-deterministic.

The level of description is also of importance in the context of symbols. Simulated neural nets compute functions in ways that are not comparable to the methods that are employed in classical symbol systems. Several levels of description have been identified and described for computer systems (see for example Hoffmann 1998). Here, I will only differentiate between the highest abstract level or functional level, i.e. "the neural net" or "the expert system", and the implementation level. The implementation level describes how the neural net or the expert system is working on the computer. This description may include descriptions of data structures and may also include details about the structure of the program at its highest level. When we look at some typical examples for such systems, we will discover very similar structures at the implementation level, which are common to all computer systems of reasonable complexity. Most probably, we will find a program written in *C* or *Pascal*, or some other language. We generally assume that these programs will be executed on some hardware correctly in their translated form, and we trust that the compiled programs behave exactly according to the instructions at the higher level language. Hoffmann, amongst others, points out that the lower levels of implementation are often deemed irrelevant (Hoffman 1997). At the implementation level the simulated

neural net and the GOFAI system are very similar indeed: both represent physical symbol processing systems. This may not be true at the application level, or functional level. The simulated neural net has no symbols and rules in easily recognizable form, which is in contrast to the GOFAI system, where facts and rules are the defining essentials.

Some of the philosophical objections against an artificial intelligence are made directly against certain levels. Penrose (Penrose 1990) and Lucas (Lucas 1964, Lucas 1970) can only apply their arguments at the implementation level. In fact, Lucas's argument can only be applied in the context of certain architectures. The arguments cannot apply to true connectionist systems because their architecture does not resemble anything to which Gödel's incompleteness theorem could be applied [35]: True parallel processing systems are not formal systems in terms of Gödel's incompleteness theorem. A formal system in the context of Gödel's theorem is a logic system comprising axioms and syntactical rules.

I have already outlined that the application of Gödel's incompleteness theorem in cases where the neural net is simulated on a von Neumann machine, is a different matter. The level of description is always an important factor for any decision whether a system is formal or not. This becomes also relevant for the question about computational equivalence at the functional level. Boden (Boden 1991) comments on the question of computational differences between GOFAI and parallel distributed processing (PDP) that

> in brief, all connectionist systems compute in a way that is different from the way in which a von Neumann machine computes, and some (such as PDP systems) consist of basic units that compute matters unlike anything that is dreamed of in folk psychology. (Boden 1991, 5)

Boden makes this statement about the functional level of a connectionist system. Only at this level are the two computational methods are truly distinct and different. I have shown that Boden's claim that "all connectionist systems compute in a way that is different from which a von Neumann machine computes" (Boden 1991, 5) is invalid at the implementation level, if the connectionist systems are implemented on a von Neumann machine. After all, at this level both systems are

"just programs" running on the same architecture. If we ignore the purpose of the programs, the simulated neural net *program* and the classical artificial intelligence *program* are arguably identical in terms of what happens at this level. While the programs might "do" different things at a higher level, they are equivalent at the lower level, because the observed actions and processes are of the same kind.

The situation can even be more confusing. It is conceivable to implement a GOFAI system on a vastly different architecture like a distributed parallel architecture [36]. At the functional level, an expert system is essentially a system comprising symbols and syntactical rules, and as such can be termed a formal system. Could a GOFAI system that is implemented on a non-formal architecture still be considered a formal system? Could it still be considered to be a GOFAI system?

Formal systems are subject to Gödel's incompleteness theorem - at least in principle. It would follow that Lucas's case against mechanism could also target artificial intelligence at the functional level, if computation could be seen as being independent of its implementation. Harnad supports the idea of implementation independence for computation and claims that

> the power of computation comes from the fact that neither the notational system for the symbols nor the particulars of the physical composition of the machine are relevant to the computation being performed. A completely different piece of hardware, using a completely different piece of software, might be performing the same formal computation. What matters are the formal properties, not the physical ones. (Harnad 1995, 384)

This position is essentially a stronger form of the generally accepted notion of implementation independence. Implementation independence assumes that lower levels in a system can be implemented in completely different ways, as long as the functionality remains according to the specifications for the levels above. These specifications are often the implementation criteria for a virtual machine. A program written in the *Python* language, for example, can be executed on various operating systems. The source program is the same for all environments,

---

[35] I ignore here any judgement about the validity of Lucas's and Penrose's argument in relation to von Neuman machines.

[36] Expert systems are running successfully in human brains for quite some time.

because it is written for the one *virtual* machine. The implementation of the "virtual" *Python* language itself is very specific for the various target systems. The different implementations make sure that, despite the different operating systems and different underlying hardware, the *Python* language behaves identically. The user of *Python* does not have to care about the details how this is done – the user deals with the same "virtual" *Python* system at all times.

If we agree with Harnad and view computation as a set of formal properties for some *virtual* machine, then any attempt to distinguish between serial and parallel computing becomes inconsequential. An artificial intelligence based on formal algorithmic principles, i.e. GOFAI, can be implemented on any architecture, as long as the underlying structures of software and hardware maintain the formal properties of the layers above. This will also be true for an artificial intelligence based on the connectionist architecture.

While it is true that neural nets are typically simulated on von Neumann machines, neural nets are considered by many not to be a theory of implementation, but they are regarded to be a (cognitive) architecture (Harnish, 2002). This is, I believe, the position that should be adopted also for GOFAI systems. Viewed in this way, connectionist systems compute in ways very differently from GOFAI systems, as Boden should have pointed out. Recall that Boden compared "connectionist systems" with von Neumann machines – "connectionist systems" should be compared with classic GOFAI systems.

Not everyone agrees with the idea that connectionist and GOFAI systems are fundamentally different. McDermott believes that

> At first glance [a neural net] *seems like* a very different model to computation from the computers we are used to, with their CPU's, instruction streams, and formal languages. These differences have caused many philosophers to view neural nets as a fundamentally different paradigm for computation from the standard digital one. … The whole thing is based on *exaggeration of superficial differences*. Anything that can be computed by a neural net can also be computed by a digital computer (McDermott 2001, 40, italics added).

McDermott's comment shows that the levels of description are important, and that the level of description should be stated explicitly as part of the argument. I have

argued that neural nets have little or no relationship to “CPUs, instruction streams, and formal languages”. McDermott compares different levels which each other: The neural net belongs at the functional level, whereas the CPUs, instruction streams and formal languages belong to a lower (implementation) level. In fact, formal languages have little or no relationship to CPUs either, because they belong to different levels themselves. McDermott’s arguments and claims are immersed in the confusion about the levels of description.

The point that connectionism is considered to be a cognitive architecture rather than a method for how to implement things (Harnish 2002) is the key. GOFAI and connectionism are to be viewed at the level of functionality, and then the two paradigms are distinct and fundamentally different. The implementation level and the tools that are employed for the implementation are, consequently, of little philosophical interest for artificial intelligence and cognitive science.

Conclusion.

I have argued that the Church-Turing thesis is often misinterpreted and that the Church-Turing thesis in its "proper" form does not concern what can be computed with computers. The concept of computation is, as far as computer science is concerned, grounded in the Church-Turing thesis and the Turing machine is the theoretical foundation on which computer science is resting. Furthermore, I have argued that the Church-Turing thesis and Turing machine concept of computing is of lesser importance for the field of artificial intelligence and cognitive science, because modern computing machinery can effectively interact with the environment and are supersets of Turing machines. Turing machines and the some associated limitations regarding the possibility of an artificial intelligence, as they have been outlined by Lucas are of little concern to the field. Gödel's incompleteness theorem is inconsequential for almost all applications in AI programming, and Lucas's attempt to prove that Mechanism, in its general form and in the more specific form as directed to digital computing machines, has failed.

In support of my claim I referred amongst others to Sloman, who claims that Turing machines are irrelevant for artificial intelligence (Sloman 2002). He agues that Turing machines are designed to run in ballistic mode, that is, Turing machines run essentially without external stimuli once their machine table and data have been set up. In artificial intelligence applications, computations are much more interactive. Especially in robotics applications, there are many sensors and motors, which may well form feed back loops. Turing's conceptual machine has no provisions for taking input from the outside world or to provide output for the control of motors or the like. Von Neumann machines have additional functional circuitry, which allows for memory locations to be altered from the outside, i.e. not under the control of the machine. This "interactive" mode, which leads towards the idea of situated cognition (see Clancey 1997, and Agre 1997), may be developed further to counter arguments based on phenomenology (Dreyfus 1992). Dreyfus himself

comments on the work by Agre and Chapman (e.g. Agre 1997, Chapman 1991) in which they explore ways to eliminate representations of the world in favor of interaction. Dreyfus says the work's result "is a system that represents the world not as a set of objects with properties, but as current functions" (Dreyfus 1992, xxxii).

Connectionism, as an alternative to the classical approach to artificial intelligence, has a different computational basis (Boden 1991, Harnad 1995). There is sufficient evidence from neurology to conclude that brains make use of analog processing (Bear et al. 1996, Churchland & Seijnowski 1992), which cannot be modeled by Turing machines. Real neural nets (brains) are examples of massive parallelism, whereas current artificial neural nets cannot be described as massively parallel computing devices, probably not even modestly parallel. The computational power and behaviour of neural nets is difficult to describe in terms of a mathematico-logical definition of computation. Because many aspects of connectionist systems are difficult to describe, their value in providing valuable insights into human cognition are challenged by some (Winograd 1990, Winograd & Flores 1986), and the possibility of any intelligence emerging out of neural nets, by others (Dreyfus 1992).

Others have argued, rightly I believe, that symbols are necessary at some level in any physical computational system, classical or connectionist (Winograd & Flores 1986, Smolensky 1990, Hoffmann 1998). I suspect that in the future, situated systems may well show that intelligent behavior can emerge for systems, which are based on symbols and that are grounded in the real world. Some of the theoretical properties of the tools, which are used to engineer these systems, are of little or no consequence. The computer, as a tool to achieve such goals, should not be the center of philosophical inquiry, especially, when some of the theoretical assumptions about computing machines do not generally apply to modern computers: I have argued in this thesis that modern computers are supersets of Turing machines. Sloman's comment applies to engineers and philosophers alike:

[A]pproaching a computer with a view to finding out what it can do is as silly as it would be for a physicist to study pencil and paper with a view what it can do. One approaches the computer in order to try to make it do something. The physicist writes things down, calculates, tries out formulae and diagrams, etc. He constructs, explores and modifies a theory. That is how to use a computer in order to study intelligence: by designing a program which will make it behave intelligently one constructs a theory, expressed in that program, about the possibility of intelligence. The failure of the theory is your own failure, not the computers. (Sloman 1978)

Bibliography.

Agre, Philip E. (1997). *Computation and Human Experience*. Cambridge UP.

Aspray, William (1990). *Computing before Computers*. Iowa UP.

Bear, M., Connors, B. and Paradiso, M. (1996). *Neuroscience: Exploring the Brain*. Baltimore. Williams & Wilkins.

Benacerraf Paul (1967). *God, The Devil, and Gödel*. The Monist, 51, 9-32.

Boden, Margaret (1991). *Horses of a Different Color?* in Ramsey, B., Stich, S. and D. Rumelhart (Ed.). *Philosophy and Connectionist Theory*. Hillsdale, New Jersey. Lawrence Erlbaum Associates.

Boucher, Andrew (1997). *Parallel Machines*. Minds and Machines, 7, 543-551.

Boyer, Charles B. (1986). *A History of Mathematics*. John Wiley & Sons.

Chalmers, David (2001). *A Computational Foundation for the Study of Cognition*. www.u.arizona.edu/~chalmers/papers/computation.html. (24/05/2001)

Chapman, David. (1991), *Vision, Instruction, and Action*. Cambridge, Massachusetts, MIT Press.

Churchland, Patricia S. and Terence J. Sejnowski (1992). *The Computational Brain*. Cambridge, Massachusetts. MIT Pess.

Clancey, William J. (1997). *Situated Cognition: On Human Knowledge and Computer Representations.* Cambridge. Cambridge UP.

Clark, Andy (2001). *Mindware: An Introduction to the Philosophy of Cognitive Science*. Oxford. Oxford UP.

Copeland, Jack (1993). *Artificial Intelligence: A Philosophical Introduction*. Oxford. Blackwell.

Copeland, Jack and Richard Sylvan (1999). *Beyond the Universal Turing Machine*. Australasian Journal of Philosophy, 77, 46-66.

Copi, Irving M. (1979). *Symbolic Logic*. New York. Macmillan.

Dennett, Daniel (1991). *Consciousness Explained*. Penguin.

Dennett, Daniel (1998). *Brainchildren: Essays on Designing Minds*. Cambridge, Massachusetts. MIT Press.

Dreyfus, Hubert L. and Stuart E. Dreyfus (1986). *Mind Over Machine*. Oxford. Basil Blackwell.

Dreyfus, Hubert, L. (1992). *What Computers Still Can't Do*. Cambridge, Massachusetts. MIT Press.

Elman, J., Bates, E., Johnson M., et al. (1996). *Rethinking Innateness: A Connectionist Perspective on Development*. Cambridge, Masachusetts. MIT Press.

Fetzer, James (1998). *People are not computers:(most) thought processes are not computational procedures*. J. Expt. Theor. Artif. Intell., 10, 371-391.

Feynman, Richard (1998). *Lectures on Computing*. Penguin

George, F. H. (1962). *Minds Machines and Gödel: Another Reply*. Journal of the Royal Institute of Philosophy, 37.

Harnad Stevan (1995). *What is computation (and is cognition that)?* Minds and Machines, 4, 377-378.

Harnish, Robert M. (2002). *Minds, Brains, Computers*. Malden. Blackwell.

Haugeland, John (1981). *Semantic Engines: An Introduction to Mind Design*. in Cummins, Robert and Denise Dellarosa Cummins (Ed.). *Minds, Brains, and Computers: The Foundations of Cognitive Science*. Oxford. Blackwell.

Haugeland, John (1986). *Artificial Intelligence: The Very Idea*. Cambridge, Massachusetts. MIT Press.

Haugeland, John (1997). *What is Mind Design?* Cambridge, Massachusetts. MIT Press.

Haugeland, John (1998). *Having Thought*. Cambridge, Massachusetts. Harvard UP.

Haykin, Simon (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Hobbes, Thomas (1651). *Leviathan*. Everyman's Library (1914).

Hodges, Andrew (1992). *Alan Turing, The Enigma*. London. Vintage.

Hoffmann, Achim (1998). *Paradigms of Artificial Intelligence*. Singapore.Springer.

Hofstadter, Douglas (1980). *Gödel, Escher, Bach: An Eternal Golden Braid*. Penguin.

Kearns, John T. (1997). *Thinking Machines: Some Fundamental Confusions*. Minds and Machines, 7, 269-287.

Kurzweil, Raymond (1990). *The Age of Intelligent Machines*. MIT Press.

Lucas J. R. (1964). *Minds, Machines and Gödel*. in Anderson Alan R. (Ed.). *Minds and Machines*. (pp. 43-59). New Jersey. Prentice-Hall.

Lucas J. R. (1970). *The Freedom of the Will*. Oxford. Clarendon Press.

McCord N. M., Illingworth W. (1991). *A Practical Guide to Neural Nets*. Addison-Wesley.

McDermott, Drew V. (2001). *Mind and Mechanism*. Cambridge, Massachuetts. MIT Press.

Minsky, M. L. and Papert S. (1969). *Perceptrons: An Introduction to Computational Geometry*. Cambridge, Massachusetts. MIT Press.

Newell, Allan (1990). *Unified Theories of Cognition*. Cambridge Massachusetts. Harvard UP.

Newell, A. and Simon H. (1997). *Computer Science as Empirical Inquiry: Symbols and Search*. in Haugeland, John (Ed.). *Mind Design II*. (pp. 81-110). Cambridge, Massachusetts. Bradford Book, MIT Press.

Penrose Roger (1990). *The Emperor's New Mind*. London. Vintage.

Putnam, Hilary (1975). *Mind, Language and Reality*. Cambridge. Cambridge UP.

Putnam, Hilary (1988). *Representation and Reality*. Cambridge, Massachusetts. MIT Press.

Putnam, Hilary (1990). *The Nature of Mental States*. in Lycan, William (Ed.). *Mind and Cognition*. Oxford. Blackwell.

Pylyshyn, Zenon (1984). *Computation and Cognition: Toward a Foundation of Cognitive Science*. Cambridge, Massachusetts. MIT Press.

Rapaport William J. (1998). *How Minds can be computational systems*. J. Expt. Theor. Artif. Intell, 10, 403-419.

Rosenblatt, F. (1958). *The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain*. in Cummins, R. and Dellarosa Cummins, D. (Ed.). *Minds, Brains, and Computers: The Foundations of Cognitive Science*. Blackwell.

Rumelhart, D. E., Hinton, G. E., Williams R.J. (1986). *Learning internal representation by error propagation*. in Rumelhart, D. E., McClelland J. L. (Ed.). *Parallel Distributed Processing*. Cambridge, Massachusetts. MIT Press.

Russel, Bertrand (1946). *History of Western Philosophy*. New York. Routledge.

Russell, Stuart J. and Peter Norvig (1995). *Artificial Intelligence - A Modern Approach*. Upper Saddle River, New Jersey. Prentice Hall.

Schank, Roger C. (1990). *What is AI anyway?* in Partridge, Derek and Wilks Yorrick (Ed.). *The Foundations of Artificial Intelligence: A Sourcebook*. Cambridge. Cambridge UP.

Searle, John (1980). *Minds, Brains, and Programs*. in Cummins, R. and Dellarosa Cummins, D. (Ed.). *Minds, Brains, and Computers: The Foundations of Cognitive Science*. Blackwell.

Searle, John (1990). *Is the Brain a Digital Computer?* Proc. & Addr. of the American Philosphical Association, 64, 21-37. (b)

Searle, John (1990). *Is the Brain's Mind a Computer Program?* Scientific American, January, 20-25. (a)

Singer, Charles (1959). *A short History of Scientific Ideas*. Oxford. Oxford UP.

Sipser, M. (1997). *Introduction to the Theory of Computation*. Boston. PWS Publishing.

Slezak Peter (1982). *Gödel's Theorem and the Mind*. Brit. J. Phil. Sci., 33, 41-52.

Sloman Aaron (1996). *Beyond Turing Equivalence*. in Millican, Peter and Clark, Andy (Ed.). *Machines and Thought*. (pp. 179-219). Oxford. Oxford UP.

Sloman Aaron (1998). *Diagrams in the Mind ?* www.cs.bham.ac.uk/~axs.

Sloman Aaron (2002). *The Irrelevance of Turing Machines to AI*. www.cs.bham.ac.uk/research/cogaff/0-index00-05.html.

Sloman, Aaron (1978). *The Computer Revolution in Philosophy*. Hassocks Essex. Harvester Press.

Smolensky, Paul (1990). *Connectionism and the foundations of AI*. in Partridge, Derek and Wilks Yorrick (Ed.). *The Foundations of Artificial Intelligence: A Sourcebook*. Cambridge. Cambridge UP.

Smolensky, Paul (1997). *Connectionist Modeling: Neural Computation/Mental Connections*. in Haugeland, John (Ed.). *Mind Design II*. Cambridge, Massachusetts. Bradford Book, MIT Press.

Turing, Alan (1937). *On computable numbers, with an application to the Entscheidungsproblem*. in Davies M. (Ed.). *The Undecidable*. Raven Press.

Turing, Alan (1947). *Lecture to the London Mathematical Society*. in Ince, D. C. (Ed.). *Mechanical Intelligence*. Amsterdam. Elsevier.

Turing, Alan (1948). *Intelligent Machinery*. in Ince, D. C. (Ed.). *Mechanical Intelligence*. (pp. 107-127). Amsterdam. Elsevier.

Turing, Alan (1950). *Computing Machinery and Intelligence*. Mind, LIX, 433-460.

Van Gelder (1991). *A Survey of the Concept of Distribution*. in Ramsey, B.; Stich, S. ; Rumelhart, D (Ed). *Philosophy and Connectionist Theory*. Lawrence Erlbaum.

Wegner, Peter (1997). *Why interaction is more powerful than algorithms*. Communications of the ACM, 40, 80-91.

Weizenbaum, Joseph (1976). *Computer Power and Human Reason*. San Francisco. Freeman & Co.

Wells, Andrew (1996). *Situated Action, Symbol Systems and Universal Computation*. Minds and Machines, 6, 33-46.

Whiteley C. H. (1962). *Minds, Machine and Gödel: A Reply to Mr Lucas*. The Journal of the Royal Institute of Philosophy, XXXVII, 61-62.

Winograd, T. (1990). *Thinking machines: Can there be? Are we?* in Partridge, Derek and Wilks Yorrick (Ed.). *The Foundations of Artificial Intelligence: A Sourcebook*. Cambridge. Cambridge UP.

Winograd, T. and F. Flores (1986). *Understanding Computers and Cognition: A New Foundation for Design*. Norwood. Ablex Publishing.