The Problem of Implementation and its Relation to the Philosophy of Cognitive Science

Henri Kauhanen

Master's Thesis
Cognitive Science
Institute of Behavioural Sciences
Faculty of Behavioural Sciences
University of Helsinki
November 2010
Instructor: Otto Lappi

HELSINGIN YLIOPISTO - HELSINGFORS UNIVERSITET - UNIVERSITY OF HELSINKI

Tiedekunta - Fakultet - Faculty	Laitos - Institution – Department
Faculty of Behavioural Sciences	Institute of Behavioural Sciences

Tekijä - Författare - Author
Henri Kauhanen

Työn nimi - Arbetets titel - Title
The Problem of Implementation and its Relation to the Philosophy of Cognitive Science

Oppiaine - Läroämne - Subject
Cognitive Science

Työn laji ja ohjaaja(t) - Arbetets art och handledare – Level and instructor	Aika - Datum - Month and year	Sivumäärä - Sidoantal - Number of pages
Master's Thesis; instructor Otto Lappi	November 2010	69

Tiivistelmä - Referat - Abstract
According to certain arguments, computation is observer-relative either in the sense that many physical systems implement many computations (Hilary Putnam), or in the sense that almost all physical systems implement all computations (John Searle). If sound, these arguments have a potentially devastating consequence for the computational theory of mind: if arbitrary physical systems can be seen to implement arbitrary computations, the notion of computation seems to lose all explanatory power as far as brains and minds are concerned.

David Chalmers and B. Jack Copeland have attempted to counter these relativist arguments by placing certain constraints on the definition of implementation. In this thesis, I examine their proposals and find both wanting in some respects. During the course of this examination, I give a formal definition of the class of 'combinatorial-state automata', upon which Chalmers's account of implementation is based. I show that this definition implies two theorems (one an observation due to Curtis Brown) concerning the computational power of combinatorial-state automata, theorems which speak against founding the theory of implementation upon this formalism.

Toward the end of the thesis, I sketch a definition of the implementation of Turing machines in dynamical systems, and offer this as an alternative to Chalmers's and Copeland's accounts of implementation. I demonstrate that the definition does not imply Searle's claim for the universal implementation of computations. However, the definition may support claims that are weaker than Searle's, yet still troubling to the computationalist. There remains a kernel of relativity in implementation at any rate, since the interpretation of physical systems seems itself to be an observer-relative matter, to some degree at least. This observation helps clarify the role the notion of computation can play in cognitive science. Specifically, I will argue that the notion should be conceived as an instrumental rather than as a fundamental or foundational one.

Avainsanat – Nyckelord - Keywords
implementation, computation, automata, dynamical systems, philosophy of cognitive science

Säilytyspaikka - Förvaringsställe - Where deposited
Library of the Faculty of Behavioural Sciences

Muita tietoja - Övriga uppgifter - Additional information

HELSINGIN YLIOPISTO - HELSINGFORS UNIVERSITET - UNIVERSITY OF HELSINKI

Tiedekunta - Fakultet - Faculty	Laitos - Institution – Department
Käyttäytymistieteellinen	Käyttäytymistieteiden laitos

Tekijä - Författare - Author
Henri Kauhanen

Työn nimi - Arbetets titel - Title
The Problem of Implementation and its Relation to the Philosophy of Cognitive Science

Oppiaine - Läroämne - Subject
kognitiotiede

Työn laji ja ohjaaja(t) - Arbetets art och handledare – Level and instructor	Aika - Datum - Month and year	Sivumäärä - Sidoantal - Number of pages
pro gradu -tutkielma; ohjaaja Otto Lappi	marraskuu 2010	69

Tiivistelmä - Referat - Abstract
Eräiden argumenttien mukaan laskenta eli komputaatio on havaitsijarelatiivista siinä mielessä, että monien fysikaalisten systeemien voidaan nähdä implementoivan useita komputaatioita (Hilary Putnam), tai että miltei kaikkien fysikaalisten systeemien voidaan nähdä implementoivan kaikki komputaatiot (John Searle). Sikäli kuin nämä argumentit ovat pitäviä, niillä voi olla kohtalokkaita seurauksia komputationaaliselle mielen teorialle. Jos mielivaltaiset fysikaaliset systeemit implementoivat mielivaltaisia komputaatioita, komputaation käsite näyttää kadottavan kaiken selittävän voimansa mielen ja aivojen toiminnan selittämisen yhteydessä.

David Chalmers ja B. Jack Copeland ovat yrittäneet vastata edellä mainittuihin relativistisiin argumentteihin asettamalla eräitä rajoitteita implementaatiorelaation määritelmälle. Tässä tutkielmassa tarkastelen heidän ehdotuksiaan ja esitän, että molempiin liittyy joitakin ongelmia. Tarkastelun myötä annan täsmällisen määritelmän ”kombinatoristen tilojen automaattien” luokalle, jolle Chalmersin implementaation teoria perustuu. Osoitan, että määritelmästä seuraa kaksi tulosta (joista toinen on Curtis Brownin aiemmin esittämä huomautus) liittyen kombinatoristen tilojen automaattien laskennalliseen voimaan. Argumentoin, että näiden tulosten vuoksi implementaation teoriaa ei tulisi perustaa kombinatoristen tilojen automaattien formalismille.

Tutkielman loppua kohden esitän vaihtoehtoisen implementaation analyysin, joka perustuu määritelmälle Turingin koneiden implementoitumisesta dynaamisissa systeemeissä. Näytän, että ehdottamani määritelmä ei implikoi Searlen universaalin implementaation väitettä. Tästä huolimatta on mahdollista, että määritelmästä seuraa tuloksia, jotka ovat Searlen väitettä heikompia mutta silti ongelmallisia komputationalismin kannalta. Implementaation teorian ytimessä näyttää joka tapauksessa säilyvän tietty relatiivisuus, sillä fysikaalisten systeemien tulkinta näyttäisi itsessään olevan jossakin määrin havaitsijarelatiivista. Tämä huomio auttaa selvittämään sitä roolia, joka komputaation käsitteellä on kognitiotieteessä. Erityisesti esitän, että käsite tulisi ymmärtää instrumentaalisena, ei fundamentaalisena tai perustan antavana käsitteenä.

Avainsanat – Nyckelord - Keywords
implementaatio, komputaatio, automaatit, dynaamiset systeemit, kognitiotieteen filosofia

Säilytyspaikka - Förvaringsställe - Where deposited
Käyttäytymistieteellisen tiedekunnan kirjasto

Muita tietoja - Övriga uppgifter - Additional information

Acknowledgements

This thesis took rather a long time to finish—such a long time, in fact, that its subject matter changed a number of times and its completion remained at all times far from certain. Now that it is completed, it is time to thank those persons, or, to be more inclusive, those entities without whose numerous contributions the thesis would in all likelihood have remained but a dim prospect.

First of all, my thanks go to Otto Lappi, my instructor, mentor, and occasional philosophical opponent, for kindly supervising the writing of this thesis and for providing valuable comments on a number of drafts. I am sure I will have tried his patience with my all-too-frequent hesitation and wavering; for his support and spurring I am most grateful.

On a more personal note, I thank my friends and fellow students, the following ones in particular, though in no particular order, for furnishing the world with meaning: Johannes, Juho, Jussi, Ville (the logician), Ville (the epistemologist), K. K., and Pipsa.

On a more metaphysical note, I would like to thank my mistress, the (now deceased) Department of English, for imbuing my life with humanist ideals to whose existence I had been largely oblivious. But I apologise to her, too, for it is true that seo hlæfdige bint þone cnapan, and I fell, or at any rate have so far fallen, far too short of fulfilling my task as her servant.

Most importantly, though, I thank my parents for having let me feel what I have wished and say what I have felt, and my brother for his camaraderie.

Helsinki, 1st November 2010

Contents

1

Introduction

Computationalism: ontological & instrumental. The physical symbol system hypothesis. Implementation. Synopsis.

According to one basic tenet of classical cognitive science, cognition is computation, in one sense or another. This thesis of *computationalism*, or the computational theory of mind, comes in many flavours. Common to all these views is the idea that there is a class of natural systems to which the predicate 'cognitive' may be applied, and that in this application the notion of computation is essential. Usually, the class is taken to consist of human beings, some animals, and perhaps certain artefacts, such as sufficiently complex robots.

Where the views differ is the precise way in which computation is thought essential to the ascription of cognitive states to a natural system. According to the most extreme view, advocated by Chalmers (1994b, 397–398) for instance, the aforementioned systems have minds *in virtue of* their implementing certain computations. Weakening down this claim a bit, we arrive at a position according to which those systems have cognitive (but not necessarily other mental) states by virtue of their implementing certain computations. Both of these views may properly be called *ontological*: they make an ontological statement, a statement about how things stand in the world as far as cognitive phenomena are concerned. In other words, they are both forms of the "thesis of computational sufficiency" (Chalmers 1994a, 2): implementing a computation is considered sufficient for the possession of cognitive states or even for the possession of subjective qualitative features of conscious experience, or qualia.

In contrast to such views there stands a more modest claim, according to which a computational description may be applied, or to some extent has to be applied, to systems that are cognitive. According to proponents of this claim, such as Marr (1977; 1982, 19–31), the notion of computation will figure, as one component, in the explanation of any cognitive phenomenon. However, the thesis of computational sufficiency is not—or need not be—endorsed. Merely implementing a computation is not considered sufficient for the possession of the cognitive process that the computation in question explains—more may be required of the system, such as a suitable neurophysiological make-up. Theses of this kind I shall call *instrumental*—they are roughly equivalent to what Chalmers (1994a, 2) calls the "thesis of computational explanation".

One well-known computational claim, falling perhaps closer to the ontological end of the computationalist spectrum, is the physical symbol system hypothesis of Newell and Simon (1976). According to this hypothesis, a system has the "means for general intelligent action" if, and only if, it is a "physical symbol system" (Newell & Simon 1976, 116). Being a physical symbol system, in turn, amounts to implementing certain computations. Precisely speaking, a physical symbol system is a system, obeying the laws of physics, that consists of symbol structures and processes operating on those structures. Symbols are "physical patterns" out of which more complex symbol structures may be constructed; the processes produce from symbols and symbol structures new symbols and structures. The symbol structures may refer to objects in the world and even to symbol structures within the system itself. In a word, a "physical symbol system is a machine that produces through time an evolving collection of symbol structures" (Newell & Simon 1976, 116).

Judging by the above descriptions, the notion of physical symbol system has not been too rigorously defined by Newell and Simon (but see Newell 1980). Moreover, although all of this is meant to connect nicely with physics, the details of this connection are never actually supplied. Perhaps it was

consideration of just such shortcomings that led Searle to pronounce that

> another stylistic feature of this literature is the haste and sometimes even carelessness with which the foundational questions are glossed over. What exactly are the anatomical and physiological features of brains that are being discussed? What exactly is a digital computer? And how are the answers to these two questions supposed to connect? (Searle 1992, 204–205.)

However, Newell and Simon (1976, 116) did remark that the notion of physical symbol system "bears a strong family resemblance to all general purpose computers", and Newell (1980) in fact went on to *define* physical symbol systems as general-purpose machines. In that case, the rather imprecise notion of physical symbol system may be dispensed with in favour of the notion of an implemented general-purpose (i.e., Turing-equivalent, see Chapter 2, below) computational structure. This move has decided advantages, since the notion of computational structure is rigorously defined and well understood within the mathematical theory of computation. Because of this, the notion of an implemented computational structure can be considered an explicatum ("that-which-explicates"; see Carnap 1958, 2) of the concept of physical symbol system. The physical symbol system hypothesis may then be reformulated as follows.

Hypothesis 1.1. *A system has the means for general intelligent action if, and only if, it implements a Turing-equivalent computational structure.*

Alternatively, the strict demand on Turing-equivalence may be waived, in which case we get the following formulation.

Hypothesis 1.2. *There is a class of computational structures,* **Cog***, such that a system has the means for general intelligent action if, and only if, it implements some structure from class* **Cog***. If a system implements a computational structure from class* **Cog***, the system is called a* cognitive *system.*

It is clear that the latter hypothesis explicates the notion of an ontological computationalist thesis discussed above.

The move from physical symbol systems to implemented computational structures leaves us with the notion of implementation, demanding explication in its own right. Computations are specified at an abstract level—computational structures, such as Turing machines, are mathematical objects existing *per se* in only the sense that mathematical objects exist. These structures are realised by being implemented in physical systems—but what, exactly, does it mean for a computational structure to be implemented in a physical system? Under what conditions are we allowed to say that a given physical system implements some computational structure, hence computes?

This question of the implementation of computation went unexamined for quite some time: it was tackled in a systematic manner for the first time by Chalmers (1994a). In light of the computational theory of mind, the late arrival of a theory of implementation is not surprising: a classical cognitive scientist simply does not need to bother with the problem of implementation, usually. If the mind is to the brain as the program is to the computer hardware, it seems clear that the study of the mind is to a considerable extent independent of the study of the brain—and the connection between the program and the implementing medium is as uninteresting as are the details of the medium itself. Seated in this functionalist ivory tower, the cognitive scientist need not bother with the brain any more than the computer scientist needs to bother with transistors or silicon chips (cf. Putnam 1960/1975, 372–373).

The question of the implementation of computation was not seriously posed until critics of computationalism brought forward certain arguments according to which very many physical systems implement very many computations. Such arguments have been presented by Putnam (1988, 121–125) and, most notably, Searle (1992, ch. 9).[1] According to Searle (1992, ch. 9), any sufficiently

[1]For reasons of historical accuracy, it should be noted that Searle presented his argument in Searle (1990) already. In what follows I shall adopt the practice of citing Searle (1992,

large physical object may be seen to implement any computation of whatever complexity. Thus, for example, the walls in Searle's office implement his word-processing program (Searle 1992, 208–209). Since the physical description of an object underdetermines its computational description in this way, computation is deemed *observer-relative* (Searle 1992, 211).

This argument, if sound, may have devastating consequences for computationalist claims of the ontological kind. If arbitrary physical systems may be seen to implement arbitrarily complex computations, it seems problematic to assert that a system is cognitive (let alone that it has a mind, with a subjective point of view, a conscious experience, and so on) *in virtue of* its implementing certain computations, for fear of panpsychism. The argument may strike a blow against instrumental computationalism as well, since it effectively broadens the domain of cognitive explanation to cover the whole of natural systems, thus possibly trivialising the notion of cognitive explanation. The inevitable conclusion, according to Searle (1992, ch. 9), is that computation cannot serve a foundational role in cognitive science. Putnam's (1988, 121–125) argument is similar in nature, although it does not afford quite as dramatic a conclusion.

Naturally, the arguments of Putnam (1988, 121–125) and Searle (1992, ch. 9) have not been spared from criticism. Here the trend has been to argue that the notion of implementation, implicit in the arguments, is ill-conceived. Thus, according to Chalmers (1994a; 1994b; 1996)[2] and Copeland (1996), computationalism can be rescued if a realistic definition of the implementation of computation is given.

In this thesis, after presenting some mathematical preliminaries and both Searle's (1992, ch. 9) and Putnam's (1988, 121–125) arguments (Chapters 2 & 3), I examine Chalmers's (1994b) and Copeland's (1996) suggestions in

ch. 9) consistently. The presentations in Searle (1990) and Searle (1992, ch. 9) are for all practical concerns identical.

[2]As with Searle, Chalmers's argument appears in more than one source, all identical in content as far as the relevant definitions are concerned. In the following I shall usually cite Chalmers (1994b).

detail (Chapters 4 & 5). My goal is to show that both of these accounts of implementation—while important milestones on the road toward a general theory of implementation—are in some respects inadequate. Although Chalmers (1994b) has been able to give a definition of the implementation of finite automata, his extension of this definition to the case of so-called combinatorial-state automata is unsuccessful. Moreover, I shall argue that Chalmers's (1994b) notion of physical system is too vague to support a rigorous theory of implementation. The problems with Copeland's (1996) account are of a similar nature: they have to do mainly with the fact that several of the concepts upon which the theory is built are inadequately precise.

In Chapter 6, I sketch an alternative account of implementation that attempts to avoid these shortcomings. I propose a definition for the implementation of Turing machines in dynamical systems and point out that the definition is strong enough to block Searle's (1992, ch. 9) relativistic claim. There remains a kernel of relativity in implementation nonetheless, since physical systems may not admit of a canonical interpretation, as will be seen in Chapter 7. The conclusion is, firstly, that the implications of implementational relativity for ontological computationalism are unclear, so that the former cannot be used to decide the fate of the latter, and secondly, that instrumental computationalism remains unaffected by Putnam's (1988, 121–125) and Searle's (1992, ch. 9) critiques. These observations motivate a closer examination of the role of computation as an explanatory notion in cognitive science. I will argue that the notion has much value if construed as an instrumental concept, but that there is little point in construing it as a foundational notion (as construed by Chalmers 1994a, for instance).

2

Mathematical Preliminaries

Set-theoretical notation. Groups & actions. The finite transducer. The Turing machine. The Church–Turing Thesis.

The technical nature of the issue at hand necessitates a mathematical mode of presentation. Due to restrictions on space, it will not be possible to review the necessary mathematics here very thoroughly, however. Accordingly, I shall assume throughout the thesis that the reader is acquainted with basic concepts of and results in naïve set theory, first-order predicate logic, abstract algebra, metric topology, computability theory, and the theory of automata and formal languages. In this brief chapter only the most crucial definitions, as well as some notational conventions and two important lemmas, are introduced.

As regards notation, the set of natural numbers is written $\mathbb{N}$ and is taken to include zero; $\mathbb{Z}$ is the set of integers. The set of positive integers will be denoted by the symbol $\mathbb{Z}_+$; i.e., $\mathbb{Z}_+ = \{1, 2, 3, \ldots\}$. For every $n \in \mathbb{Z}_+$, we define the set J_n by putting $J_n = \{z \in \mathbb{Z}_+ \mid z \leq n\}$. The set of real numbers is $\mathbb{R}$; real intervals are referred to using brackets, e.g., $[0, 1[= \{x \in \mathbb{R} \mid 0 \leq x < 1\}$. For ordered pairs and tuples in general, ordinary parentheses are used. The cardinality of set A is denoted by $|A|$; thus, $|\{0, 1\}| = 2$, for example.

Functions, or mappings, are treated extensionally, as subsets of Cartesian products. If $f : A \to B$ is a function and $X \subseteq A$, then the symbol $f(X)$ stands for the image of X in the function f; i.e.,

$$f(X) = \{b \in B \mid b = f(x) \text{ for some } x \in X\}.$$

Set A is *countable* if there exists a one-to-one onto mapping (a bijection) $f : A \to N$ for some $N \subseteq \mathbb{N}$; otherwise, A is *uncountable*. If there exists a one-to-one onto mapping $f : A \to \mathbb{N}$, then A is said to be *countably infinite*. For all $n \in \mathbb{Z}_+$, the symbol A^n refers to the n-dimensional Cartesian product of A. In other words,

$$A^n = \{(a_1, \ldots, a_n) \mid a_i \in A \text{ for all } i \in J_n\}.$$

We also stipulate that $A^0 = \{\varnothing\}$.

The symbol A^ω denotes the countably infinite Cartesian product of A. This is defined as the set

$$A^\omega = \{f \mid f : \mathbb{N} \to A \text{ is a function and } f(n) \in A \text{ for all } n \in \mathbb{N}\}.^1$$

An element of A^ω (an ω-tuple) is thus a function. In what follows, such elements are loosely referred to using the notation

$$(f(0), f(1), f(2), \ldots).$$

This notation will be found useful for such a function f whose values are constant from some $n \in \mathbb{N}$ onwards.

The following two lemmas will be of use later on. The first one is too straightforward to require demonstration. A proof of the second one is given in Appendix A.

Lemma 2.1. *Let $n \in \mathbb{N}$ and let A be a finite set. Then the Cartesian product A^n is finite as well.*

Lemma 2.2. *Let A be a set, finite or infinite. Then A has at least two elements if and only if the countably infinite Cartesian product A^ω is infinite.*

Let A and M be sets and let $f : M \to M$ and $* : A \times A \to A$ be mappings; let us write $a * b$ for the image $*((a, b))$ of $(a, b) \in A \times A$. If f is one-to-one

[1] For a motivation for this rather cumbersome definition, see, e.g., Abian (1965, 161).

and onto, it is called a *transformation* of the set M. The pair $(A, *)$ is a *group* if the following three conditions are met:

(G1) for all $a, b, c \in A$, $a * (b * c) = (a * b) * c$,

(G2) there exists an $e \in A$ such that for all $a \in A$, $e * a = a * e = a$,

(G3) for each $a \in A$ there exists an $a^{-1} \in A$ such that $a * a^{-1} = a^{-1} * a = e$.

A *linearly ordered group* is a triple $(A, *, <)$, where $(A, *)$ is a group and $<$ is a linear order on A (i.e., $< \subseteq A \times A$ is such a relation that for all $a, b \in A$, exactly one of the conditions $a < b$, $b < a$, and $a = b$ holds). An *action* of the group $(A, *)$ on the set M is a set of mappings $\{f_a : M \to M \mid a \in A\}$, where each f_a is a transformation of the set M satisfying the following conditions:

(A1) for all $a, b \in A$, $f_{a*b} = f_a \circ f_b$ (where $\circ$ signifies functional composition),

(A2) for each $a \in A$, $f_{a^{-1}} = f_a^{-1}$ (where f^{-1} is the inverse of f).

A *partition* of a set A is any collection $\mathcal{B} = \{B \mid B \subseteq A\}$ of subsets of A such that

(P1) $B \neq \varnothing$ for all $B \in \mathcal{B}$,

(P2) $B \cap C = \varnothing$ for all $B, C \in \mathcal{B}$ with $B \neq C$,

(P3) $\bigcup_{B \in \mathcal{B}} B = A$.

It follows from these properties that if $x \in A$, then $x \in B$ for exactly one $B \in \mathcal{B}$. This set B is called the *equivalence class* of x in the partition $\mathcal{B}$; it is denoted by $[x]_{\mathcal{B}}$.

An *alphabet* is any finite, non-empty set. The elements of an alphabet are called *symbols*. A *string* is an ordered tuple of symbols drawn from an alphabet. Precisely speaking, if Σ is an alphabet and $n \in \mathbb{Z}_+$, then $(a_1, \ldots, a_n)$ is a string if and only if $a_i \in \Sigma$ for all $i \in J_n$. The more concise notation $a_1 \ldots a_n$ is usually used for strings. Strings can be *concatenated*: if $u = a_1 \ldots a_n$ and $w = b_1 \ldots b_m$ are strings, then uw is defined as the string $a_1 \ldots a_n b_1 \ldots b_m$. If a

is a symbol and $n \in \mathbb{Z}_+$, the notation a^n is used to signify the string obtained by concatenating a $n-1$ times to itself; in other words, $a^n = a_1 \dots a_n$, where $a_i = a$ for all $i \in J_n$. There is a special string called the *empty string*; it is denoted by ϵ and defined by setting $\epsilon = \varnothing$. This can also be thought to represent the concept of 'empty symbol'. Also, for every symbol a of any alphabet we stipulate that $a^0 = \epsilon$. The set of strings it is possible to draw from an alphabet Σ is denoted by Σ^*; this set is defined as $\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^i$ (note that $\Sigma^0 = \{\varnothing\}$ so that $\epsilon \in \Sigma^0$ and consequently $\epsilon \in \Sigma^*$). A *language* is any subset $L \subseteq \Sigma^*$.

Throughout the thesis, the term *computational structure* is taken to refer to any structure defined in the theory of computation (computability theory).[2] Important examples are the finite transducer and the Turing machine.

Definition 2.1. *A* finite transducer *is a quintuple* $(Q, \Sigma, \Gamma, \delta, q_0)$*, where*

(i) Q *is a finite, non-empty set (called the set of states),*

(ii) Σ *is an alphabet (the input alphabet),*

(iii) Γ *is an alphabet (the output alphabet),*

(iv) $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q \times (\Gamma \cup \{\epsilon\})$ *is a mapping (the transition function),*

(v) $q_0 \in Q$ *(the initial state).*

Given a string $u \in \Sigma^*$ *as input, the finite transducer reads one symbol of* u *at a time, transits from state to state, and emits symbols from* $\Gamma \cup \{\epsilon\}$ *as output, as determined by its transition function* δ*. The emitted symbols form a string* $w \in \Gamma^*$*, not necessarily unique, so that the machine computes a relation in* $\Sigma^* \times \Gamma^*$*.*

This definition of the finite transducer differs somewhat from the more common Moore machine and Mealy machine studied in the theory of automata (see Hopcroft & Ullman 1979, 42–45), since empty input and output (so-called

[2] For an introduction to computability theory, see, e.g., Hopcroft and Ullman (1979).

"ϵ-moves") are allowed. This adjustment is done so that the automaton need neither always have input in order to behave nor produce output on all of its moves. It may be noted that the adjustment results in a kind of nondeterminism, since the automaton may follow more than one trace of state transitions for a given input string, and consequently may emit several different output strings for one input string.[3]

Definition 2.2. *A* Turing machine *is an abstract device with an infinitely extended 'tape', divided into countably infinitely many squares, together with a 'scanner' that moves along this tape, reading and writing symbols on it by certain rules. As input the machine is given a string u drawn from some alphabet Σ; u is placed on the tape so that each square of the tape is occupied by exactly one symbol. The machine has a finite non-empty set Q of states and an auxiliary alphabet Γ, which is disjoint from Σ. Also, Γ contains a distinguished symbol B called the blank symbol; initially (before u is placed on the tape of the machine), all the squares of the tape carry the blank symbol. The rules by which the machine performs operations on the tape are embodied in the transition function*

$$\delta : Q \times (\Sigma \cup \Gamma) \to Q \times (\Sigma \cup \Gamma \cup \{L, R\}),$$

which is a partial function undefined on a single state $h \in Q$ for all $a \in \Sigma \cup \Gamma$ and defined on all $q \in Q \setminus \{h\}$ for all $a \in \Sigma \cup \Gamma$. The elements L and R signify movement of the scanner one square left or right, respectively; the state h is called the halting state of the machine. An element $((q, a), (q', a')) \in \delta$ of the transition function is to be interpreted as follows: if in state q and reading

[3] For a simple illustration of this nondeterminism, consider the simple one-state transducer $(\{q_0\}, \{1\}, \{0, 1\}, \{((q_0, 1), (q_0, 0)), ((q_0, \epsilon), (q_0, 1))\}, q_0)$, shown below as a directed graph (an edge labelled x/y means that x is the symbol read and y the symbol emitted):

It is easily seen that this machine, if given as input the string 1, gives as output the language $\{1^m 01^n \mid m, n \in \mathbb{N}\}$. Therefore, if we denote the relation computed by this transducer by R, then $(1, w) \in R$ for all $w \in \{1^m 01^n \mid m, n \in \mathbb{N}\}$.

symbol a, transit to state q' and replace the symbol a with the symbol a' (in case $a' \notin \{L, R\}$) or move the scanner left or right (in case $a' \in \{L, R\}$).

Note the difference between the *blank* symbol B and the *empty* string or symbol ϵ. If an automaton prints B, it really prints something—a symbol that *represents* that the place the symbol occupies is to be considered 'empty'. If an automaton 'prints' ϵ, it really does not print anything at all.

A computational structure $\mathcal{C}$ is called *Turing-complete* if every Turing-computable function is $\mathcal{C}$-computable (i.e., is computable by $\mathcal{C}$).[4] $\mathcal{C}$ is *Turing-equivalent* if it is Turing-complete and if every $\mathcal{C}$-computable function is Turing-computable. If $\mathcal{C}$ is Turing-complete but not Turing-equivalent, it is called *hypercomputational* (for a survey of hypercomputational structures, see Copeland 2002). The classical view, implying that hypercomputational structures are not to be considered genuinely computational (or effective), is embodied in the *Church–Turing Thesis* (see Church 1936/1965, sect. 7; Turing 1936–7/1965, sect. 9):

Hypothesis 2.1 (Church–Turing Thesis)**.** *Any effective, mechanical procedure can be carried out by some Turing machine. Equivalently, any procedure of this kind can be recursively defined.*

[4] In the case of structures which compute relations instead of functions thanks to nondeterminism, we adopt the following convention: if $\mathcal{C}$ is such a (nondeterministic) structure and R is the relation computed by $\mathcal{C}$, then a function f is $\mathcal{C}$-computable if and only if $f \subseteq R$.

3

Arguments for Universal Implementation

Searle's Thesis & Putnam's Realisation Theorem. The meaning of these.

With his Chinese Room argument, Searle (1980) attempted to show that syntactic symbol manipulation (i.e., computation) does not suffice for the possession of mental states, thereby undermining the aims of what he calls "strong artificial intelligence" (in our terminology, a variant of ontological computationalism). According to this argument, the fact that some system implements some computation goes no way toward guaranteeing that the system has intentionality, understanding, and so on. Another way of expressing the matter is the dictum "semantics is not intrinsic to syntax" (Searle 1992, 210).

By contrast, Searle's (1992, ch. 9) newer argument aims to demonstrate that "syntax is not intrinsic to physics" (Searle 1992, 210). In other words, the physical description of a system *underdetermines* its computational description: for any system there exist multiple computational descriptions that are in principle compatible with its physical description. In yet other words, computation is "observer-relative" (Searle 1992, 211). There is no fact of the matter as to which computation a physical system implements; it will implement any computation we may wish it to implement, as long as it has sufficiently many discriminable parts.

Here is Searle's own statement of his claim:

> For any program and for any sufficiently complex object, there is some description of the object under which it is implementing the program.

> Thus for example the wall behind my back is right now implementing the Wordstar program, because there is some pattern of molecule movements that is isomorphic with the formal structure of Wordstar. But if the wall is implementing Wordstar, then if it is a big enough wall it is implementing any program, including any program implemented in the brain. (Searle 1992, 208–209.)

I shall henceforth refer to this claim as *Searle's Thesis*.

Hypothesis 3.1 (Searle's Thesis)**.** *For any program and for any sufficiently complex (physical) object, there is some description of the object under which it is implementing the program.*

Searle appears to think that his thesis is an analytical truth, given the way we have defined computation: "[i]f computation is defined in terms of the assignment of syntax, then everything would be a digital computer, because any object whatever could have syntactical ascriptions made to it", and "[t]he ascription of syntactical properties is always relative to an agent or observer" (Searle 1992, 207–208). The universality of the thesis is then claimed to follow from the immensely complex microstructure of physical systems. The assertion that any (sufficiently complex) object implements any program of arbitrary complexity seems, nevertheless, *prima facie* incredible—at least as long as we are not provided with a constructive method whereby to effect such an implementation or ascription of syntactical properties. Fortunately, Putnam (1988, 121–125) has attempted something of the kind, so let us next turn to his argument.[1]

In thermodynamics, the concept of *open system* refers to a system that can exchange both energy and mass with its surroundings. According to Putnam

[1]Later, in Chapter 5, I shall examine Copeland's (1996) analysis of Searle's Thesis, which does give a kind of method by which to implement arbitrary computations in arbitrary systems. In the course of that examination, it will also be seen that Searle's formulation of his thesis is not entirely unambiguous, a point which is important as far as the implications of Searle's Thesis for different varieties of computationalism are concerned. I shall defer discussion of this point until Chapter 5, taking Searle's formulation (Hypothesis 3.1) at face value for the moment.

(1988, 121–125), every ordinary open system implements every finite automaton without input and output. This implementation result is put forward as a theorem, and Putnam offers a detailed proof. The proof is based on a liberal grouping of the states of the open system under consideration so that the evolution of the system is seen to exactly mirror the state transitions of the automaton, and as it stands, the proof seems to be sound. The result has received much attention (Chalmers 1994a; 1994b; 1996; Chrisley 1994; Cocos 2002; Scheutz 1999; Shagrir 2005), so much so in fact that it has even been entitled "Putnam's Realization Theorem" (Scheutz 1999, 162).[2]

In its original formulation the theorem states that "[e]very ordinary open system is a realization of every abstract finite automaton" (Putnam 1988, 121). This formulation is unfortunate, however, since the theorem does not cover automata with input or output. A more honest formulation would be the following.

Theorem 3.1 (Putnam's Realisation Theorem)**.** *Every ordinary open system implements every finite automaton without input and output.*

Moreover, Putnam never defines explicitly the notion of 'finite automaton without input and output'. However, it is clear from his writing that he has the following in mind:

Definition 3.1. *A* finite automaton without input and output *is a triple* (Q, δ, q_0)*, where*

(i) Q *is a finite non-empty set (the set of states),*

(ii) $\delta : Q \to Q$ *is a mapping (the transition function),*

(iii) $q_0 \in Q$ *(the initial state).*

The astute reader will notice that this sort of structure is actually unknown to the theory of computation. I will have more to say about this later. But

[2]In line with standard functionalist vocabulary, Putnam (1988, 121–125) uses the term 'realisation' rather than 'implementation', but here the two terms are synonymous.

first, let us sketch a proof of Putnam's theorem to get a firmer grasp of what is being claimed.

The proof of the theorem relies on two physical principles, the "Principle of Continuity" and the "Principle of Noncyclical Behaviour" (Putnam 1988, 121). The former is needed to ensure that the state transitions of the open system are causal, the latter to guarantee the universality of the claim. According to the Principle of Continuity, "[t]he electromagnetic and gravitational fields are continuous, except possibly at a finite or denumerably infinite set of points" (Putnam 1988, 121); the principle is assumed to have the status of a physical law. The Principle of Noncyclical Behaviour, on the other hand, states that the gravitational and electromagnetic fields are noncyclical at each point of the boundary of the open system and at every point of the system sufficiently close to the boundary (Putnam 1988, 121). In particular, this implies that the system is in different maximal states at different times: that states are not repeated during the system's evolution. The principle is justified as follows:

> This principle will hold true of all systems that can 'see' (are not shielded from electromagnetic and gravitational signals from) a clock. Since there are natural clocks from which no ordinary open system is shielded, all such systems satisfy this principle. (Putnam 1988, 121.)

Letting $\mathcal{S}$ stand for an open system, the maximal state of the system at time t, written $m(t)$, is defined as "the value of all the field parameters at all the points inside the boundary of $\mathcal{S}$ at t" (Putnam 1988, 122). In other words, $m(t)$ is a complete specification of the values of all the relevant variables of the system $\mathcal{S}$ at time t. Maximal states are points in the phase space of $\mathcal{S}$, which will be denoted by the letter M. The symbol $B(t)$ will stand for the maximal state of the boundary of $\mathcal{S}$ at time t.

In proving the theorem itself, an auxiliary result will be needed:

Lemma 3.1. *For any time instant t, the maximal state $m(t)$ is the only maximal state of $\mathcal{S}$ compatible with $B(t)$.*

Given the Principle of Continuity, this result is intuitive—if the open system is required to be continuous at its boundary, then only one boundary state will be compatible with the system's inside once the system's maximal state has been fixed. Proving the lemma in a rigorous manner is, however, tedious, and the proof is therefore relegated to Appendix A.

With Lemma 3.1 in hand, the theorem itself can be shown as follows. (The following proof is, in effect, a notational variant of that given by Putnam 1988, 121–125.)

Proof of Theorem 3.1. Let $\mathcal{S}$ be an ordinary open system with phase space M; let $m(t)$ and $B(t)$ be the maximal states of the system and its boundary, respectively, at time t. Let $\mathcal{A} = (Q, \delta, q_0)$ be a finite automaton without input and output, where Q is a non-empty finite set of states, $q_0 \in Q$ is the initial state, and $\delta : Q \to Q$ is the transition function. Since our proof is constructive, we may without loss of generality take the set of states to be $Q = \{q_0, q_1\}$, and the transition function to be $\delta = \{(q_0, q_1), (q_1, q_0)\}$. The automaton we will consider is, in other words, a simple two-state oscillator.

We need to show that the system $\mathcal{S}$ implements the automaton $\mathcal{A}$—in other words, to show that $\mathcal{S}$ oscillates between two states over any desired real-time interval. Let $I \subseteq \mathbb{R}$ then be an arbitrary closed real interval. Suppose we wish to implement n state transitions of $\mathcal{A}$; we then partition I by letting

$$I = [t_1, t_2[\cup [t_2, t_3[\cup \ldots \cup [t_n, t_{n+1}[\cup \{t_{n+1}\},$$

where $t_1 < t_2 < \ldots < t_{n+1}$. Next, for each subinterval $[t_i, t_{i+1}[$ of I we define an *interval state* s_i by setting

$$s_i = \{m(t) \mid t \in [t_i, t_{i+1}[\}.$$

Thus, each s_i is a subset of M and the system $\mathcal{S}$ is in state s_i if and only if it is in one of the maximal states it assumes during the subinterval $[t_i, t_{i+1}[$.

It follows from the Principle of Noncyclical Behaviour that the interval states are distinct, or precisely speaking that $s_i \neq s_j$ for all $i, j \in J_n$ with $i \neq j$.

Let us finally define two states of the system $\mathcal{S}$, α and β, as follows:

$$\alpha = \bigcup\{s_i \mid i \in J_n \text{ and } i \text{ odd}\}, \text{ and}$$

$$\beta = \bigcup\{s_i \mid i \in J_n \text{ and } i \text{ even}\}.$$

This completes the construction. The system $\mathcal{S}$ now oscillates between two states α and β during the interval $[t_1, t_{n+1}[$, and the right endpoint can be easily included if desired. Moreover, state α corresponds to state q_0 of the automaton $\mathcal{A}$, while state β corresponds to automaton state q_1.[3]

It remains to be shown that the state transitions of the system are causal: that being in state α during the subinterval $[t_1, t_2[$ causes $\mathcal{S}$ to assume state β during $[t_2, t_3[$ (and similarly for the other transitions). For this, take any $t \in [t_1, t_2[$. By Lemma 3.1, $m(t)$ is then the only maximal state compatible with $B(t)$, the maximal state of the boundary of $\mathcal{S}$ at t. Therefore, given the information that the system $\mathcal{S}$ was in state α at time t, the boundary condition $B(t)$, and other laws of nature, the mathematically omniscient demon of Laplace can determine the subsequent evolution of $\mathcal{S}$—particularly, that it assumes state β over the next subinterval, $[t_2, t_3[$.

We have shown that the system $\mathcal{S}$ implements the automaton $\mathcal{A}$, and since the construction would apply to any finite automaton without input and output (the construction made no essential use of the particular set Q and function δ), we derive the theorem by universal generalisation. Q.E.D.

The first thing to note about Putnam's Realisation Theorem is that it holds for the peculiar class of finite automata without input and output only. In spite of this, several authors have placed much importance on the theorem. Here is

[3]Note that it is essential here that the interval states s_i are distinct. Otherwise $\mathcal{S}$ might transit through for instance the sequence of states $\alpha, \alpha, \beta, \alpha, \beta, \ldots$ instead of the desired sequence $\alpha, \beta, \alpha, \beta, \ldots$.

Chalmers's description of the situation:

> If this is right, a simple system such as a rock implements any automaton one might imagine. Together with the thesis of computational sufficiency, this would imply that a rock has a mind, and possesses many properties characteristic of human mentality. If Putnam's result is correct, then, we must either embrace an extreme form of panpsychism or reject the principle on which the hopes of artificial intelligence rest. (Chalmers 1996, 309–310.)

This remark is all the more surprising when one considers Putnam's own discussion of the theorem's domain of application:

> If a physical object does not have motor organs or sensors of the specified kind, then, of course, it cannot be a model of a description which refers to a kind of automaton which, *ex hypothesi*, possesses motor organs and sensors of that kind. And even if it does possess such 'inputs' and 'outputs', it may behave in a way which violates predictions which follow from the description (e.g., print two '1's in a row when it is a theorem that the machine with the given description never does this). So there is no hope that the theorem just proved will also hold, unchanged, for automata which have inputs and outputs which have been specified (or at least constrained) in physical terms. (Putnam 1988, 124.)

For the purposes of cognitive science, in particular, this means that Putnam's theorem—its mathematical sophistication notwithstanding—may not be terribly consequential as far as the various formulations of computationalism are concerned. One would not expect the class **Cog** (see Hypothesis 1.2) to include finite automata without input and output.

In spite of this, however, Putnam's Realisation Theorem does motivate a thorough analysis of the notion of implementation, since it may well be possible to restrict the theorem in some way to cover more interesting kinds of automata. Furthermore, the threat posed by Searle's (1992, ch. 9) more

general (albeit less precise) claim still exists. In fact, as we will see in Chapter 5, Copeland (1996) has been able to give a constructive version of Searle's Thesis, and his construction parallels that of Putnam (1988, 121–125) to some extent. There exists, then, a genuine theoretical motivation for a general theory of the implementation of computation, and it is to the attempts to formulate such a theory, and to their criticism, that we now turn.

4

Implementation: Combinatorial-State Automata

Implementation as state-to-state correspondence. Limitations of finite automata. The combinatorial-state automaton. Limitations of the latter. Critique of Chalmers's theory of implementation.

Both Searle's Thesis and Putnam's Realisation Theorem indicate that if the notion of implementation is not fixed in place in a restrictive enough manner, counter-intuitive or otherwise unwanted consequences follow. If we do not wish every open system to implement every finite-state automaton (without input and output), we need to flesh out our concept of implementation, or realisation, so that unwanted instances are exempted. In short, we need to *define* under which conditions a physical system can be said to implement a computational structure.

According to Chalmers (1994b, 392), a "physical system implements a given computation when the causal structure of the physical system mirrors the formal structure of the computation". Elaborating on this further, he writes:

> A physical system implements a given computation when there exists a grouping of physical states of the system into state-types and a one-to-one mapping from formal states of the computation to physical state-types, such that formal states related by an abstract state-transition relation are mapped onto physical state-types related by a corresponding causal state-transition relation. (Chalmers 1994b, 392.)

When this idea is explicated in the formalism of finite transducers, we have the following definition (the notation used here differs somewhat from that used

by Chalmers 1994b; a difference in notation, naturally, implies no difference in content):

Definition 4.1 (Chalmers 1994b, 392–393)**.** *Let* $\mathcal{S}$ *be a physical system, and let* M *be the set of physical state-types of* $\mathcal{S}$*,* I *the set of inputs to* $\mathcal{S}$*, and* O *the set of outputs of* $\mathcal{S}$*. We assume the sets* I *and* O *may contain 'empty' input and output, that is to say, those situations in which* $\mathcal{S}$ *is not given input or does not produce output. Let* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0)$ *be a finite transducer. Then the system* $\mathcal{S}$ implements *the automaton* $\mathcal{A}$ *if and only if there is a one-to-one mapping* $\varphi : (Q \times (\Sigma \cup \{\epsilon\})) \times (Q \times (\Gamma \cup \{\epsilon\})) \rightarrow (M \times I) \times (M \times O)$ *(the implementation mapping) such that for every pair* $((q, a), (q', b)) \in \delta$*, the following conditional holds: if* $\mathcal{S}$ *is in state-type* $m \in M$ *and is receiving input* $i \in I$*, and* $\varphi((q, a), (q', b)) = ((m, i), (m', o))$ *for some* $m' \in M$ *and* $o \in O$*, this reliably causes* $\mathcal{S}$ *to enter state-type* m' *and produce output* o*.*

Even though the definition is a bit involved, one can see that it satisfies the central requirement discussed in the above quotation: the formal state transitions of the automaton are precisely mirrored by the causal transitions of the implementing system. What is more, according to Chalmers (1994b), the definition evades Searle's Thesis by requiring the causal transitions of the implementing system to be 'reliable'. It is not enough to just pick some states of the system and group them together conveniently to form state-types so that suitable causal connections may be established between the state-types (cf. the procedure used in the proof of Putnam's Realisation Theorem, pp. 17–18, above); rather, "the connection between connected states must be *reliable* or *lawful*, and not simply a matter of happenstance" (Chalmers 1996, 313). I will have more to say about this feature of the definition a little later.

Reflecting on this definition, Chalmers notes that

> the relation between an implemented computation and an implementing system is one of isomorphism between the formal structure of the former and the causal structure of the latter. In this way, we can see that as far

> as the theory of implementation is concerned, a computation is simply an *abstract specification of causal organization*. (Chalmers 1994b, 396.)

As Scheutz (2001) has pointed out, however, this is not strictly true—the mathematical definition of isomorphism between two structures (see, e.g., Väänänen 1987, 78) demands more than is spelled out by Chalmers's (1994b) definition of implementation. In a word, while Definition 4.1 does guarantee that every state transition of the implemented automaton is mirrored in the implementing system, it does not guarantee that every causal transition of the system is mirrored in the implemented automaton. According to Scheutz (2001), this leads to certain difficulties: under Definition 4.1 a physical system will implement certain computations that one would not naturally regard as being implemented. Accordingly, he has adjusted Chalmers's (1994b) definition so that full isomorphism between the computation and the implementing system is guaranteed, and has refined the analysis further with the concepts of 'characteristic automaton' and 'bisimulation'. I shall not pursue this line of development here, however, as I think there are other problems, to be spelled out in a moment, with the Chalmersian account of implementation.

In a certain sense, finite automata are an inappropriate formalism for cognitive science. Computationally speaking, finite automata are markedly inferior to for example Turing machines. For instance, it is well-known that no finite automaton accepts the language $\{a^n b^n \mid n \in \mathbb{Z}_+\}$,[1] although it is comparatively easy to design a Turing machine to do this, and although it is a trivial task for most humans. This fact is very relevant to cognitive science, since it is one of the reasons that led Chomsky (1956) to abandon finite automata as a model of natural language. It thus seems that we need a definition of the implementation of computational structures that exceed finite automata in their computational power.

In the case of the finite transducer it was fairly clear how a definition of im-

[1] This can be shown using the "pumping lemma" (Hopcroft & Ullman 1979, 55–56).

plementation might be attempted, and the analysis is easily adapted to other kinds of finite automata. This has to do with the simplicity of the computational structure in question and, more importantly, with the fact that the structure quite closely resembles the behaviour we usually ascribe to physical systems—the causal structure of the system mirrors the formal structure of the computation, as the slogan goes. But how would one go about defining implementation conditions for, say, Turing machines? Surely it is unreasonable to suggest that each physical system implementing some Turing machine should literally contain a tape with symbols on it and a scanner scuttling back and forth—if only for the reason that in the mathematical structure, the length of the tape is unbounded. In this case, a definition of implementation that relies on the notion of isomorphism seems, at least *prima facie*, not quite up to the task.

Chalmers (1994b) attempts to overcome this difficulty by introducing a new computational structure, one he calls the *combinatorial-state automaton*. The formalism of combinatorial-state automata is put forward as a general solution to the riddle of implementation: "[t]he theory of implementation for combinatorial-state automata provides a basis for the theory of implementation in general" (Chalmers 1994b, 396). This claim is based on the Turing-completeness of the combinatorial-state automaton formalism. Unfortunately, Chalmers's definition of the class of combinatorial-state automata (see Chalmers 1994b, 394) is not very rigorous. I shall presently formulate a rigorous definition of the automata in question, the upshot of this undertaking being that each combinatorial-state automaton is seen to be either a finite automaton with limited computational power, or an infinite automaton with hypercomputational power. I shall then argue that for this reason, combinatorial-state automata cannot provide a basis for the theory of implementation in general.

Let us then take a look at Chalmers's description of combinatorial-state

automata in order to flesh out the details. According to Chalmers (1994b, 394), these automata differ from finite automata in that their states, inputs, and outputs are not monadic labels, but tuples. In fact, Chalmers calls them vectors—however, he does not define the required operations of vector addition and scalar multiplication in the case under consideration (which is not surprising, for what should one take the sum of two states of a computational structure to mean?).[2] It seems that what Chalmers has in mind here is one special case of the concept of vector space, the Euclidean space $\mathbb{R}^n$. However, he never takes advantage of this assumption of three vector spaces with all the structure the vector space axioms guarantee them. Thus we may do away with the assumption and treat the states, inputs, and outputs simply as tuples.

Continuing with the description, then, we learn that the input and output tuples of a combinatorial-state automaton are finite in length, but the state tuples may be infinite (Chalmers 1994b, 394). However, the elements of the tuples are drawn, in each case, from a finite set. If the state tuples of a combinatorial-state automaton are finite in length, the automaton is said to be *finite*; otherwise, it is *infinite* (Chalmers 1994b, 394).

Formalising this a bit, we have first of all three finite sets, from among whose elements the tuples are drawn. That is, we have first of all a set of "substates" Q from which the state tuples are constructed. Letting $\widehat{Q}$ stand for the set of state tuples, we thus have that either $\widehat{Q} \subseteq Q^i$ for some $i \in \mathbb{Z}_+$ (in case the automaton in question is finite) or $\widehat{Q} \subseteq Q^\omega$ (if the automaton is infinite). The set of input tuples $\widehat{\Sigma}$ is similarly constructed from an alphabet Σ; thus, $\widehat{\Sigma} \subseteq \Sigma^j$ for some $j \in \mathbb{Z}_+$ (since input tuples are stipulated to be finite). The set of output tuples $\widehat{\Gamma}$ is defined analogously. Although Chalmers (1994b) never states it explicitly, it seems that the sets $\widehat{Q}$, $\widehat{\Sigma}$ and $\widehat{\Gamma}$ are assumed to be non-empty. Since nothing of interest arises from empty sets of states and empty alphabets, we may indeed assume these sets to be non-empty.

[2] For the rather elaborate definition of vector space, see, e.g., Burton (1967, 249–250).

Since the sets Q, Σ and Γ have been stipulated to be finite, the sets Q^i, Σ^i and Γ^i are also finite for any $i \in \mathbb{Z}_+$ by Lemma 2.1. Since $\widehat{Q} \subseteq Q^i$, $\widehat{\Sigma} \subseteq \Sigma^j$ and $\widehat{\Gamma} \subseteq \Gamma^k$ for some $i, j, k \in \mathbb{Z}_+$ if the automaton is finite, it follows that the sets $\widehat{Q}$, $\widehat{\Sigma}$ and $\widehat{\Gamma}$ are finite as well. Hence, in the case of a finite combinatorial-state automaton, the sets of state, input, and output tuples are all finite. However, if the automaton is infinite, then $\widehat{Q} \subseteq Q^\omega$, which does not guarantee that $\widehat{Q}$ is finite—for, by Lemma 2.2, Q^ω is almost always infinite in this case (it is finite only in the trivial case that Q is either empty or a singleton). In sum, the set of state tuples $\widehat{Q}$ of a finite combinatorial-state automaton is necessarily finite, whereas the $\widehat{Q}$ of an infinite combinatorial-state automaton may be either finite or infinite.

How is the transition function of a combinatorial-state automaton defined? Chalmers writes:

> State-transition rules are determined by specifying, for each element of the state-vector, a function by which its new state depends on the old overall state-vector and input-vector, and the same for each element of the output-vector. (Chalmers 1994b, 394.)

While this is not quite as formal as one would wish it to be, it seems that Chalmers has the following in mind: the state transition function takes each pair of state tuple and input tuple to some state tuple and some output tuple. In other words, the function is a mapping from the Cartesian product of the set of state tuples and the set of input tuples to the Cartesian product of the set of state tuples and the set of output tuples. In symbols, the transition function of a combinatorial-state automaton is a mapping $\delta : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \to \widehat{Q} \times (\widehat{\Gamma} \cup \{\epsilon\})$, supposing we include ϵ-moves.[3]

Drawing these considerations together, then, we have the following

[3]To be precise, Chalmers writes in the above block quotation of two functions, one for the state tuples and another for the output tuples—that is to say, of two functions of the forms $\delta_1 : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \to \widehat{Q}$ and $\delta_2 : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \to \widehat{\Gamma} \cup \{\epsilon\}$. However, such two functions can trivially be coalesced into one function, as I have chosen to do. This is simply a matter of notational convenience.

leviathan of a definition.

Definition 4.2. *A* combinatorial-state automaton *is an octuple* $(Q, \Sigma, \Gamma, \widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \delta, q_0)$*, where*

(i) Q *is a finite, non-empty set;*

(ii) Σ *and* Γ *are alphabets;*

(iii) either $\widehat{Q} \subseteq Q^i$ *for some* $i \in \mathbb{Z}_+$*, or* $\widehat{Q} \subseteq Q^\omega$*;*

(iv) $\widehat{\Sigma} \subseteq \Sigma^j$ *for some* $j \in \mathbb{Z}_+$*;*

(v) $\widehat{\Gamma} \subseteq \Gamma^k$ *for some* $k \in \mathbb{Z}_+$*;*

(vi) $\widehat{Q}$*,* $\widehat{\Sigma}$*,* $\widehat{\Gamma}$ *are non-empty;*

(vii) $\delta : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \to \widehat{Q} \times (\widehat{\Gamma} \cup \{\epsilon\})$ *is the transition function;*

(viii) $q_0 \in \widehat{Q}$ *is the initial state tuple.*

If $\widehat{Q} \subseteq Q^i$ *for some* $i \in \mathbb{Z}_+$*, the automaton is called* finite*; if* $\widehat{Q} \subseteq Q^\omega$*, the automaton is* infinite*.*

The next result follows at once:

Theorem 4.1. *Any combinatorial-state automaton whose set of state tuples is finite is a finite transducer. In other words, if* $\mathcal{C} = (Q, \Sigma, \Gamma, \widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \delta, q_0)$ *is a combinatorial-state automaton and* $\widehat{Q}$ *is finite, then* $\mathcal{C}' = (\widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \delta, q_0)$ *is a finite transducer.*

Proof. It is routine to check that conditions (i)–(v) of Definition 2.1 are true of $\mathcal{C}'$:

(i) By Definition 4.2, $\widehat{Q}$ is non-empty. By hypothesis, it is finite.

(ii) By Definition 4.2, Σ is finite. Hence, Σ^i is also finite for any $i \in \mathbb{Z}_+$ by Lemma 2.1, and since $\widehat{\Sigma} \subseteq \Sigma^i$, we have that $\widehat{\Sigma}$ is finite. Also, $\widehat{\Sigma}$ is non-empty by Definition 4.2. Therefore, $\widehat{\Sigma}$ is an alphabet.

(iii) An analogous argument shows that $\widehat{\Gamma}$ is an alphabet.

(iv) $\delta : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \rightarrow \widehat{Q} \times (\widehat{\Gamma} \cup \{\epsilon\})$ by Definition 4.2.

(v) Again, $q_0 \in \widehat{Q}$ by Definition 4.2. Q.E.D.

As a corollary to this theorem, we get a result noted also by Chalmers (1994b, 395):

Corollary. *Every finite combinatorial-state automaton is a finite transducer.*

Proof. As we have seen, the set of state tuples of a finite combinatorial-state automaton is finite. Q.E.D.

Intuitively, Theorem 4.1 holds because the "combinatorial" nature of the states of a combinatorial-state automaton is not actually made use of in the computation; the state transitions are determined solely by the *tuples*, not by the *elements* of the tuples. This is reminiscent of Fodor and Pylyshyn's (1988; see also Fodor & McLaughlin 1990) critique of such neural network architectures whose operations are insensitive to the inner structure of their representations. Constituent structures (such as trees in natural language syntax) are an extremely powerful way of representing the structure of the world but, if the processes of the cognitive system cannot operate on those structures, the structures are rendered useless.

However, the above theorem has nothing to say about such combinatorial-state automata whose set of state tuples is infinite. Thus, Chalmers's (1994b, 395) claim for the Turing-completeness of combinatorial-state automata may well hold for this kind of automata.

And it does—in fact, an even stronger result holds for infinite combinatorial-state automata. Brown (2004) has pointed out (though not proved) that these automata are in fact *too* effective: they exceed Turing machines in their computational power. Both classically computable and classically uncomputable functions are computable by infinite combinatorial-state automata, so that the formalism is hypercomputational. Here is one way of demonstrating the matter.

Theorem 4.2. *For any function $f : \mathbb{N} \to \mathbb{N}$, there exists an infinite combinatorial-state automaton that computes f.*

Proof. Let $f : \mathbb{N} \to \mathbb{N}$ be arbitrary; we show how to construct the desired automaton $\mathcal{C} = (Q, \Sigma, \Gamma, \widehat{Q}, \widehat{\Sigma}, \widehat{\Gamma}, \delta, q_0)$. Let $Q = \{a, b, c\}$, $\Sigma = \Gamma = \{1\}$, and $\widehat{\Sigma} = \Sigma^1 = \Sigma$ and $\widehat{\Gamma} = \Gamma^1 = \Gamma$. We define three mappings $q, r, s : \mathbb{N} \to Q^\omega$ as follows: for each $n \in \mathbb{N}$, we put

$$q(n) = (\underbrace{a, a, \ldots, a}_{n+1 \text{ times}}, c, c, c, \ldots),$$

$$r(n) = (\underbrace{b, b, \ldots, b}_{n+1 \text{ times}}, c, c, c, \ldots),$$

$$s(n) = (\underbrace{c, c, \ldots, c}_{n+1 \text{ times}}, a, a, a, \ldots).$$

Then q, r and s are obviously one-to-one and, moreover, the images $q(\mathbb{N})$, $r(\mathbb{N})$ and $s(\mathbb{N})$ are pairwise disjoint. We set $\widehat{Q} = q(\mathbb{N}) \cup r(\mathbb{N}) \cup s(\mathbb{N})$ and $q_0 = q(0)$.

Next, we define the transition function $\delta : \widehat{Q} \times (\widehat{\Sigma} \cup \{\epsilon\}) \to \widehat{Q} \times (\widehat{\Gamma} \cup \{\epsilon\})$ as follows: for each $n \in \mathbb{N}$, we put

$$\begin{cases} \delta(q(n), \epsilon) = (r(n), \epsilon), \\ \delta(q(n), 1) = (q(n+1), \epsilon), \\ \delta(r(n), \epsilon) = (s(f(n)), \epsilon), \\ \delta(r(n), 1) = (s(f(n)), \epsilon), \\ \delta(s(n), \epsilon) = (s(n-1), 1), \text{ if } n > 0, \\ \delta(s(n), 1) = (s(n-1), 1), \text{ if } n > 0, \\ \delta(s(0), \epsilon) = (s(0), \epsilon), \\ \delta(s(0), 1) = (s(0), \epsilon). \end{cases}$$

This completes the construction.

$\mathcal{C}$ is given input as follows: each $n \in \mathbb{N}$ is coded as the string

$$u_n = \underbrace{11\ldots1}_{n \text{ times}}.$$

Then, as can be checked from the above definition of δ, given u_n as input, one of the output strings emitted by $\mathfrak{C}$ will be the string

$$u_{f(n)} = \underbrace{11\ldots1}_{f(n) \text{ times}},$$

in other words, the code of $f(n)$. Writing R for the relation computed by $\mathfrak{C}$, we therefore have that $\{(u_n, u_{f(n)}) \mid n \in \mathbb{N}\} \subseteq R$, so that the function f is $\mathfrak{C}$-computable. Q.E.D.

Corollary. *The formalism of (infinite) combinatorial-state automata is hypercomputational.*

Whether this state of affairs counts as for or against the formalism of combinatorial-state automata depends, of course, on one's philosophical commitments: on whether one is willing to include hypercomputational structures among computational entities. However it be, there are independent reasons for discrediting infinite combinatorial-state automata. Namely, I shall next argue that such automata cannot be implemented, in the way pictured by Chalmers (1994b), in any reasonable physical system.

To see this, let us consider the implementation conditions of combinatorial-state automata envisaged by Chalmers (1994b). Essentially, the definition of the implementation of a combinatorial-state automaton is to be analogous to that of the implementation of a finite transducer. The implementing system is to mirror in its structure and function the logical structure and state transitions of the automaton. Hence, in order that a physical system $\mathfrak{S}$ implement a given combinatorial-state automaton, there has to be a decomposition of the system into states that correspond to the state tuples of the automaton (Chalmers 1994b, 394). In other words, there has to be a set of states M of the system $\mathfrak{S}$ so that the elements of M stand in a one-to-one relation to the tuples in the set of state tuples $\widehat{Q}$ of the automaton. If the combinatorial-state automaton in question has a finite set of states, there is no problem in princi-

ple. We simply find out whether it is possible to partition the physical system so that the requisite number of states can be found, and so that the causal transitions between these states mirror the abstract state-transition relations of the automaton 'reliably'.

But suppose the automaton we wish to implement has an infinite set of state tuples. In this case, in order for the physical system $\mathcal{S}$ to implement the automaton, it is necessary that the system's state space be *infinitely divisible*. This is problematic, since it is highly unclear whether relevant state spaces admit of such division without bound. For instance, as far as we know, any given brain has only finitely many *computationally relevant* states—states that matter from an information-processing point of view. Thus, it is unclear whether combinatorial-state automata with an infinite set of state tuples are in fact implementable in the physical systems cognitive science is interested in. On the other hand, the fact that combinatorial-state automata with a finite set of state tuples are implementable in principle in such systems is not of much help, since we saw such automata to be nothing but finite transducers (Theorem 4.1). It is therefore questionable whether "[t]he theory of implementation for combinatorial-state automata provides a basis for the theory of implementation in general" (Chalmers 1994b, 396).[4]

Nor is this all. As we saw with Definition 4.1, Chalmers's account of implementation rests on a notion of 'reliable causation':

> What is required is not just a mapping from states of the system onto states of the [automaton ...]. The added requirement that the mapped states must satisfy reliable state-transition rules is what does all the work. (Chalmers 1994b, 396.)

While the requirement is perfectly intuitive, it suffers from a lack of rigour that renders the theory less attractive than at first seemed. If all we can say of the

[4] As an interesting aside, we may note that Turing (1936–7/1965, 136) argued against an infinite number of states in the context of computing machines on the grounds that if an infinite number of states be allowed, "some of them will be 'arbitrarily close' and will be confused".

implementation relation is that it needs to be one-to-one and 'reliable', how much does this explicate and expand upon our pre-theoretic, intuitive notion of implementation? The technical formalism of combinatorial-state automata loses much of its appeal, and much of its utility, if the best we can do is conjoin it with the untechnical, imprecise notion of 'reliable causation'.

A further difficulty is generated by a relativity inherent in Chalmers's definition, as Scheutz (1999; 2001) has pointed out. Recall that the definition requires, for an implementation relation to hold, that the states of the computational structure be in a certain correspondence with "state-types" of the physical system. But just what are these state-types, and how are they to be individuated? Are they dictated to us by some physical theory? If so, what is that theory? If not, we are free to define these state-types *at will*. And while such a move may evade the Scylla of having to identify a physical theory which can give us the "correct" state-types, it seems to drive us straight into the Charybdis of Searle's Thesis. I shall return to this topic in Chapter 7, since it plagues other theories of implementation as well.

In conclusion, I list the following problems with Chalmers's (1994b) account of implementation. Firstly, the formalism of combinatorial-state automata, at least in its present state, offers no improvement upon classical computational structures. Finite combinatorial-state automata are nothing but finite transducers (Theorem 4.1); most infinite combinatorial-state automata are hypercomputational (Theorem 4.2) and possibly non-implementable. Secondly, although Chalmers's definition of implementation (Definition 4.1) seemingly blocks Searle's Thesis (and Putnam's Realisation Theorem) by requiring the causal transitions of the implementing system to be reliable, this notion of "reliable causation" is yet too underspecified and imprecise to support a general theory of implementation. Finally, there is a problem with the individuation of physical systems inherent in Chalmers's account: how exactly is a physical system to be divided into state-types?

5

Implementation: A Model-Theoretic Approach

Functions & algorithms. The model-theoretic view of implementation. Explication of Searle's Thesis. The problem of vague notions.

The main problem with Definition 4.1 was its specificity to one computational formalism, one that moreover seems inadequate for the purposes of cognitive explanation. On the other hand, Chalmers's (1994b) idea of a general theory of implementation, to be couched in the formalism of combinatorial-state automata, was found wanting. Is there, then, any prospect of a general theory of implementation?

One possibility is to abandon the notion of computation altogether and explore the related question, "Under what conditions does a physical system realise a given function?" This is the strategy adopted by Scheutz (1999), and it has the advantage of abstracting away from the details of any specific computational structure, thus seemingly offering a general definition of when a system can be said to be embodying a function. However, for the purposes of cognitive science, at least as classically conceived, this will not do. To abandon computation in favour of functional realisation is to take a backward step from computationalism to behaviourism.

To see this, we need only consider that to realise or embody a function, it suffices for a physical system to get the input–output pairings, defined by the function, right. To *compute* that function the system needs to do more, since any computational structure specifies not just a function, but an *algorithm* for arriving at the values of the (computable) function. By the same token, a

physical system that implements some computational structure will not realise the functions computed by the structure in just any old way—the system has to, in one way or another, respect the algorithm by which the function is computed by the computational structure.

Therefore, the notion of algorithm cannot be done away with. Let us now turn to Copeland's (1996) account of implementation, which respects this intuition and builds upon the notion of computations as formal specifications of algorithms. Copeland (1996, 337) notes that an algorithm is always specific to some computational architecture: the algorithm calls for certain primitive operations of the architecture that may not be available in another architecture.[1] On the other hand, it is possible to provide a *formal specification* of any algorithm and the architecture that supports this algorithm. This formal specification may be thought of as a set of axioms specifying the behaviour of the architecture in question (Copeland 1996, 337–338).

In order that an entity can be said to compute it is first of all necessary to "label" that entity (Copeland 1996, 338). A *labelling scheme* for an entity is a "designation of certain parts of the entity as label-bearers" (Copeland 1996, 338). Moreover, it is required of the labelling that there be a "method for specifying the label borne by each label-bearing part at any given time" (Copeland 1996, 338). The purpose of the labelling scheme is to associate parts of the entity with elements of the formal specification of the algorithm-architecture pair. As such, the labelling scheme performs the same duty as the implementation mapping φ of Definition 4.1.

Let f be a function and α_f an algorithm that takes the arguments of f as input and delivers the values of f as output (or, more generally, takes encodings of the arguments of f as input and delivers encodings of the values of f as output). Let $S(\mathcal{A}, \alpha_f)$ be a formal specification of α_f and its supporting

[1] Copeland (1996) uses the term 'architecture' in place of the term I have been using throughout this thesis, 'computational structure'. The terms are synonymous, and for the purposes of the present chapter, I shall adopt Copeland's terminology, using it side by side with mine.

architecture $\mathcal{A}$. Let e be an entity and L a labelling scheme for that entity. Then the ordered pair (e, L) is said to be a *model* of $S(\mathcal{A}, \alpha_f)$ if and only if the axioms of $S(\mathcal{A}, \alpha_f)$ are true of e under the labelling L (Copeland 1996, 338). With this notion in hand, we may define:

Definition 5.1 (Copeland 1996, 338)**.** *An entity e* computes *a function f if and only if there exist a labelling scheme L for that entity and a formal specification $S(\mathcal{A}, \alpha_f)$ such that (e, L) is a model of $S(\mathcal{A}, \alpha_f)$.*

Note that this definition is perfectly general: no restrictions are placed on the nature of the entity e. The conditions for when a *physical* system is computing a given function are thus couched in a general account of computation. The sense in which a computational structure (such as a Turing machine) computes differs, however, from the sense in which a physical object (such as a desktop computer) computes. The former computes by virtue of being a computational structure—in fact, 'computation' itself is defined relative to computational structures. The latter, in contrast, computes by virtue of being a model—under some or other labelling scheme—of a computational structure. Thus, computation within $\mathcal{A}$ must not be confounded with computation within e; the word 'computation' is being used homonymously.

More importantly, however, the definition is general in the sense that it is not specific to any computational structure, such as the finite transducer or the Turing machine. It is because of this universality that the definition may be considered a general definition of implementation.

Despite this generality, and despite its intuitive appeal, Definition 5.1 will however not do, according to Copeland (1996). The problem is that the definition is open to Searle's Thesis. Here it is cast in the terms of the present account of implementation, and in Copeland's (1996) notation:

> For any entity x (with a sufficiently large number of discriminable parts) and for any architecture-algorithm specification y there exists a labelling scheme L such that $\langle x, L\rangle$ is a model of y. (Copeland 1996, 343.)

We need to be careful here to interpret the small print, "with a sufficiently large number of discriminable parts", the right way. Two interpretations, in fact, suggest themselves. On one, Searle's Thesis is taken to claim that there is some preset number p of discriminable parts that it suffices for an entity to have in order for it to model *any* formal specification whatsoever—in other words, to implement each and every computation. On another, Searle's Thesis is taken to claim that for each formal specification there exists a number p of discriminable parts that it suffices for an entity to have in order to model this specification. Thus, the order of quantifiers varies, as can be readily seen in the following explications of these two interpretations:

Hypothesis 5.1 (Searle's Thesis, First Interpretation)**.** *There is a $p \in \mathbb{Z}_+$ such that: for any entity e with at least p discriminable parts, for any architecture-algorithm specification $S(\mathcal{A}, \alpha_f)$, there is a labelling scheme L such that (e, L) is a model of $S(\mathcal{A}, \alpha_f)$.*

Hypothesis 5.2 (Searle's Thesis, Second Interpretation)**.** *For any finite architecture-algorithm specification $S(\mathcal{A}, \alpha_f)$, for any entity e, there is a $p \in \mathbb{Z}_+$ such that: if e has at least p discriminable parts, then there is a labelling L of e such that (e, L) is a model of $S(\mathcal{A}, \alpha_f)$.*

In Hypothesis 5.2, I have added the requirement that the architecture-algorithm specification must be finite; the reason for this adjustment will become evident shortly.

It is unclear which of these interpretations Copeland (1996) has in mind.[2] However, it is easy to show that, on the first interpretation (Hypothesis 5.1), Searle's Thesis does not hold. On the second interpretation (Hypothesis 5.2), on the other hand, the thesis does hold, and consequently becomes a theorem of Copeland's account of implementation. Indeed, Copeland (1996, 339) calls

[2]The same goes for Searle—cf. his original formulation of his thesis: "[f]or any program and for any sufficiently complex object, there is some description of the object under which it is implementing the program" (Searle 1992, 208). Again, it is hard to tell the order of the quantifiers in this statement.

the result "Searle's Theorem", and gives a rather thorough proof (Copeland 1996, 343–346). Without going into its intricate details, I shall next recount the essentials of Copeland's proof in order to establish the truth of Hypothesis 5.2, on the one hand, and the falsity of Hypothesis 5.1, on the other.

The proof proceeds from consideration of an example of a formal specification, that of a very simple register machine M, and a sufficiently large entity e, namely, one of the walls in Searle's office. The architecture of M is as follows. It consists of three 8-bit registers: an instruction register I, a data buffer D, and an accumulator A. That is to say, any one of these registers is capable of containing a representation of an 8-bit binary number, such as 00000010. Like any computational structure, M has some primitive operations, such as adding the contents of D to the contents of A and storing the result in A. The operation to be performed is determined by the contents of the instruction register I. The formal specification $S(M, \alpha_f)$ of M, where f is the function under consideration and α_f its associated algorithm, is then a finite set of axioms of the form "if the contents of I are such-and-such, ACTION-IS(I, D, A)", where ACTION-IS(I, D, A) specifies the action to be taken by the machine—what to do with its registers (Copeland 1996, 341).

Copeland then proceeds to show how it is possible to give a labelling of the wall so that the wall is a model of $S(M, \alpha_f)$ under this labelling. The labelling itself hinges upon a procedure Copeland (1996, 344) calls "Searlification". This consists in picking out regions of the wall to represent 8-bit binary numbers in a convenient manner so that the axioms of $S(M, \alpha_f)$ can be seen to be true of the wall under this labelling. Since the microstructure of a wall is constantly evolving, there is no hope of carving out fixed regions of the wall to represent the registers of M in a time-invariant way; rather, the locus of a register-label in the wall may change as the computation proceeds. Copeland then shows that the axioms of $S(M, \alpha_f)$, when interpreted as material implications quantified over any desired time interval, are true of the wall under such a Searlified

labelling. I shall not recite the details of this process here; the interested reader may find them in Copeland (1996, 343–346).

With the procedure of Searlification, the axioms of the particular architecture-algorithm specification $S(M, \alpha_f)$ are seen to be true of Searle's wall e. The truth of Hypothesis 5.2 then follows by universal generalisation, since a similar procedure of Searlification could be applied to any specification and any entity (Copeland 1996, 343). Hypothesis 5.2 becomes Theorem 5.1:

Theorem 5.1. *For any finite architecture-algorithm specification $S(\mathcal{A}, \alpha_f)$, for any entity e, there is a $p \in \mathbb{Z}_+$ such that: if e has at least p discriminable parts, then there is a labelling L of e such that (e, L) is a model of $S(\mathcal{A}, \alpha_f)$.*

I shall return to an examination of the consequences of this result in a moment. First, however, let us see why Copeland's argument is insufficient to derive Hypothesis 5.1, the stronger interpretation of Searle's Thesis.

To see this, we need only take heed of the fact that according to Hypothesis 5.1, there is a certain preset number $p \in \mathbb{Z}_+$ of parts that it suffices for an entity to have in order to constitute a model of any formal specification $S(\mathcal{A}, \alpha_f)$, under some labelling. But now, however large this p may be, it is possible to define an architecture $\mathcal{A}'$ and an algorithm α'_f in such a way that the architecture-algorithm specification $S(\mathcal{A}', \alpha'_f)$ requires p' parts of an entity, with $p' > p$, if that entity is to constitute a model of $S(\mathcal{A}', \alpha'_f)$, under any labelling. For instance, if $\mathcal{A}$ is a register machine, we may simply define the machine $\mathcal{A}'$ to have registers that can store so large binary numbers that no entity with exactly p or less parts could be a model of $S(\mathcal{A}', \alpha'_f)$ under any Searlified labelling. This possibility is afforded by the fact that the set of natural numbers is not bounded from above; there is no greatest number; for any number, we can find one that is even greater. It is then not true that any entity with p discriminable parts will be a model of $S(\mathcal{A}', \alpha'_f)$ under a Searlified labelling L. Hence, Hypothesis 5.1 fails. The above reasoning also establishes why Hypothesis 5.2 (or Theorem 5.1) requires the architecture-algorithm spec-

ification to be finite. If infinite registers, for instance, were allowed, then it would not be possible to establish a fixed number $p \in \mathbb{Z}_+$ of discriminable parts it would suffice an entity to have in order to constitute a model of the specification.[3]

How about Theorem 5.1, then? Although much weaker than Hypothesis 5.1, it is still strong enough to be unsettling to the computationalist. According to a computationalist of the ontological variety, there are computations whose implementation guarantees the possession of cognitive (or other mental) states. By Theorem 5.1, it follows that merely possessing sufficiently many discriminable parts suffices for the possession of mental states.[4] So, at least by the computationalist's lights, Definition 5.1 is unsatisfactory. Why exactly? In Copeland's (1996) diagnosis, the Searlified labelling scheme used in proving Theorem 5.1 constitutes a *nonstandard interpretation*. Accordingly, he proposes to modify the definition so that only standard, or "honest", models are accepted:

Definition 5.2 (Copeland 1996, 348)**.** *An entity e* computes *a function f if and only if there exist a labelling scheme L for that entity and a formal specification $S(\mathcal{A}, \alpha_f)$ such that (e, L) is an honest model of $S(\mathcal{A}, \alpha_f)$.*

What, then, does standardness or honesty consist in? Copeland proposes two criteria: "[f]irst, the labelling scheme must not be ex post facto", and "[s]econd, the interpretation associated with the model must secure the truth of appropriate counterfactuals concerning the machine's behaviour" (Copeland 1996, 350–351).

As to the first criterion, recall that the labelling used in proving Theorem 5.1 was constructed, by the procedure of Searlification, after the formal

[3] It is true, though, that this problem may be sidestepped by assuming physical space to be infinitely divisible (i.e., divisible without bound). However, I shall here assume, with Turing (see footnote 4, p. 31, above), that infinite divisibility implies, if nothing else, at least a topological inconvenience.

[4] I assume the class **Cog** (see Hypothesis 1.2) to include such structures whose formal specifications are finite, so that Theorem 5.1 applies. This assumption will hardly be denied by anyone, seeing that a Turing machine, whose tape has been truncated to a length k, can be finitely specified *for any arbitrarily large* $k \in \mathbb{N}$.

specification of the architecture-algorithm pair $S(M, \alpha_f)$ had been fixed. This method parallels the construction used in proving Putnam's Realisation Theorem, in which the states α and β of the open system were defined after a particular automaton had been fixed (see pp. 17–18, above). Copeland (1996, 348) expresses the point rather eloquently: "all the computational activity occurred *outside* the wall". Apparently, a *non* ex post facto labelling (or definition of system states) would then be one defined independently of the architecture-algorithm specification (or computation) in question.

As to the second criterion, Copeland (1996, 349–350) stresses the point that, in the case of his register machine M, the procedure of Searlification proceeded by interpreting the axioms of $S(M, \alpha_f)$ as material implications. However, the ACTION-IS statements of $S(M, \alpha_f)$ are supposed to support suitable counterfactuals concerning the machine's behaviour (Copeland 1996, 341–342). When interpreted as material implications with the procedure of Searlification, the statements lose their counterfactual-supporting force. For example, since any material implication with a false antecedent is true, a whole host of such implications are true of Searle's office wall, but none of these conditionals give any "guidance as to how M is designed to behave" (Copeland 1996, 342). An honest or standard interpretation of $S(M, \alpha_f)$, in contrast to a Searlified one, would have support the necessary counterfactuals (whatever they be). As an analogue of Searlified labellings, Copeland (1996, 347–348) cites the model-theoretic Löwenheim–Skolem Theorem, from which the *prima facie* paradoxical situation follows that the statement "$|\mathbb{N}| < |\mathbb{R}|$" is true in a model which includes only natural numbers and sets of natural numbers. He points out that in the nonstandard model the statement, while true, is not *about* real numbers in the way it is about them if considered in a standard model that includes real numbers. Similarly, while the axioms of $S(M, \alpha_f)$ are true of Searle's wall, this fact

> goes no way toward showing that the wall *acted* in accordance with the

> instructions in the algorithm. The wall so acted only if the referent of '$\mathbb{R}$' in Skolem's countable model is uncountable! (Copeland 1996, 348.)

While this is all very well, the suggestion that an honest model of a computing machine must "secure the truth of appropriate counterfactuals concerning the machine's behaviour" (Copeland 1996, 350–351) is, at the end of the day, quite vague. What we need is a general formulation of this requirement for all types of entity and all types of architecture-algorithm specification, but as far as I can see, Copeland has not provided such a general formulation. Similar remarks apply to the notions of 'entity' and 'formal specification'. They purport to offer a general analysis of implementation, but their generality comes at the expense of rigour and precision. However, if we are to tackle the issue of implementation, and find out whether implementation of computation is observer-relative or not, we need a theory which is both general and exact, so that the discussion may proceed on clear terms. In the next chapter, I shall sketch an account of implementation that tries to avoid these problems caused by intuitive yet imprecise notions, by replacing those notions with other, precise (yet hopefully none the less intuitive) notions. The driving idea will still be to construe implementation as a model-theoretic relation; however, my goal is to give a definition of the standardness of interpretation that does not rely on counterfactuals.

6

Implementation: Dynamical Systems

Classicism, connectionism, & dynamicism. Definition of the implementation of Turing machines in dynamical systems. Demonstration that the definition does not imply Searle's Thesis. Discussion of the definition.

At this point, our quest for a definition of implementation has become more philosophical than cognitive-scientific. Starting with rather innocent notions from the theory of automata and computation, we have been led to ponder divisibility issues and the peculiarities of the infinite—classical metaphysical riddles already tackled by the philosophically minded Greeks. From an *aporia* such as this it is natural to seek a way out. In this chapter, I propose to do away with the problematic notion of 'physical system', giving a definition of the implementation of Turing machines in dynamical systems. I shall demonstrate that the new definition does not imply Searle's Thesis, although it may still imply weaker kinds of relativistic claims.

It has become customary to distinguish three paradigms within contemporary cognitive science: classicism, connectionism, and dynamicism. Classicism, standing on the foundation provided by computationalism, assumes that the most fruitful way of explaining cognitive phenomena is to explain them using symbolic computational frameworks—at the end of the day, a classicist explanation of a cognitive phenomenon amounts to picking out, among all the structures of the class **Cog** (see Hypothesis 1.2), the one that correctly mirrors the phenomenon in question. To this style of explanation the connectionist

paradigm is much opposed, devising as it does cognitive models inspired by neuroscience. The more recent dynamicist paradigm (e.g., van Gelder 1998) relies on the concept of dynamical system as the proper explanatory framework for the study of cognition.

As far as sundry details and minutiae are concerned, the three paradigms often seem opposed to each other—the infamous debate between Smolensky (1988) and Fodor and McLaughlin (1990) is a case in point. At bottom, however, the paradigms share many assumptions. For one, each paradigm is committed to mechanism, the belief that the mind is a mechanical device whose operations can be explained by mechanical laws of one form or another. Moreover, the oft-quoted difference between classicism and connectionism, that artificial neural networks are not Turing-equivalent, is simply false (see Siegelmann & Sontag 1995). Thus, even though *at surface* the three paradigms are much unlike one another, *at bottom* they agree to a great extent. One of the aims of the present chapter is to show how the dynamicist and classicist paradigms can be brought to bear on each other.

Perhaps the most important single fact concerning Turing machines is that they are *syntactic* beings: as Fodor and Pylyshyn (1988) have stressed, the strings with which a Turing machine operates have the potential of having an inner structure to which the machine's operations are sensitive. I shall now sketch a theory that attempts to show how this syntactic nature of classical computing machines can be recast in terms of dynamical systems theory. The outcome is an account of implementation that hopefully avoids the problems I have previously identified with the Chalmersian and Copelandian positions. In this undertaking, I propose to replace the vague notions of 'physical system' and of 'entity' with the precise notion of dynamical system, and the notion of 'formal specification' with the concept of Turing machine. These choices, naturally, restrict the resulting analysis in certain ways; their justification lies in their facilitating discussion in precise, unambiguous terms. The account

to be sketched gives, moreover, a rigorous definition for the standardness of interpretation, something that was found lacking in Copeland's (1996) theory of implementation.

The concept of dynamical system embodies the notion of a system, capable of assuming several different states, evolving over time. In its most general formulation, a dynamical system consists of a phase space together with an action of a group on this space (see, e.g., Arnol'd 1992, 13–16, 57–61):

Definition 6.1. *A* dynamical system *is a triple* $(M, (T, +, <), \{f_t\})$ *(or* $(M, T, \{f_t\})$ *for short), where* M *is a set,* $(T, +, <)$ *is a linearly ordered group, and* $\{f_t\} = \{f_t$ *is a transformation of* $M \mid t \in T\}$ *is an action of* $(T, +, <)$ *on* M*.*

The set M is the phase space of the system. For each $t \in T$, the action of the group $(T, +, <)$ defines a transformation $f_t : M \to M$ of the phase space M; for any $m \in M$, $f_t(m)$ is the state of the system at time t, with initial state m. A dynamical system will be called *noncyclical*, if $f_t(m) \neq f_u(m)$ for all $m \in M$ and $t, u \in T$ with $t \neq u$. If a dynamical system is not noncyclical, it is called *cyclical*. Any subset $F \subseteq M$ is called a *phase*. The *trajectory* of a point $m \in M$ in phase space is the set $\xi(m) = \{f_t(m) \mid t \in T\}$.

Note that by this definition, the phase space of a dynamical system need not be a metric space or even a topological space—indeed, it needs be only a set. For most systems of interest, the phase space of course has a metric structure, often Euclidean. Moreover, time in these systems is usually represented by the group of real numbers, rather than by any arbitrary linearly ordered group. As van Gelder (1998) has argued, such systems are of great interest to cognitive science primarily for two reasons. Firstly, they allow us to make quantitative statements concerning the relations between their states, thanks to their metrics. Secondly, they capture the dynamic "flow" of time, thanks to time being represented by the reals—thereby forcing us to focus our attention on the nature of state transitions, rather than on the nature of the states

themselves. Such dynamical systems, when taken as explanations or models of cognitive phenomena, stand in opposition to classical Turing-machine explanations in both these respects. However, as I shall next argue, the two paradigms do have a common ground, or an interface, since Turing machines can readily be conceived as dynamical systems.

Let $\mathcal{D} = (M, (T, +, <), \{f_t\})$, then, be a dynamical system. In order that $\mathcal{D}$ can be seen to implement a computational structure, it has to be "discretised" in one way or another. Strictly speaking, such a procedure is only needed if M is uncountable, but in the case of systems with a countable phase space we may always take the operation of discretisation to be the identity mapping. Note that a sort of discretisation was performed in the proof of Putnam's Realisation Theorem when the interval states were defined; what follows is, in effect, an extension of this Putnamian procedure, or "Putnamification".

Any countable partition $\mathcal{F}$ of the phase space M will be called a *grouping* of $\mathcal{D}$. A grouping forms the basis for the procedure of discretising $\mathcal{D}$: it will function as the phase space of the resulting discrete system. The next step is to define an action $\{g_t\}$ of the group $(T, +, <)$ on this phase space. In order that the resulting discrete system can be called a discretisation *of* $\mathcal{D}$, we need to ensure that the evolution of the former honours or mirrors that of the latter; we need to define the transformations $g_t : \mathcal{F} \to \mathcal{F}$ in a 'natural' way. This corresponds to defining the g_t so that the discrete process is in state $[m]_\mathcal{F}$ at time t exactly when $\mathcal{D}$ is in state m at time t. Setting $g_t([m]_\mathcal{F}) = [f_t(m)]_\mathcal{F}$ for all $t \in T$ and for all $m \in M$ gives the desired result.[1]

Putting all this together, now, we have the following definition.

Definition 6.2. *The* discretisation *of a dynamical system* $\mathcal{D} = (M, T, \{f_t\})$ *by its grouping* $\mathcal{F}$*, written* $\mathcal{D}|_\mathcal{F}$*, is the triple* $(\mathcal{F}, T, \{g_t\})$*, where* $\{g_t\}$ *is an action*

[1]Note that $[m]_\mathcal{F}$ ranges over all sets $F \in \mathcal{F}$ as m ranges over the phase space M.

of $(T, +, <)$ *on* $\mathcal{F}$ *defined as follows: for each* $t \in T$ *and* $m \in M$,

$$g_t([m]_{\mathcal{F}}) = [f_t(m)]_{\mathcal{F}}. \qquad (*)$$

Figuratively speaking, the procedure of discretisation prepares the system under consideration for purposes of syntactic interpretation. Hence, the operation could also be described as the 'syntactification' of the system.

Consider next a Turing machine $\mathcal{M}$. At any step of its computation, the contents of the machine's tape can be described as an infinite sequence

$$\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots$$

of symbols, the tape squares being indexed by the integers. Since only a finite number of computation steps have elapsed since the computation commenced, all tape symbols are blank from some square onwards, in both directions. Formally speaking, there exist $n_B, m_B \in \mathbb{Z}$ such that for all $n \in \mathbb{Z}$, if $n > n_B$ or $n < m_B$, then $x_n = B$.[2] The squares with $n > n_B$ or $n < m_B$ are then irrelevant as far as the machine's present configuration is concerned. Hence, the total configuration of the machine is represented by a finite tuple of the form

$$(x_{m_B}, x_{m_B+1}, \ldots, x_{k-1}, q, x_k, x_{k+1}, \ldots, x_{n_B-1}, x_{n_B}),$$

where q is the current state of the machine and x_k is the symbol currently in the scanner. Following Turing (1936–7/1965, 118), I shall call a description of this kind a *complete configuration* of $\mathcal{M}$.[3] The complete configuration is a function of the input string given to the machine and the current step of computation; the complete configuration of $\mathcal{M}$ at step $s \in \mathbb{N}$ with input $w \in \Sigma^*$ will be denoted by the expression $C_{w,s}$. Note that $C_{w,0} = (q, x_0, x_1, \ldots, x_n)$, where

[2] Recall that the letter B is used to signify the blank symbol. For other notational conventions, consult Definition 2.2.

[3] Another term that appears in the literature is 'instantaneous description' (see, e.g., Hopcroft & Ullman 1979, 148).

$x_0 x_1 \ldots x_n = w$.

The set of complete configurations of $\mathcal{M}$ will be denoted by $\mathfrak{C}(\mathcal{M})$; thus $\mathfrak{C}(\mathcal{M}) = \{C_{w,s} \mid w \in \Sigma^*,\ s \in \mathbb{N}\}$. (In the case of halting machines, we adopt the convention that if the machine goes into the halting state at computation step s_h, then $C_{w,s}$ is constant for all $s > s_h$.) If $C_{w,s} \in \mathfrak{C}(\mathcal{M})$ and $n \in \mathbb{N}$, then $\mathfrak{c}_n(C_{w,s})$ denotes the unique complete configuration $C_{w,s+n}$.

The idea now is to map the complete configurations of $\mathcal{M}$ to states of a discretisation $\mathcal{D}|_{\mathcal{F}}$ of the dynamical system $\mathcal{D}$, and to map system-time to discrete machine-time.

Definition 6.3. *Let $\mathcal{M}$ be a Turing machine and $\mathcal{D}|_{\mathcal{F}} = (\mathcal{F}, T, \{g_t\})$ a discretisation of a dynamical system $\mathcal{D}$. An* interpretation *of $\mathcal{M}$ is any pair (φ, τ) of mappings $\varphi : \mathfrak{C}(\mathcal{M}) \to \mathcal{F}$ and $\tau : T \to \mathbb{N}$.*

For the standardness of interpretation I propose the following criterion, or definition:

Definition 6.4. *An interpretation (φ, τ) of $\mathcal{M}$ is* standard *if the following conditions are met:*

(i) φ is one-to-one,

(ii) for all $t \in T$ and for all $C, D \in \mathfrak{C}(\mathcal{M})$,

$$\varphi(C) = g_t(\varphi(D)) \ \text{ if and only if } \ C = \mathfrak{c}_{\tau(t)}(D).$$

Standardness, then, consists in a sort of homomorphism between the discretisation and the Turing machine. Condition (i) states that the interpretation is not haphazard, in the sense that distinct complete configurations are required to map to distinct states of the discretisation. Condition (ii), in turn, asserts that the evolution of $\mathcal{M}$ mirrors the evolution of $\mathcal{D}|_{\mathcal{F}}$, and vice versa.

If the pair (φ, τ) is a standard interpretation, $\mathcal{D}|_{\mathcal{F}}$ is said to be a *model* of $\mathcal{M}$, written $\mathcal{D}|_{\mathcal{F}} \models \mathcal{M}$. The present account of implementation is then summarised by the next definition.

Definition 6.5. *Let $\mathcal{D}$ be a dynamical system and let $\mathcal{M}$ be a Turing machine. Then $\mathcal{D}$* implements *$\mathcal{M}$ if and only if there exists a grouping $\mathcal{F}$ of $\mathcal{D}$ so that $\mathcal{D}|_{\mathcal{F}} \models \mathcal{M}$.*

In the general case, Definition 6.5 does not support Searle's Thesis. To see this, consider the following *reductio*. Take a *stationary* system $\mathcal{D} = (M, T, \{f_t\})$, that is, a system for which $f_t(m) = m$ for all $m \in M$, $t \in T$. Let $\mathcal{F}$ be any grouping of $\mathcal{D}$ and consider the discretisation $\mathcal{D}|_{\mathcal{F}} = (\mathcal{F}, T, \{g_t\})$. Since $\mathcal{D}$ is stationary, it follows from condition $(*)$ of Definition 6.2 that $\mathcal{D}|_{\mathcal{F}}$ is stationary as well. Next, take a Turing machine $\mathcal{M}$, with input alphabet $\Sigma = \{0, 1\}$, that replaces all occurrences of 0 with 1 and all occurrences of 1 with 0. We may define a machine of this kind so that it satisfies the condition $C_{w,0} \neq C_{w,1}$ for all $w \in \Sigma^*$. Next, consider an interpretation (φ, τ) of $\mathcal{M}$; suppose it is standard. Since $C_{w,0} \neq C_{w,1}$, it follows by condition (i) of Definition 6.4 that $\varphi(C_{w,0}) \neq \varphi(C_{w,1})$. On the other hand, since $C_{w,1} = \mathfrak{c}_1(C_{w,0})$, it follows by condition (ii) of said definition that $\varphi(C_{w,1}) = g_{\tau^{-1}(1)}(\varphi(C_{w,0}))$. Putting these together, we have that $g_{\tau^{-1}(1)}(\varphi(C_{w,0})) \neq \varphi(C_{w,0})$. In other words, $\mathcal{D}|_{\mathcal{F}}$ is not stationary—a contradiction—and we have the following result:

Observation 6.1. *Definition 6.5 does not imply Searle's Thesis.*

This counter-example demonstrates the way conditions (i) and (ii) of Definition 6.4 work together to ensure that not everything implements every computation. The implementing system has to reflect the structure and evolution of the system implemented. Of course, it may still be possible to derive counter-intuitive results from this definition, in which case the definition is to be modified or altogether rejected. This is however not the place for a thorough study of the theorems Definitions 6.4 and 6.5 derive or fail to derive. For present purposes it is most important that the definitions do not derive Searle's Thesis in its most universal formulation, and that they offer a starting point from which to develop the theory further.

For purposes of illustration, though, consider the following. One might be tempted to attempt a generalisation of Putnam's Realisation Theorem by mapping the initial complete configurations $C_{w,0}$ of a Turing machine $\mathcal{M}$ to points of the phase space of a dynamical system $\mathcal{D} = (M, T, \{f_t\})$ in a one-to-one manner. Then, assuming that the system is noncyclical, it would seem possible to form a grouping $\mathcal{F}$ in a cunning way (cf. the definition of the states α and β in the proof of Putnam's Realisation Theorem, pp. 17–18, above) so that $\mathcal{D}|_{\mathcal{F}}$ would be seen to be a model of $\mathcal{F}$. If, as Putnam (1988, 121–125) contends, every open system is in fact noncyclical, this procedure would show that every open system implements every Turing machine. The reasoning here is, however, fallacious; the argument tacitly supposes that the system $\mathcal{D}$ has a countable infinity of trajectories. For, if two different initial complete configurations $C_{w,0}$ are mapped to states that belong to the same trajectory, then it will not be possible to define an interpretation of $\mathcal{M}$ that is standard—the condition of standardness will be broken in the same way as in the above counterexample. Therefore, in addition to the assumption of noncyclicality we need to assume (at least) that the phase space M of $\mathcal{D}$ is partitioned by an infinite set of trajectories, and so we are far from universal realisation.

It is an advantage of the present definition of implementation that it is, like Copeland's (1996) definition, very general. The dynamical system under consideration may be of any sort: it need not be a physical system, with its phase space and state transformations described by a physical theory, at all. For this reason, Definition 6.5 gives a criterion for studying the computational nature of any dynamical system whatsoever, be it physical, cognitive, social, or what have you.

On another level, the suggested definition is general in the sense that it lays down conditions for the implementation of Turing machines, the formalism under which all classical computational formalisms can be subsumed. Moreover, if the Church–Turing Thesis (Hypothesis 2.1) is endorsed, the definition gains

further support. Of course, the definition has nothing to say about hypercomputational structures. If the latter are accepted as genuinely computational (*pace* the Church–Turing Thesis), then the definition can be considered only approximatively useful, since our *desideratum* is a general theory of implementation of *all* kinds of computational structures.

At any rate, the suggested definition of implementation has virtue in that it dispenses with the vague notions of 'entity', 'formal specification of computational architecture and algorithm', and 'reliable causation'. The latter one was seen to be an integral part of Chalmers's (1994b) account of implementation (Definition 4.1); however, unless the meanings of both causation and reliability are significantly explicated, the idea remains obscure. The move to dynamical systems has the benefit that the problematic notion of causation is not needed at all. The requirement of reliability, on the other hand, is built into condition (ii) of Definition 6.4: the implementing system will transit from state to state reliably, in the sense that its evolution mirrors that of the implemented computation. What is more, this reliability is preserved from input to input: a standard interpretation is required to map all initial complete configurations of the Turing machine and, *ipso facto*, all possible traces of computation, in the required manner. There is no need of considerations pertaining to 'appropriate counterfactuals' of any sort.

7

Computation and Cognition

The problem of φ-groupings. Implications for ontological computationalism. Implications for instrumental computationalism. Argument for a pluralist conception of science, in general, & of cognitive science, in particular. Concluding remarks.

Having said all that, a potentially uncomfortable issue still remains with the account of implementation sketched in the previous chapter. In fact, this problem is in no way peculiar to that account, but plagues Chalmers's (1994b) and Copeland's (1996) theories as well. The problem has to do with the issue of how physical systems are to be interpreted.

Scheutz (1999; 2001) has already identified this trouble and has placed it under due scrutiny. In his terminology, the problem has to do with "φ-groupings", a *φ-grouping* being a division of a physical system into parts that together make up the whole system. An example of such a grouping is afforded by Chalmers's (1994b) definition of implementation (Definition 4.1), where the physical system $\mathcal{S}$ is divided into state-types. Likewise, Copeland's (1996) theory of implementation hinges upon this sort of division, as evidenced by the necessity of dividing an entity into parts that can then be labelled, or "Searlified". Finally, the account sketched in the previous chapter depends crucially on the notion of grouping (understood here in the technical sense given to that term therein).

The problem then, as Scheutz (1999; 2001) has pointed out, is that a φ-grouping such as any one of these is more or less arbitrary. The definitions of

implementation, in and of themselves, place no restrictions on the formation of the grouping; nor are such restrictions suggested by physical theory. This implies that implementation of computation is a relative matter after all, since no physical system has a *canonical* φ-grouping. Even if a non-relative definition of implementation could be fashioned in the sense that it would be seen that any one φ-grouping can implement at most one computation, this would appear to be to no avail, since any physical system can be φ-grouped in a number of ways.

According to Scheutz (1999; 2001), the problem of φ-groupings is the severest problem threatening the ambitions of any non-relativist theory of implementation. Worse still, if φ-groupings are to be de-relativised, this de-relativisation must come from the level of physical theory, argues Scheutz (1999; 2001). A de-relativisation attempt at the level of computability theory would certainly result in a circular argument.

But what is the bearing of all this on cognitive science? More explicitly, how do the accounts of implementation previously investigated, and the relativity of φ-groupings, relate to the two brands of computationalism identified at the beginning of this thesis, ontological and instrumental computationalism? I shall next examine the two in turn.

It is clear that Searle's Thesis, if true, leads the ontological computationalist to panpsychism. However, as we have seen, all accounts of implementation here studied appear strong enough to refute Searle's Thesis. Chalmers's (1994b) account evades the thesis by requiring that the implementing system be reliable in its causal 'mirroring' of the computation in question. Copeland's (1996) theory counters Searle's Thesis with the notion of model honesty. And, the account sketched in the previous chapter demonstrably does not suffer from the thesis, as shown by the counter-example produced in that chapter. On the other hand, all three accounts of implementation may be open to relativistic claims weaker than Searle's Thesis. For instance, we saw that Copeland

(1996) countered Theorem 5.1 by requiring the modelling relation between the implemented computation and the implementing entity to be 'honest'. However, as long as we do not possess a rigorous definition of this 'model honesty', relativistic claims may enter through the back door.

But even if this were the case—even if it were a theorem of our chosen account of implementation that some physical system implements more than one computation—what would this imply from the point of view of ontological computationalism? The implications seem difficult to state. Even if a physical system implements multiple computations, not all of these need belong to the class **Cog** of 'cognitive computations' (see Hypothesis 1.2). Moreover, at least *prima facie* there seems to be no reason to exclude the possibility that a physical system could be implementing several computations from **Cog** at the same time, as has been pointed out by Chrisley (1994, 405). The ontology of the mental being as ill-understood as it is, it is not necessarily impossible that a physical system could, at a single instant of time, possess several states of mind (as long as it does not possess *all* states of mind).

However, the observer-relativity of φ-groupings may still pose a threat to ontological computationalism, even if multiple simultaneous mental states are not deemed an impossibility. For, if the selection of a φ-grouping for a physical system is subjective in the sense that groupings do not arise from physical theory, but may be arbitrarily defined by the observer, the system will gain and lose mental states at the observer's whim, according to his or her idiosyncratic ways of "carving Nature". This is an absurd situation, for we would not naturally expect the existence of mental states to be observer-dependent. Rather, we would expect that Nature has to be carved *at her joints*, if meaningful (not to mention true, or truth-approaching) ontological statements are to be made. Therefore, if we accept the relativity of φ-groupings as a substantial property of the relation of implementation, it seems that ontological computationalism will lose much of its attractiveness. If, on the other hand, we wish to retain

our belief in ontological computationalism, we had better find a way of de-relativising φ-groupings. Of course, nothing in our present state of knowledge precludes the *possibility* of de-relativisation. A more sophisticated future theory of implementation may very well be able to tie computations to physical systems in such a way that the observer-relativity of φ-groupings vanishes.

It is therefore unclear what implications, exactly, the issue of implementation has for ontological computationalism. The fact that a system may implement several simultaneous computations need not, in principle, discredit ontological computationalism. On the other hand, it seems that φ-groupings need to be de-relativised, if the ism of ontological computationalism is to retain its modifier 'ontological'. How about instrumental computationalism, the thesis that computational notions play an important instrumental role in cognitive explanation? I shall next argue that the relativity of φ-groupings need not pose any serious threat to this variety of computationalism, and that computational notions are important, if not indispensable, for cognitive explanation, in the present state of inquiry at least.

My ultimate aim in presenting this argument is to suggest that it may be more fruitful to construe implementation as a relation between theories than as a relation between ontologies. Before going into the details, I shall take a moment here to define some key concepts. By 'physical theory', I shall refer collectively to those empirical theories which fall under the heading 'physics'. Physical theory, then, contains such theoretical terms as 'electric charge', 'momentum', 'energy', and so on. By 'empirical theory', in its most general sense, I mean a syntactic-semantic structure which is linked, in one way or another, to "the phenomena". I assume theories to have an amount of empirical content in the sense that there have to be some procedures which link theoretical terms to observational practices. However, not all terms of a theory need have empirical content in this sense. Rather, the meaning of many (if not most) theoretical terms may be determined in a holistic manner

through the meanings of other theoretical terms, or at least it may be the case that "theoretical sentences have their evidence not as single sentences but only as larger blocks of theory" (Quine 1969, 80–81). Regardless of how exactly the semantics of a given theory is fixed in place, it is the semantics that links the theory to reality, and the latter I assume to be the same for all theories. It follows that physics, for example, is not a theory which explains how "physical things" behave, but a theory which explains how things behave from one point of view, the physical point of view (cf. Mach 1897, ch. 1). In other words, substances are not inherently physical—the meaning of the adjective 'physical' is dependent on the structure of those particular theories which we have (due to historical reasons, *inter alia*) come to call *physical* theories.

These remarks do not apply *in toto* to non-empirical or analytical theories, such as those of pure mathematics, whose semantics need not be at all denotative of some outside reality.[1] Computability theory, in particular, is not an empirical theory in our sense. However, concepts developed within computability theory can be—and have been—utilised and applied in certain empirical theories, which is why we label these theories 'computational'. Well-known examples of computational theories in this sense, whether classical or connectionist, include the GPS (Newell & Simon 1963/2000), SHRDLU (Winograd 1973/2000), and NETtalk (Sejnowski & Rosenberg 1987). In what follows, I shall rely more on Marr's (1982) computational theory of vision, but in principle any one of these theories, or models, may be taken as an exemplar of computational explanation in cognitive science. My purpose in the following argument, then, will be to defend the claim that the aim of the theory of implementation is to explicate the relation between 'physical' and 'computational' theories, understood in these senses.

Prima facie, the relativity of φ-groupings does seem to pose a challenge

[1] Unless we embrace Platonism, that is. But even if we did, analytical theories would be denotative of a reality different from the reality of empirical theories, so that a distinction would remain between the two kinds of theory.

to instrumental computationalism, as well. If, given a physical system $\mathcal{S}$, we are free to select a φ-grouping for it *at will* in order to see that it implements some or other computation, does this not imply that this sort of cognitive explanation is thoroughly subjective? Does it not show that the hope of an *objective* computational cognitive science is a forlorn one?

This is precisely the core of Searle's (1992, ch. 9) critique of computational cognitive science. At the end of the day, he says, the problem with computationalism is that "[c]omputational states are not *discovered within* the physics, they are *assigned to* the physics" (Searle 1992, 219). Again,

> [t]he multiple realizability of computationally equivalent processes in different physical media is not just a sign that the processes are abstract, but that they are not intrinsic to the system at all. They depend on an interpretation from outside. We were looking for some facts of the matter that would make brain processes computational; but given the way we have defined computation, there never could be any such facts of the matter. We can't, on the one hand, say that anything is a digital computer if we can assign a syntax to it, and then suppose there is a factual question intrinsic to its physical operation whether or not a natural system such as the brain is a digital computer. (Searle 1992, 209–210, emphasis omitted.)

In other words, computational characterisations do not provide true explanations of the explananda, by Searle's lights:

> As simulations go, the word processing program simulates a typewriter better than any AI program I know of simulates the brain. But no sane person thinks: 'At long last we understand how typewriters work, they are implementations of word processing programs.' It is simply not the case in general that computational simulations provide causal explanations of the phenomena simulated. (Searle 1992, 218.)

Or, more succinctly: "once you understand how the frog's visual system *actually works*, the 'computational level' is just irrelevant" (Searle 1992, 218).

While I would agree with Searle that computational states are not discovered within the physics but are assigned to it, I think a case can be made for the usefulness, perhaps even necessity, of computational theories in cognitive explanation. Computational states are not discovered within the physics, but whom does this really surprise, given the fact that physical theory is not at all concerned with computation, and does not even contain the relevant terminology and theoretical concepts? To form an analogy, states of society (something we might call "social states") are not discovered within the physics either, but this does not imply that social science is "observer-relative", or that the sociological level of explanation is "just irrelevant". This is certainly the case if you are not a reductionist, but even if you are, an eliminativist picture, whereby the upper-level theory is viewed as completely superfluous and reducible to the underlying theory in a straightforward manner, is no longer very attractive (Churchland 1986, ch. 7). In short: if you resolve to reduce present-day sociology, you end up reducing the wrong theory, since present-day sociology is hardly the final word in its domain. If, on the other hand, you admit that the theory to be reduced is some more sophisticated future version of sociology, you effectively concede that the present-day theory cannot be eliminated (that it is not "just irrelevant"), but must be allowed to co-evolve with the reducing theory until a meaningful reduction, not just a blind elimination, can be effected.

Pace Searle, it seems that you cannot understand the visual system, for instance, without making use of some sort of computational theory. "Almost never can a complex system of any kind be understood as a simple extrapolation from the properties of its elementary components" (Marr 1982, 19). To understand *what* the physical object is doing and *why*, it does not suffice to measure frequencies of action potentials, concentrations of potassium ions, or whatever Searle has in mind when speaking of "the physics".[2] Nor do you get

[2]It is to be noted that these things do not, properly speaking, belong to physics, but to physiology. However, if examination of, say, action potentials does not guarantee an under-

a feeling for *how* the object is doing whatever it is doing by focusing on small-scale details. The details are simply too staggering. If computational states are not discovered within the physics, then too bad for the physics, since computational states (or processes) are what you want. And so you *assign* them, hoping that the subsequent evolution of theories on all levels will eventually establish why some assignments are better than others.

Thus, while it may be the case that "once you understand how the frog's visual system *actually works*, the 'computational level' is just irrelevant" (Searle 1992, 218), it appears at least as likely that you will not reach that state of understanding without first doing some computational thinking, and without doing it well, too. It would seem unlikely for anyone to be able to make much sense of the role of the cells of the lateral geniculate nucleus in visual processing, for example, if some preliminary answers had not first been given to such questions as: What is the system doing? and How could it do it? The genius of Marr was to fashion a higher-level theory of vision that addressed precisely these kinds of questions, using concepts borrowed from computability theory. Only then was it possible to give a meaningful and theoretically fruitful interpretation of certain low-level physiological findings (see Marr & Hildreth 1980; Marr 1982, ch. 2) and thereby, we may suppose, to come to understand how the system "actually works".

I have rehearsed these Marrian remarks to such an extent to lend support to a more general call for a more pluralist conception of science, intertheoretic reduction, and the co-evolution of theories. In such a conception, no theory is regarded as metaphysically superior to the rest (although some may be empirically superior, in the sense that they explain more observations than do the others, or do this better in one sense or another). Nor is any theory immune to revisions occasioned not just by facts (observations), but by other theories as well. This is meant to apply both within levels and between levels.

---

standing of the system's large-scale behaviour, then, *a fortiori*, neither does examination of, say, electric fields or atomic structure guarantee such understanding.

In particular, there is no principle—apart from a metaphysical prejudice of a physicalist slant—that renders physical theory immune from criticisms occasioned by some higher-level theory. The currently favoured theory of information processing in the brain, for example, is based on a model of the neuron which is, in all essential respects, similar to the units studied by McCulloch and Pitts (1943/1965). Is it not at least conceivable that major physiological advances may in the future be influenced by higher-level theories, considering that McCulloch and Pitts's (1943/1965) achievement was of precisely this kind (see Piccinini 2004)? Surely the computational level is not irrelevant.[3]

To adopt too reductionist a picture of the scientific enterprise is to miss the important instrumental, heuristic, or propaedeutic value of research conducted on multiple levels, with theories influencing each other both within and between levels. The reductionist picture also tends to lead to *metaphysical chauvinism*: the elevation of some one theory, above the rest, to an ontologically and epistemologically superior position. For present-day materialists, hence for most of those who have contributed to the debate surrounding implementation, this metaphysically superior theory is the physical theory. Consequently, implementation of computation comes to be construed as an asymmetric, irreversible relation. One kind of stuff, somehow ephemeral, is said to be implemented or realised in another kind of stuff, somehow more secure. However, physics is itself an evolving branch of natural science, and not a First Philosophy. Therefore, it may be more fruitful to regard implementation not as a relation between mathematical objects and physical objects, but as a relation between theories, the implication being that a theory of implementation itself cannot be given *a priori*, but must be subjected to continual development as the two theories joined by the relation co-evolve. This does not mean that

[3] A pluralist conception of science such as this can be argued for by means of other historical examples as well. See, e.g., Feyerabend's (1975, chs. 6–12) case study of Aristotelianism and Copernicanism at the time of Galileo. Although Feyerabend (1975) does not address the issue of computational explanation, I am, in my defence of "pluralist cognitive science", much indebted to his "epistemological anarchism".

the theory of implementation cannot be cast in precise, mathematical terms, or that its ultimate aim could not be the formulation of mappings or interpretations which relate computational structures to physical structures. However, these structures themselves have to be suggested by computational *theories*, and by physical *theories*. The concept of physical system, in particular, has no meaning over and above that given it by physical theory, so that an analysis of implementation that relies on entirely armchair-sprung, pre-theoretic notions of "physical systems" is bound to be wide of the mark.

The reductionist picture has led Chalmers (1994a), for instance, to look for a "computational foundation for the study of cognition"—an account of implementation which would secure the fundamental role of computation in cognitive explanation by tying computations to *bona fide* physical stuff. My aim in this final chapter has been to argue that the study of cognition needs no foundation, although it does need what Kuhn (1964) called 'paradigms'. It also needs, or at least can profit from, a rich network of theories on various levels, theories which interact and do not assume immunity from pleas of revision by their fellow theories. The theory of implementation, in particular, is one theory among many. Analyses of implementation, then, are valuable not because they demonstrate how and why "stuff" of one kind is realised in "stuff" of a more fundamental kind, but because they clarify intertheoretic relations and serve the important heuristic, or instrumental, value of conducting research on multiple levels at the same time.

In conclusion, the development of theories of the implementation of computation has been able to dissolve some fears relating to the possible observer-relativity of computation (Searle's Thesis). On the other hand, each account of implementation here studied is incomplete and suffers from some problems. There is, then, motivation for refinement of theory. Finally, analysis of the notion of implementation seems to show that the implications for ontological computationalism are difficult to state, so that the fate of this thesis cannot be

decided by any of the theories of implementation currently available. On the other hand, a relativity in implementation need not threaten the ambitions of instrumental computationalism, which, I have argued, remains a viable attitude to explanation in cognitive science.

A

Proofs

Lemma 2.2 (p. 8)**.** *Let A be a set, finite or infinite. Then A has at least two elements if and only if the countably infinite Cartesian product A^ω is infinite.*

Proof. If A is empty or its cardinality is 1, it is plain that A^ω has only finitely many elements.

Suppose then that $|A| = p$ for some $p \geq 2$. Assume, for a *reductio ad absurdum*, that there exists a $k \in \mathbb{N}$ such that $|A^\omega| = k$. Then by definition k is the number of such functions $f : \mathbb{N} \to A$ that satisfy the condition

$$f(i) \in A \quad \text{for all} \quad i \in \mathbb{N}. \tag{$*$}$$

Because $\mathbb{N}$ is infinite, there is a subset $J = \{j_1, \ldots, j_k\} \subseteq \mathbb{N}$ having k elements. Similarly, by assumption the set A contains at least two elements, so there is a subset $B = \{x, y\} \subseteq A$.

Let us now investigate the number of functions $g : J \to B$ that satisfy

$$g(i) \in B \quad \text{for all} \quad i \in J.$$

Let the set of these functions be G. If $g \in G$, the value of the function g, $g(i)$, for arbitrary $i \in J$, can be either x or y. By elementary combinatorics, the set G thus contains

$$\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{k \text{ times}} = 2^k$$

functions; in other words, $|G| = 2^k$.

Let us next expand each function $g \in G$ to a function $g' : \mathbb{N} \to A$ by setting

$$g'(i) = \begin{cases} g(i), & \text{if } i \in J, \\ x, & \text{if } i \in \mathbb{N} \setminus J. \end{cases}$$

The number of these functions g' is precisely the cardinality of the set G, that is to say, 2^k. On the other hand, each function g' satisfies condition $(*)$, that is to say belongs to the set

$$A^\omega = \{f \mid f : \mathbb{N} \to A \text{ and } f(i) \in A \text{ for all } i \in \mathbb{N}\}.$$

Hence, $|A^\omega| \geq 2^k$. But this is contrary to our assumption that $|A^\omega| = k$, for it can be shown that $k < 2^k$ for all $k \in \mathbb{N}$. Contradiction. Q.E.D.

Lemma 3.1 (p. 16)**.** *For any time instant t, the maximal state $m(t)$ is the only maximal state of $\mathfrak{S}$ compatible with $B(t)$.*

Proof. The strategy is to show that, given the Principle of Noncyclical Behaviour, any other maximal state would lead the system $\mathfrak{S}$ to violate the Principle of Continuity. In outline, the following argument is similar to the one whereby Putnam (1988, 121–125) argues for the claim, though the presentation here is more formal.

Classically, $\mathfrak{S}$ will be embedded in the Euclidean space $\mathbb{R}^3$; let us denote the metric (the distance function) of this space by d. Considered as a spatial object, $\mathfrak{S}$ is then a bounded region of $\mathbb{R}^3$. Let us denote by S (note typeface) the set of interior points of this region, so that S is an open subset of $\mathbb{R}^3$. The boundary of the system is then simply the topological frontier ∂S of S. Let $F_t : S \cup \partial S \to \mathbb{R}^3$ be the gravitational field associated with $\mathfrak{S}$ and its boundary, at time t. In other words, the vector $F_t(x)$ gives the strength of the field at the point x, for all $x \in S \cup \partial S$.

(In fact, nothing in what follows turns on our using the Euclidean space $\mathbb{R}^3$ for the embedding space and for the vector space that is the range of the

gravitational field. The proof will go through without problem with more general vector spaces, or even with metric spaces, although from the point of view of physics the spaces of course have the structure of a vector space.)

Let t' be an instant of time with $t' \neq t$. To show that $m(t')$ is incompatible with $B(t)$, it suffices to show that the mapping $G_t : S \cup \partial S \to \mathbb{R}^3$, defined by

$$G_t(x) = \begin{cases} F_t(x) & \text{if } x \in \partial S \\ F_{t'}(x) & \text{if } x \in S \end{cases},$$

is discontinuous at nondenumerably many points. The mapping G_t, in other words, is the "counterfactual gravitational field" of $\mathfrak{S}$ and its boundary, assuming that the maximal state of the boundary is $B(t)$ and that the maximal state of the system itself is $m(t')$.

By the Principle of Noncyclical Behaviour, the gravitational field is noncyclical at each point of the boundary of $\mathfrak{S}$, as well as at each point of $\mathfrak{S}$ lying sufficiently close to the boundary. Formally: there is an $\epsilon > 0$ such that for each $x \in S \cup \partial S$, if $d(x, \partial S) < \epsilon$, then $F_t(x) \neq F_{t'}(x)$ for all times t, t' with $t \neq t'$. (Here $d(x, \partial S)$ is the distance of the point x from the set ∂S; it is defined as usual as the number $d(x, \partial S) = \inf\{d(x, y) \mid y \in \partial S\}$.) Now let $C_t(\partial S)$ be the set defined by

$$C_t(\partial S) = \{b \in \partial S \mid F_t \text{ is continuous at } b\};$$

by the Principle of Continuity, this set is nondenumerable for all times t. Likewise, for all $b \in \partial S$, define the set $b(\epsilon)$ by putting

$$b(\epsilon) = \{x \in S \mid d(x, \partial S) < \epsilon\}.$$

Now let $b \in C_{t'}(\partial S)$ and $x \in b(\epsilon)$ be arbitrary. Then $F_{t'}$ is continuous at b,

so that

$$\lim_{\substack{x \to b \\ x \in b(\epsilon)}} F_{t'}(x) = F_{t'}(b).$$

Then, since $G_t(x) = F_{t'}(x)$ for all $x \in S$ and hence for all $x \in b(\epsilon)$, we have

$$\lim_{\substack{x \to b \\ x \in b(\epsilon)}} G_t(x) = F_{t'}(b).$$

However, by the Principle of Noncyclical Behaviour $F_{t'}(b) \neq F_t(b)$, so that

$$\lim_{\substack{x \to b \\ x \in b(\epsilon)}} G_t(x) \neq G_t(b),$$

since $G_t(b) = F_t(b)$. Hence, G_t is not continuous at b. Since $b \in C_{t'}(\partial S)$ was chosen arbitrarily and since $C_{t'}(\partial S)$ is nondenumerable, we have shown that G_t is discontinuous at nondenumerably many points. Thus, $\mathbb{S}$ will violate the Principle of Continuity in its gravitational field if $m(t')$ is substituted for $m(t)$. Q.E.D.

References

Abian, A. (1965). *The theory of sets and transfinite arithmetic*. Philadelphia: Saunders.

Arnol'd, V. I. (1992). *Ordinary differential equations* (R. Cooke, Trans.). Berlin: Springer.

Brown, C. (2004). Implementation and indeterminacy. In J. Weckert & Y. Al-Saggaf (Eds.), *Conferences in research and practice in information technology* (Vol. 37). Retrieved May 2007, from `http://www.crpit.com/confpapers/CRPITV37Brown.pdf`.

Burton, D. M. (1967). *Introduction to modern abstract algebra*. Reading, MA: Addison-Wesley.

Carnap, R. (1958). *Introduction to symbolic logic and its applications* (W. H. Meyer & J. Wilkinson, Trans.). New York, NY: Dover.

Chalmers, D. J. (1994a). *A computational foundation for the study of cognition.* Retrieved May 2007, from `http://consc.net/ai-papers.html`. Unpublished manuscript.

Chalmers, D. J. (1994b). On implementing a computation. *Minds and Machines*, 4, 391–402.

Chalmers, D. J. (1996). Does a rock implement every finite-state automaton? *Synthese*, 108, 309–333.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2 (3), 113–124.

Chrisley, R. L. (1994). Why everything doesn't realize every computation. *Minds and Machines*, 4, 403–420.

Church, A. (1936/1965). An unsolvable problem of elementary number theory.

In M. Davis (Ed.), *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions* (pp. 89–107). Hewlett, NY: Raven Press.

Churchland, P. S. (1986). *Neurophilosophy: Toward a unified science of the mind-brain*. Cambridge, MA: MIT Press.

Cocos, C. (2002). Computational processes: A reply to Chalmers and Copeland. *Sats — Nordic Journal of Philosophy*, 3, 25–49.

Copeland, B. J. (1996). What is computation? *Synthese*, 108, 335–359.

Copeland, B. J. (2002). Hypercomputation. *Minds and Machines*, 12, 461–502.

Feyerabend, P. (1975). *Against method*. London: NLB.

Fodor, J. & McLaughlin, B. P. (1990). Connectionism and the problem of systematicity: Why Smolensky's solution doesn't work. *Cognition*, 35, 183–204.

Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28, 3–71.

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

Kuhn, T. S. (1964). *The structure of scientific revolutions*. Chicago, IL: University of Chicago Press.

Mach, E. (1897). *Contributions to the analysis of the sensations* (C. M. Williams, Trans.). Chicago, IL: Open Court.

Marr, D. (1977). Artificial intelligence—a personal view. *Artificial Intelligence*, 9, 37–48.

Marr, D. (1982). *Vision*. New York, NY: Freeman.

Marr, D. & Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London B*, 207, 187–217.

McCulloch, W. S. & Pitts, W. H. (1943/1965). A logical calculus of the ideas immanent in nervous activity. In W. S. McCulloch, *Embodiments of mind*

(pp. 19–39). Cambridge, MA: MIT Press.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, 4, 135–183.

Newell, A. & Simon, H. A. (1963/2000). GPS, a program that simulates human thought. In R. Cummins & D. D. Cummins (Eds.), *Minds, brains, and computers: The foundations of cognitive science: An anthology* (pp. 84–94). Malden, MA: Blackwell.

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

Piccinini, G. (2004). The first computational theory of mind and brain: A close look at McCulloch and Pitts's "Logical calculus of ideas immanent in nervous activity". *Synthese*, 141, 175–215.

Putnam, H. (1960/1975). Minds and machines. In *Mind, language and reality: Philosophical papers* (Vol. 2, pp. 362–385). Cambridge: Cambridge University Press.

Putnam, H. (1988). *Representation and reality*. Cambridge, MA: MIT Press.

Quine, W. V. (1969). Epistemology naturalized. In *Ontological relativity and other essays* (pp. 69–90). New York, NY: Columbia University Press.

Scheutz, M. (1999). When physical systems realize functions. *Minds and Machines*, 9, 161–196.

Scheutz, M. (2001). Computational versus causal complexity. *Minds and Machines*, 11, 543–566.

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3, 417–424.

Searle, J. R. (1990). Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association*, 64, 21–37. Retrieved May 2007, from `http://users.ecs.soton.ac.uk/harnad/Papers/Py104/searle.comp.html`.

Searle, J. R. (1992). *The rediscovery of the mind*. Cambridge, MA: MIT Press.

Sejnowski, T. J. & Rosenberg, C. R. (1987). Parallel networks that learn to

pronounce English text. *Complex Systems*, 1, 145–168.

Shagrir, O. (2005). The rise and fall of computational functionalism. In Y. Ben-Menahem (Ed.), *Hilary Putnam* (pp. 220–250). Cambridge: Cambridge University Press.

Siegelmann, H. T. & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, 50, 132–150.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11, 1–23.

Turing, A. M. (1936–7/1965). On computable numbers, with an application to the Entscheidungsproblem. In M. Davis (Ed.), *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions* (pp. 116–151). Hewlett, NY: Raven Press.

van Gelder, T. (1998). The dynamical hypothesis in cognitive science. *Behavioral and Brain Sciences*, 21, 615–628.

Väänänen, J. (1987). *Matemaattinen logiikka* [Mathematical logic]. Helsinki: Gaudeamus.

Winograd, T. (1973/2000). A procedural model of language understanding. In R. Cummins & D. D. Cummins (Eds.), *Minds, brains, and computers: The foundations of cognitive science: An anthology* (pp. 95–113). Malden, MA: Blackwell.